

toc,lof,lot

Matplotlib

Release 2.2.4

John Hunter, Darren Dale, Eric Firing, Michael Droettboom and the matplotlib

February 28, 2019

CONTENTS

Part I

User's Guide

HISTORY

Note: The following introductory text was written in 2008 by John D. Hunter (1968-2012), the original author of Matplotlib.

Matplotlib is a library for making 2D plots of arrays in [Python](#). Although it has its origins in emulating the MATLAB graphics commands, it is independent of MATLAB, and can be used in a Pythonic, object oriented way. Although Matplotlib is written primarily in pure Python, it makes heavy use of [NumPy](#) and other extension code to provide good performance even for large arrays.

Matplotlib is designed with the philosophy that you should be able to create simple plots with just a few commands, or just one! If you want to see a histogram of your data, you shouldn't need to instantiate objects, call methods, set properties, and so on; it should just work.

For years, I used to use MATLAB exclusively for data analysis and visualization. MATLAB excels at making nice looking plots easy. When I began working with EEG data, I found that I needed to write applications to interact with my data, and developed an EEG analysis application in MATLAB. As the application grew in complexity, interacting with databases, http servers, manipulating complex data structures, I began to strain against the limitations of MATLAB as a programming language, and decided to start over in Python. Python more than makes up for all of MATLAB's deficiencies as a programming language, but I was having difficulty finding a 2D plotting package (for 3D [VTK](#) more than exceeds all of my needs).

When I went searching for a Python plotting package, I had several requirements:

- Plots should look great - publication quality. One important requirement for me is that the text looks good (antialiased, etc.)
- Postscript output for inclusion with TeX documents
- Embeddable in a graphical user interface for application development
- Code should be easy enough that I can understand it and extend it
- Making plots should be easy

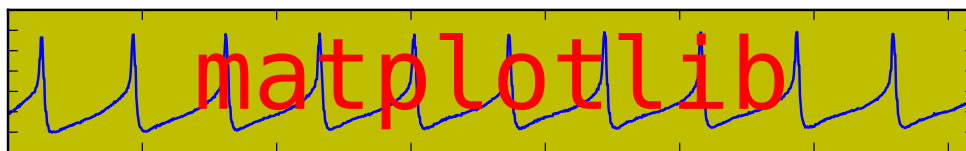
Finding no package that suited me just right, I did what any self-respecting Python programmer would do: rolled up my sleeves and dived in. Not having any real experience with computer graphics, I decided to emulate MATLAB's plotting capabilities because that is something MATLAB does very well. This had the added advantage that many people have a lot of MATLAB experience, and thus they can quickly get up to steam plotting in python. From a developer's perspective, having

a fixed user interface (the pylab interface) has been very useful, because the guts of the code base can be redesigned without affecting user code.

The Matplotlib code is conceptually divided into three parts: the pylab interface is the set of functions provided by `matplotlib.pylab` which allow the user to create plots with code quite similar to MATLAB figure generating code ([Pyplot tutorial](#)). The Matplotlib frontend or Matplotlib API is the set of classes that do the heavy lifting, creating and managing figures, text, lines, plots and so on ([Artist tutorial](#)). This is an abstract interface that knows nothing about output. The backends are device-dependent drawing devices, aka renderers, that transform the frontend representation to hardcopy or a display device ([What is a backend?](#)). Example backends: PS creates [PostScript®](#) hardcopy, SVG creates [Scalable Vector Graphics](#) hardcopy, Agg creates PNG output using the high quality [Anti-Grain Geometry](#) library that ships with Matplotlib, GTK embeds Matplotlib in a [Gtk+](#) application, GTKAgg uses the Anti-Grain renderer to create a figure and embed it in a [Gtk+](#) application, and so on for [PDF](#), [WxWidgets](#), [Tkinter](#), etc.

Matplotlib is used by many people in many different contexts. Some people want to automatically generate PostScript files to send to a printer or publishers. Others deploy Matplotlib on a web application server to generate PNG output for inclusion in dynamically-generated web pages. Some use Matplotlib interactively from the Python shell in Tkinter on Windows. My primary use is to embed Matplotlib in a [Gtk+](#) EEG application that runs on Windows, Linux and Macintosh OS X.

Matplotlib's original logo (2003 – 2008).



INSTALLING

Note: If you wish to contribute to the project, it's recommended you [install the latest development version](#).

Contents

- Installing
 - Installing an official release
 - * Windows
 - * macOS
 - * Linux
 - * Test Data
 - Third-party distributions of Matplotlib
 - * Scientific Python Distributions
 - * Linux : using your package manager
 - Installing from source
 - * Dependencies
 - * Building on Linux
 - * Building on macOS
 - * Building on Windows
 - Wheel builds using conda packages
 - Conda packages

2.1 Installing an official release

Matplotlib and most of its dependencies are all available as wheel packages for macOS, Windows and Linux distributions:

```
python -mpip install -U pip
python -mpip install -U matplotlib
```

Note: The following backends work out of the box: Agg, ps, pdf, svg and TkAgg.

For support of other GUI frameworks, LaTeX rendering, saving animations and a larger selection of file formats, you may need to install [additional dependencies](#).

Although not required, we suggest also installing IPython for interactive use. To easily install a complete Scientific Python stack, see [Scientific Python Distributions](#) below.

2.1.1 Windows

In case Python 2.7 or 3.4 are not installed for all users, the Microsoft Visual C++ 2008 ([64 bit](#) or [32 bit](#) for Python 2.7) or Microsoft Visual C++ 2010 ([64 bit](#) or [32 bit](#) for Python 3.4) redistributable packages need to be installed.

2.1.2 macOS

If you are using Python 2.7 on a Mac you may need to do:

```
xcode-select --install
```

so that subprocess32, a dependency, may be compiled.

To use the native OSX backend you will need a framework build of Python.

2.1.3 Linux

On extremely old versions of Linux and Python 2.7 you may need to install the master version of subprocess32 (see [comments](#)).

2.1.4 Test Data

The wheels (*.whl) on the [PyPI download page](#) do not contain test data or example code. If you want to try the many demos that come in the Matplotlib source distribution, download the *.tar.gz file and look in the examples subdirectory. To run the test suite:

- extract the lib/matplotlib/tests or lib/mpl_toolkits/tests directories from the source distribution;

- install test dependencies: [pytest](#), [mock](#), Pillow, MiKTeX, GhostScript, ffmpeg, avconv, ImageMagick, and [Inkscape](#);
- run `py.test path\to\tests\directory`.

2.2 Third-party distributions of Matplotlib

2.2.1 Scientific Python Distributions

[Anaconda](#) and [Canopy](#) and [ActiveState](#) are excellent choices that "just work" out of the box for Windows, macOS and common Linux platforms. [WinPython](#) is an option for windows users. All of these distributions include Matplotlib and lots of other useful (data) science tools.

2.2.2 Linux : using your package manager

If you are on Linux, you might prefer to use your package manager. Matplotlib is packaged for almost every major Linux distribution.

- Debian / Ubuntu: `sudo apt-get install python3-matplotlib`
- Fedora: `sudo dnf install python3-matplotlib`
- Red Hat: `sudo yum install python3-matplotlib`
- Arch: `sudo pacman -S python-matplotlib`

2.3 Installing from source

If you are interested in contributing to Matplotlib development, running the latest source code, or just like to build everything yourself, it is not difficult to build Matplotlib from source. Grab the latest tar.gz release file from [the PyPI files page](#), or if you want to develop Matplotlib or just need the latest bugfixed version, grab the latest git version Install from source.

The standard environment variables CC, CXX, PKG_CONFIG are respected. This means you can set them if your toolchain is prefixed. This may be used for cross compiling.

```
export CC=x86_64-pc-linux-gnu-gcc
export CXX=x86_64-pc-linux-gnu-g++
export PKG_CONFIG=x86_64-pc-linux-gnu-pkg-config
```

Once you have satisfied the requirements detailed below (mainly Python, NumPy, libpng and FreeType), you can build Matplotlib.

```
cd matplotlib
python -mpip install .
```


We provide a `setup.cfg` file which you can use to customize the build process. For example, which default backend to use, whether some of the optional libraries that Matplotlib ships with are installed, and so on. This file will be particularly useful to those packaging Matplotlib.

If you have installed prerequisites to nonstandard places and need to inform Matplotlib where they are, edit `setuptools.py` and add the base dirs to the `basedir` dictionary entry for your `sys.platform`; e.g., if the header of some required library is in `/some/path/include/someheader.h`, put `/some/path` in the `basedir` list for your platform.

2.3.1 Dependencies

Matplotlib requires a large number of dependencies:

- `Python` (`>= 2.7` or `>= 3.4`)
- `NumPy` (`>= 1.7.1`)
- `setuptools`
- `dateutil` (`>= 2.1`)
- `pyarsing`
- `libpng` (`>= 1.2`)
- `pytz`
- `FreeType` (`>= 2.3`)
- `cycler` (`>= 0.10.0`)
- `six`
- `backports.functools_lru_cache` (for Python 2.7 only)
- `subprocess32` (for Python 2.7 only, on Linux and macOS only)
- `kiwisolver` (`>= 1.0.0`)

Optionally, you can also install a number of packages to enable better user interface toolkits. See [What is a backend?](#) for more details on the optional Matplotlib backends and the capabilities they provide.

- `tk` (`>= 8.3`, `!= 8.6.0` or `8.6.1`): for the Tk-based backends;
- `PyQt4` (`>= 4.6`) or `PySide`: for the Qt4-based backend;
- `PyQt5`: for the Qt5-based backend;
- `pygtk` (`>= 2.4`): for the GTK and the GTKAgg backend;
- `wxpython` (`>= 2.9` or later): for the WX or WXAgg backend;
- `cairoffi` (`>= v0.8`): for cairo based backends;
- `pycairo`: for GTK3Cairo;
- `Tornado`: for the WebAgg backend;

For better support of animation output format and image file formats, LaTeX, etc., you can install the following:

- [ffmpeg/avconv](#): for saving movies;
- [ImageMagick](#): for saving animated gifs;
- [Pillow](#) (≥ 2.0): for a larger selection of image file formats: JPEG, BMP, and TIFF image files;
- [LaTeX](#) and [GhostScript](#) (for rendering text with LaTeX).

Note: Matplotlib depends on a large number of non-Python libraries. [pkg-config](#) can be used to find required non-Python libraries and thus make the install go more smoothly if the libraries and headers are not in the expected locations.

Note: The following libraries are shipped with Matplotlib:

- Agg: the Anti-Grain Geometry C++ rendering engine;
 - qhull: to compute Delaunay triangulation;
 - ttconv: a true type font utility.
-

2.3.2 Building on Linux

It is easiest to use your system package manager to install the dependencies.

If you are on Debian/Ubuntu, you can get all the dependencies required to build Matplotlib with:

```
sudo apt-get build-dep python-matplotlib
```

If you are on Fedora, you can get all the dependencies required to build Matplotlib with:

```
sudo dnf builddep python-matplotlib
```

If you are on RedHat, you can get all the dependencies required to build Matplotlib by first installing yum-builddep and then running:

```
su -c "yum-builddep python-matplotlib"
```

These commands do not build Matplotlib, but instead get and install the build dependencies, which will make building from source easier.

2.3.3 Building on macOS

The build situation on macOS is complicated by the various places one can get the libpng and FreeType requirements (MacPorts, Fink, /usr/X11R6), the different architectures (e.g., x86, ppc,

universal), and the different macOS versions (e.g., 10.4 and 10.5). We recommend that you build the way we do for the macOS release: get the source from the tarball or the git repository and install the required dependencies through a third-party package manager. Two widely used package managers are Homebrew, and MacPorts. The following example illustrates how to install libpng and FreeType using brew:

```
brew install libpng freetype pkg-config
```

If you are using MacPorts, execute the following instead:

```
port install libpng freetype pkgconfig
```

After installing the above requirements, install Matplotlib from source by executing:

```
python -mpip install .
```

Note that your environment is somewhat important. Some conda users have found that, to run the tests, their PYTHONPATH must include `/path/to/anaconda/.../site-packages` and their DYLD_FALLBACK_LIBRARY_PATH must include `/path/to/anaconda/lib`.

2.3.4 Building on Windows

The Python shipped from <https://www.python.org> is compiled with Visual Studio 2008 for versions before 3.3, Visual Studio 2010 for 3.3 and 3.4, and Visual Studio 2015 for 3.5 and 3.6. Python extensions are recommended to be compiled with the same compiler.

Since there is no canonical Windows package manager, the methods for building FreeType, zlib, and libpng from source code are documented as a build script at [matplotlib-winbuild](#).

There are a few possibilities to build Matplotlib on Windows:

- Wheels via [matplotlib-winbuild](#)
- Wheels by using conda packages
- Conda packages

Wheel builds using conda packages

This is a wheel build, but we use conda packages to get all the requirements. The binary requirements (png, FreeType,...) are statically linked and therefore not needed during the wheel install.

The commands below assume that you can compile a native Python lib for the Python version of your choice. See [this howto](#) for how to install and setup such environments. If in doubt: use Python `>= 3.5` as it mostly works without fiddling with environment variables:

```
# create a new environment with the required packages
conda create -n "matplotlib_build" python=3.5 numpy python-dateutil pyparsing pytz tornado
↪ "cycler>=0.10" tk libpng zlib freetype
```

(continues on next page)

(continued from previous page)

```

activate matplotlib_build
# if you want a qt backend, you also have to install pyqt (be aware that pyqt doesn't mix well if
# you have created the environment with conda-forge already activated...)
conda install pyqt
# this package is only available in the conda-forge channel
conda install -c conda-forge msinttypes
# for Python 2.7
conda install -c conda-forge backports.functools_lru_cache

# copy the libs which have "wrong" names
set LIBRARY_LIB=%CONDA_PREFIX%\Library\lib
mkdir lib || cmd /c "exit /b 0"
copy %LIBRARY_LIB%\zlibstatic.lib lib\z.lib
copy %LIBRARY_LIB%\libpng_static.lib lib\png.lib

# Make the header files and the rest of the static libs available during the build
# CONDA_DEFAULT_ENV is a env variable which is set to the currently active environment
↪path
set MPLBASEDIRLIST=%CONDA_PREFIX%\Library\;.

# build the wheel
python setup.py bdist_wheel

```

The build_alllocal.cmd script in the root folder automates these steps if you have already created and activated the conda environment.

Conda packages

This needs a [working installed C compiler](#) for the version of Python you are compiling the package for but you don't need to setup the environment variables:

```

# only the first time...
conda install conda-build

# the Python version you want a package for...
set CONDA_PY=3.5

# builds the package, using a clean build environment
conda build ci\conda_recipe

# install the new package
conda install --use-local matplotlib

```


TUTORIALS

This page contains more in-depth guides for using Matplotlib. It is broken up into beginner, intermediate, and advanced sections, as well as sections covering specific topics.

For shorter examples, see our [examples page](#). You can also find [external resources](#) and a [FAQ](#) in our [user guide](#).

3.1 Introductory

These tutorials cover the basics of creating visualizations with Matplotlib, as well as some best-practices in using the package effectively.

Note: [Click here to download the full example code](#)

3.1.1 Sample plots in Matplotlib

Here you'll find a host of example plots with the code that generated them.

Line Plot

Here's how to create a line plot with text labels using `plot()`.

Multiple subplots in one figure

Multiple axes (i.e. subplots) are created with the `subplot()` function:

Images

Matplotlib can display images (assuming equally spaced horizontal dimensions) using the `imshow()` function.

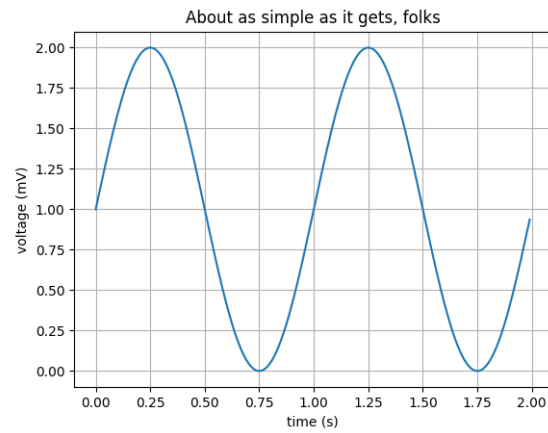


Fig. 1: Simple Plot

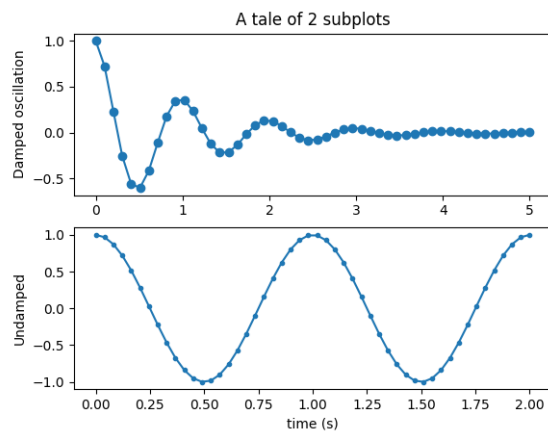
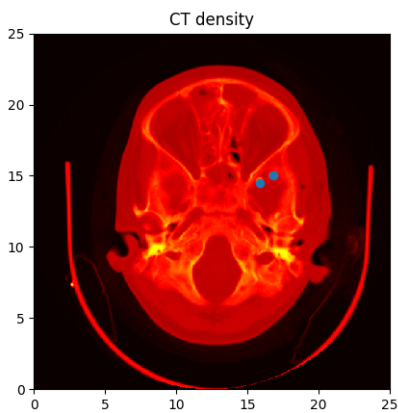


Fig. 2: Subplot

Fig. 3: Example of using `imshow()` to display a CT scan

Contouring and pseudocolor

The `pcolormesh()` function can make a colored representation of a two-dimensional array, even if the horizontal dimensions are unevenly spaced. The `contour()` function is another way to represent the same data:

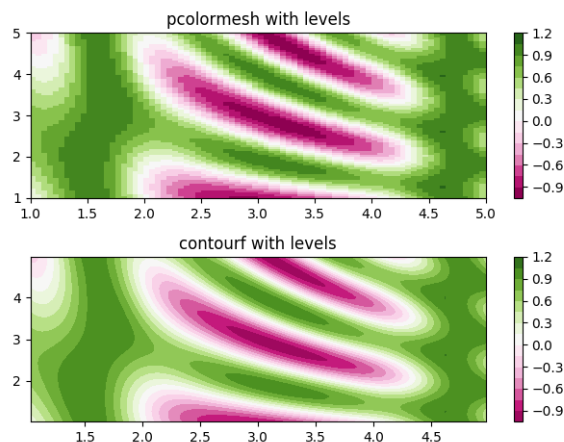


Fig. 4: Example comparing `pcolormesh()` and `contour()` for plotting two-dimensional data

Histograms

The `hist()` function automatically generates histograms and returns the bin counts or probabilities:

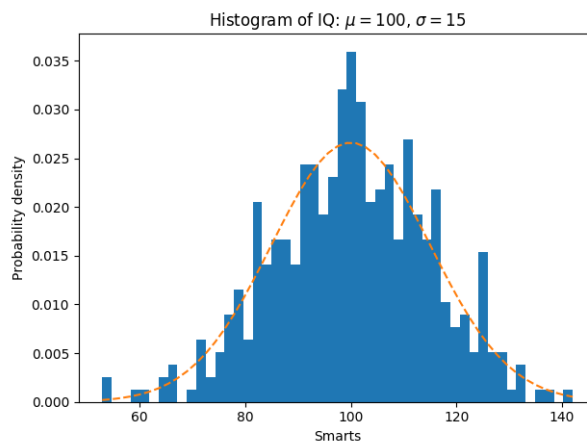


Fig. 5: Histogram Features

Paths

You can add arbitrary paths in Matplotlib using the `matplotlib.path` module:

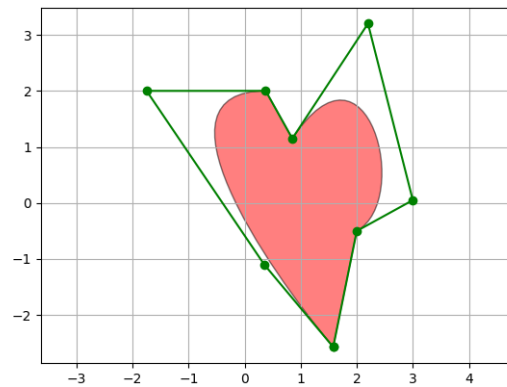


Fig. 6: Path Patch

Three-dimensional plotting

The `mplot3d` toolkit (see [Getting started](#) and [mplot3d-examples-index](#)) has support for simple 3d graphs including surface, wireframe, scatter, and bar charts.

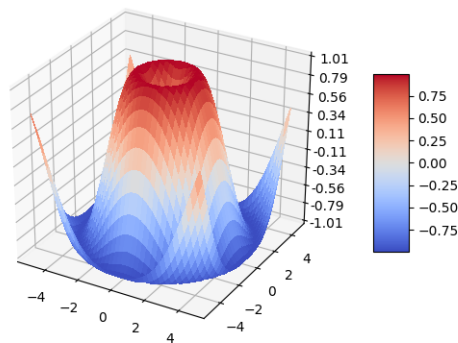


Fig. 7: Surface3d

Thanks to John Porter, Jonathon Taylor, Reinier Heeres, and Ben Root for the `mplot3d` toolkit. This toolkit is included with all standard Matplotlib installs.

Streamplot

The `streamplot()` function plots the streamlines of a vector field. In addition to simply plotting the streamlines, it allows you to map the colors and/or line widths of streamlines to a separate parameter, such as the speed or local intensity of the vector field.

This feature complements the `quiver()` function for plotting vector fields. Thanks to Tom Flanagan and Tony Yu for adding the `streamplot` function.

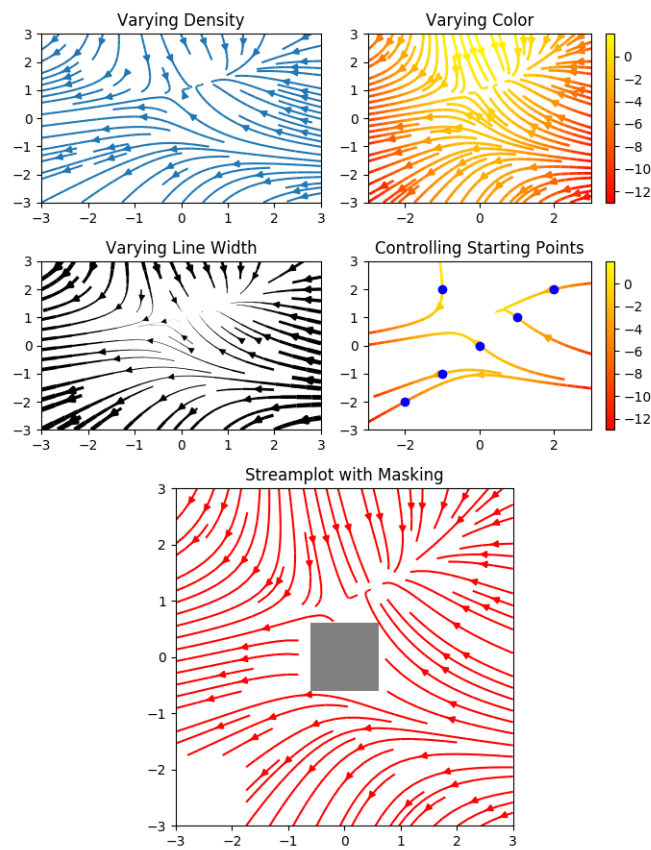


Fig. 8: Streamplot with various plotting options.

Ellipses

In support of the [Phoenix](#) mission to Mars (which used Matplotlib to display ground tracking of spacecraft), Michael Droettboom built on work by Charlie Moad to provide an extremely accurate 8-spline approximation to elliptical arcs (see [Arc](#)), which are insensitive to zoom level.

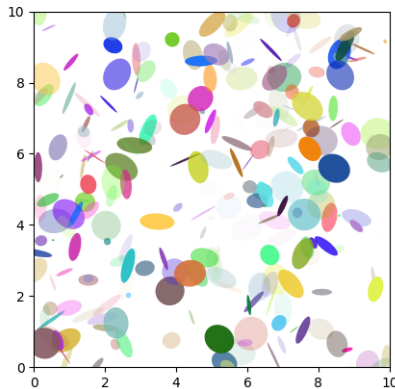


Fig. 9: Ellipse Demo

Bar charts

Use the `bar()` function to make bar charts, which includes customizations such as error bars:

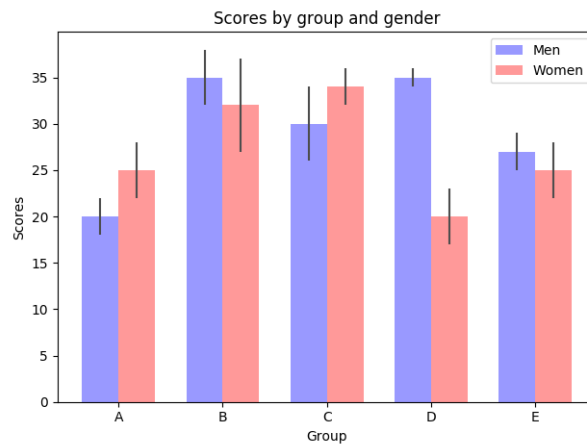


Fig. 10: Barchart Demo

You can also create stacked bars ([bar_stacked.py](#)), or horizontal bar charts ([barh.py](#)).

Pie charts

The `pie()` function allows you to create pie charts. Optional features include auto-labeling the percentage of area, exploding one or more wedges from the center of the pie, and a shadow effect.

Take a close look at the attached code, which generates this figure in just a few lines of code.

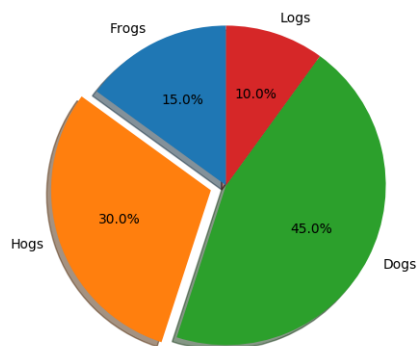


Fig. 11: Pie Features

Tables

The `table()` function adds a text table to an axes.

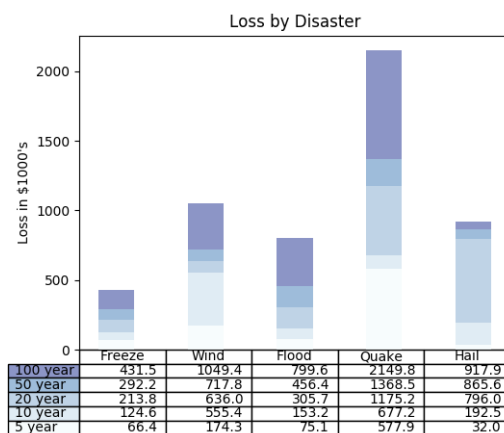


Fig. 12: Table Demo

Scatter plots

The `scatter()` function makes a scatter plot with (optional) size and color arguments. This example plots changes in Google's stock price, with marker sizes reflecting the trading volume and colors varying with time. Here, the `alpha` attribute is used to make semitransparent circle markers.

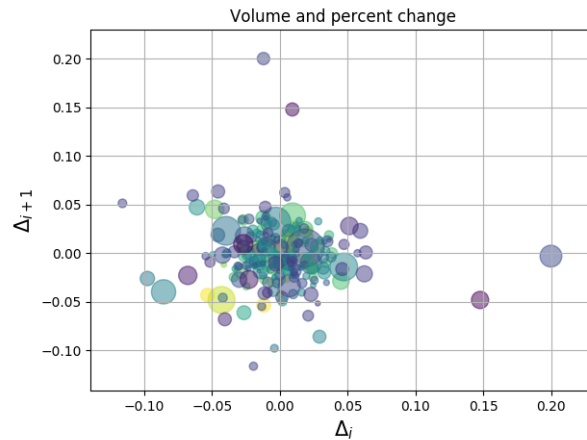


Fig. 13: Scatter Demo2

GUI widgets

Matplotlib has basic GUI widgets that are independent of the graphical user interface you are using, allowing you to write cross GUI figures and widgets. See [matplotlib.widgets](#) and the [widget examples](#).

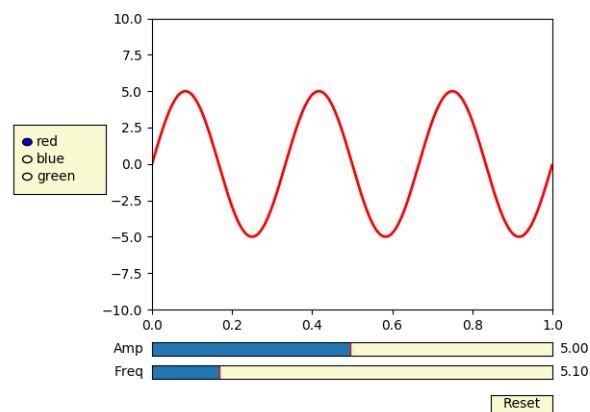


Fig. 14: Slider and radio-button GUI.

Filled curves

The `fill()` function lets you plot filled curves and polygons:

Thanks to Andrew Straw for adding this function.

Date handling

You can plot timeseries data with major and minor ticks and custom tick formatters for both.

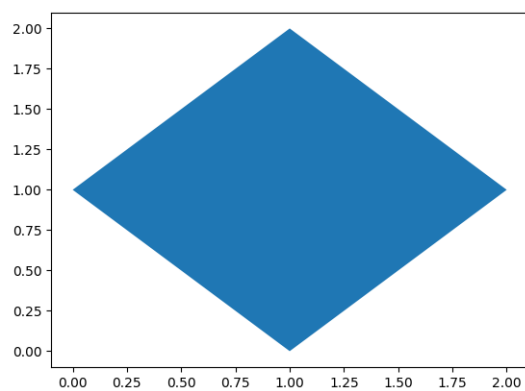


Fig. 15: Fill

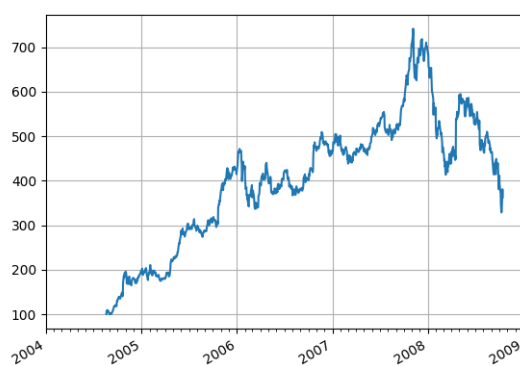


Fig. 16: Date

See `matplotlib.ticker` and `matplotlib.dates` for details and usage.

Log plots

The `semilogx()`, `semilogy()` and `loglog()` functions simplify the creation of logarithmic plots.

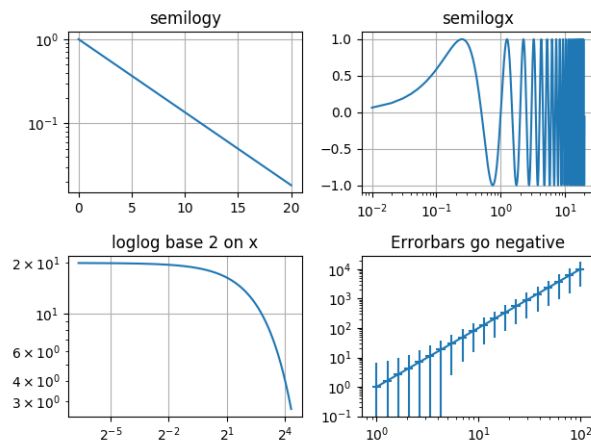


Fig. 17: Log Demo

Thanks to Andrew Straw, Darren Dale and Gregory Lielens for contributions log-scaling infrastructure.

Polar plots

The `polar()` function generates polar plots.

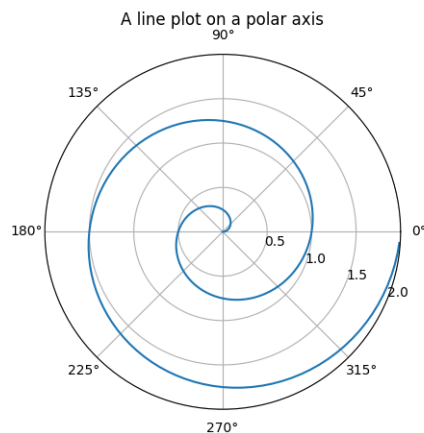


Fig. 18: Polar Demo

Legends

The `legend()` function automatically generates figure legends, with MATLAB-compatible legend-placement functions.

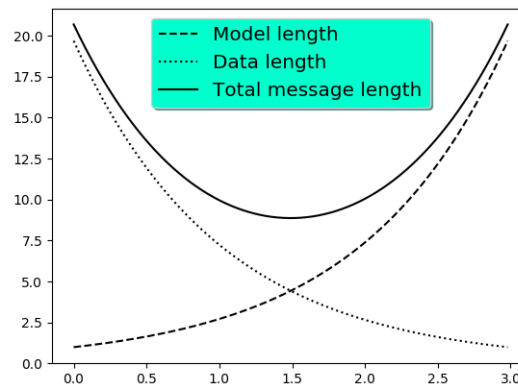


Fig. 19: Legend

Thanks to Charles Twardy for input on the legend function.

TeX-notation for text objects

Below is a sampling of the many TeX expressions now supported by Matplotlib's internal `mathtext` engine. The `mathtext` module provides TeX style mathematical expressions using [FreeType](#) and the DejaVu, BaKoMa computer modern, or [STIX](#) fonts. See the `matplotlib.mathtext` module for additional details.

Matplotlib's `mathtext` infrastructure is an independent implementation and does not require TeX or any external packages installed on your computer. See the tutorial at [Writing mathematical expressions](#).

Native TeX rendering

Although Matplotlib's internal math rendering engine is quite powerful, sometimes you need TeX. Matplotlib supports external TeX rendering of strings with the `usetex` option.

EEG GUI

You can embed Matplotlib into pygtk, wx, Tk, or Qt applications. Here is a screenshot of an EEG viewer called `pbrain`.

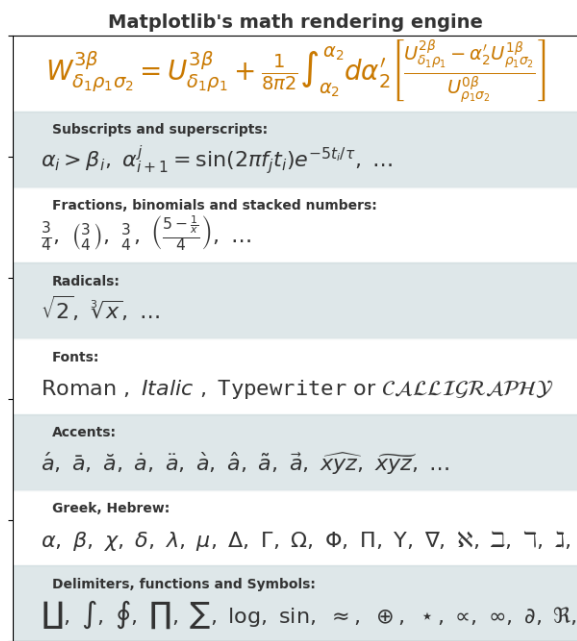


Fig. 20: Mathtext Examples

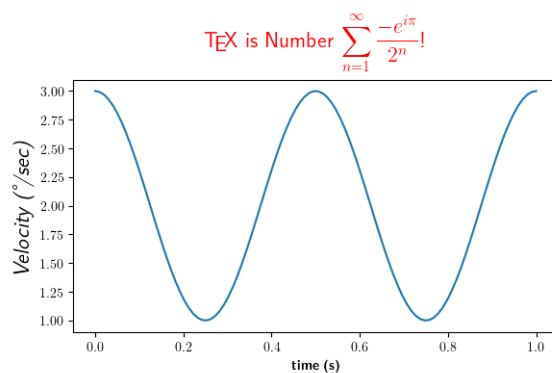
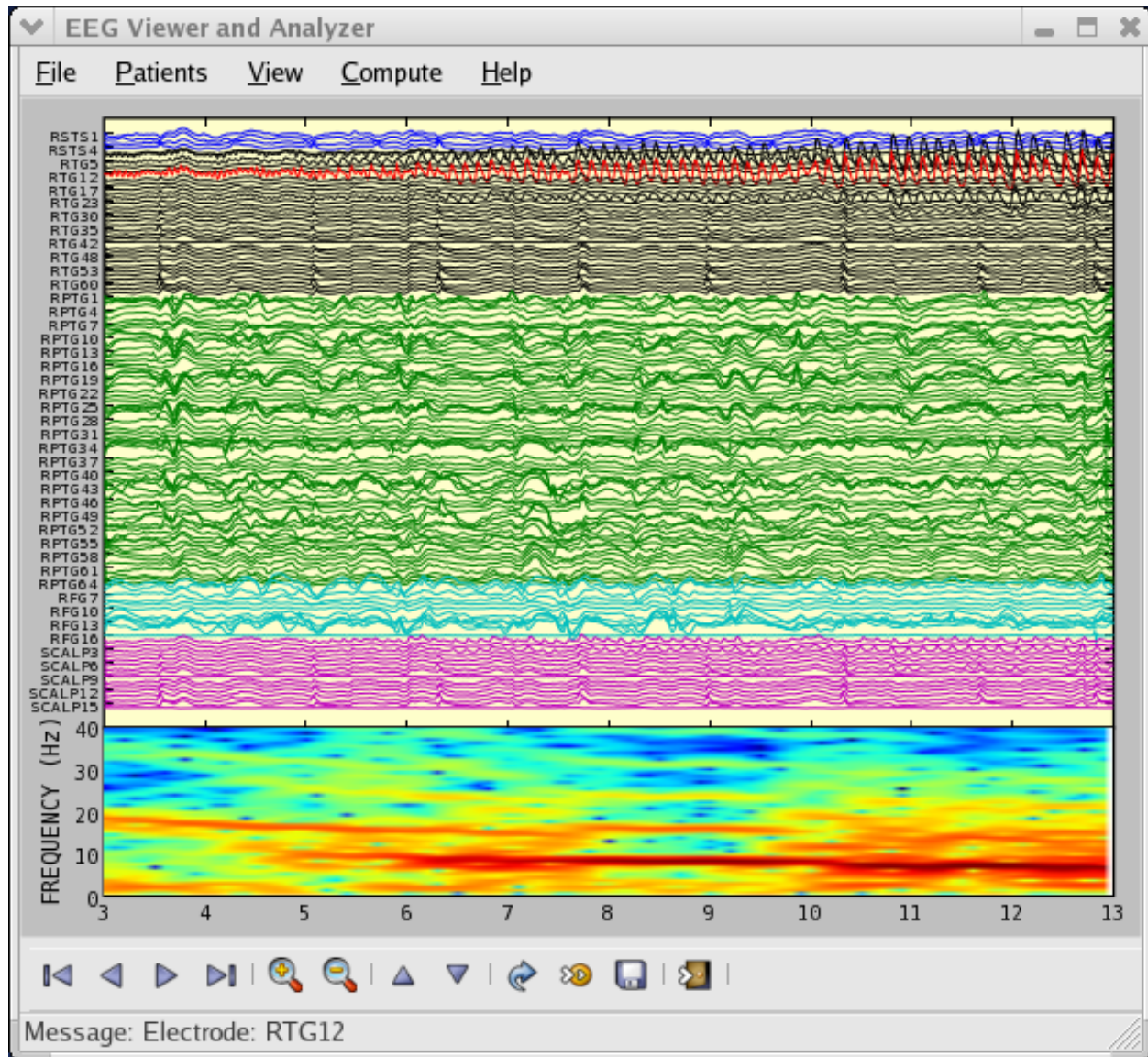


Fig. 21: Tex Demo



The lower axes uses `specgram()` to plot the spectrogram of one of the EEG channels.

For examples of how to embed Matplotlib in different toolkits, see:

- [/gallery/user_interfaces/embedding_in_gtk2_sgskip](#)
- [/gallery/user_interfaces/embedding_in_wx2_sgskip](#)
- [/gallery/user_interfaces/mpl_with_glade_sgskip](#)
- [/gallery/user_interfaces/embedding_in_qt_sgskip](#)
- [/gallery/user_interfaces/embedding_in_tk_sgskip](#)

XKCD-style sketch plots

Just for fun, Matplotlib supports plotting in the style of `xkcd`.

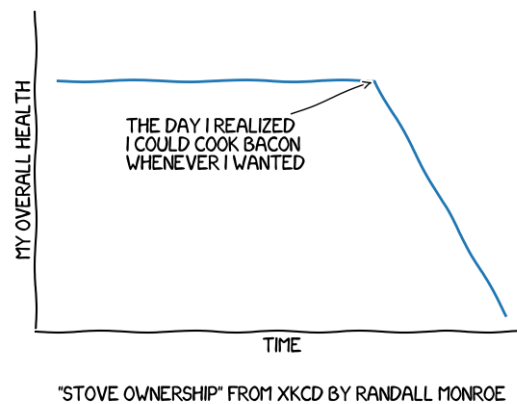


Fig. 22: Xkcd

Subplot example

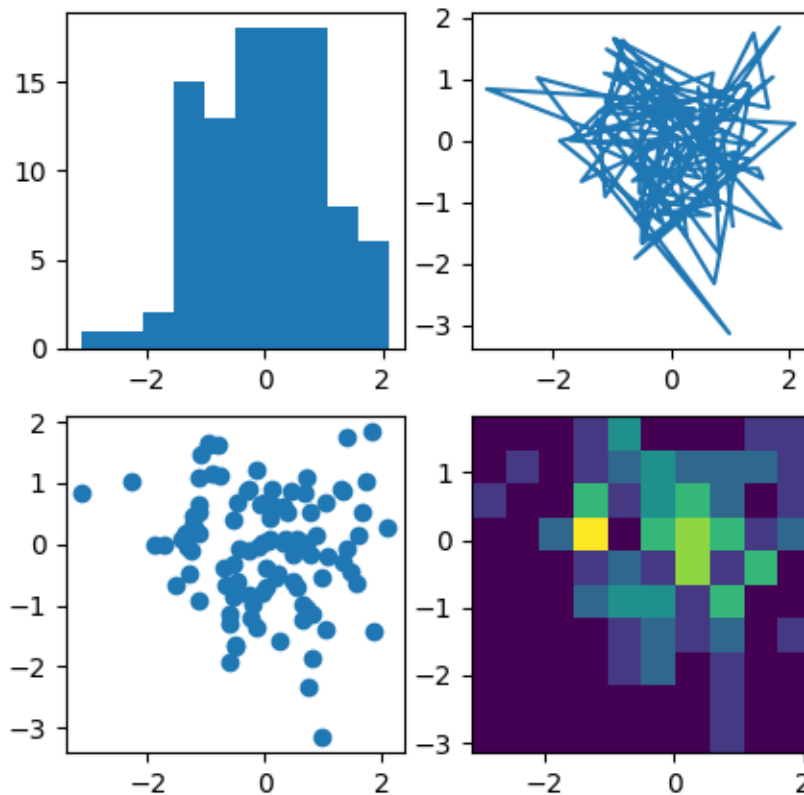
Many plot types can be combined in one figure to create powerful and flexible representations of data.

```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(19680801)
data = np.random.randn(2, 100)

fig, axs = plt.subplots(2, 2, figsize=(5, 5))
axs[0, 0].hist(data[0])
axs[1, 0].scatter(data[0], data[1])
axs[0, 1].plot(data[0], data[1])
axs[1, 1].hist2d(data[0], data[1])

plt.show()
```



Note: [Click here to download the full example code](#)

3.1.2 Customizing Matplotlib with style sheets and rcParams

Tips for customizing the properties and default styles of Matplotlib.

Using style sheets

The style package adds support for easy-to-switch plotting "styles" with the same parameters as a `matplotlibrc` file (which is read at startup to configure matplotlib).

There are a number of pre-defined styles provided by Matplotlib. For example, there's a pre-defined style called "ggplot", which emulates the aesthetics of `ggplot` (a popular plotting package for R). To use this style, just add:

```
import numpy as np
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
import matplotlib as mpl
plt.style.use('ggplot')
data = np.random.randn(50)
```

To list all available styles, use:

```
print(plt.style.available)
```

Out:

```
['seaborn-paper', 'seaborn-poster', 'seaborn-darkgrid', 'fast', 'ggplot', 'seaborn-pastel', 'bmh',
 → 'seaborn-deep', 'fivethirtyeight', 'grayscale', 'seaborn', 'seaborn-talk', 'seaborn-bright',
 → 'seaborn-white', 'seaborn-ticks', 'seaborn-dark-palette', 'seaborn-dark', '_classic_test',
 → 'seaborn-colorblind', 'dark_background', 'seaborn-whitegrid', 'seaborn-notebook', 'tableau-
 → colorblind10', 'Solarize_Light2', 'seaborn-muted', 'classic']
```

Defining your own style

You can create custom styles and use them by calling `style.use` with the path or URL to the style sheet. Additionally, if you add your `<style-name>.mplstyle` file to `mpl_configdir/stylelib`, you can reuse your custom style sheet with a call to `style.use(<style-name>)`. By default `mpl_configdir` should be `~/.config/matplotlib`, but you can check where yours is with `matplotlib.get_configdir()`; you may need to create this directory. You also can change the directory where matplotlib looks for the `stylelib/` folder by setting the `MPLCONFIGDIR` environment variable, see matplotlib configuration and cache directory locations.

Note that a custom style sheet in `mpl_configdir/stylelib` will override a style sheet defined by matplotlib if the styles have the same name.

For example, you might want to create `mpl_configdir/stylelib/presentation.mplstyle` with the following:

```
axes.titlesize : 24
axes.labelsize : 20
lines.linewidth : 3
lines.markersize : 10
xtick.labelsize : 16
ytick.labelsize : 16
```

Then, when you want to adapt a plot designed for a paper to one that looks good in a presentation, you can just add:

```
>>> import matplotlib.pyplot as plt
>>> plt.style.use('presentation')
```

Composing styles

Style sheets are designed to be composed together. So you can have a style sheet that customizes colors and a separate style sheet that alters element sizes for presentations. These styles can easily be combined by passing a list of styles:

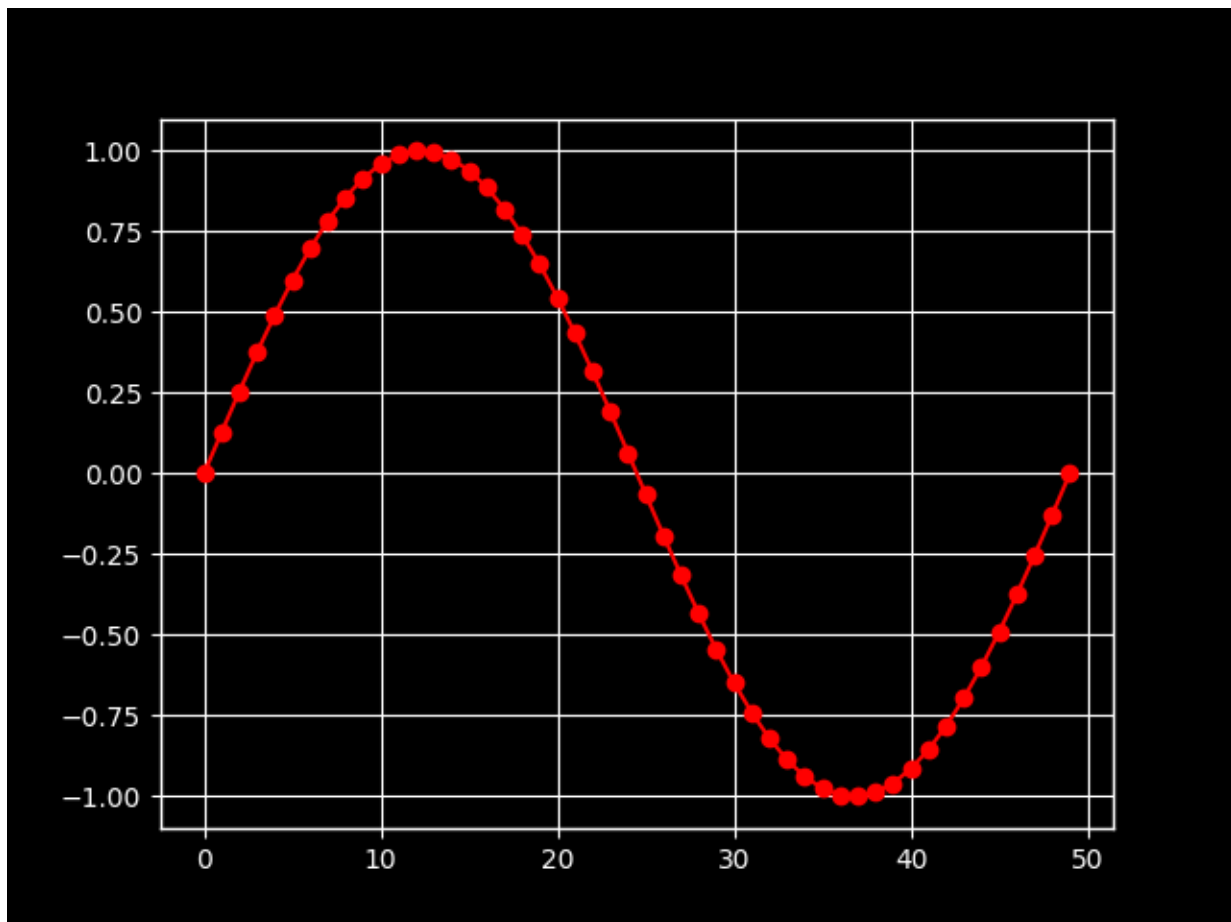
```
>>> import matplotlib.pyplot as plt
>>> plt.style.use(['dark_background', 'presentation'])
```

Note that styles further to the right will overwrite values that are already defined by styles on the left.

Temporary styling

If you only want to use a style for a specific block of code but don't want to change the global styling, the style package provides a context manager for limiting your changes to a specific scope. To isolate your styling changes, you can write something like the following:

```
with plt.style.context(('dark_background')):
    plt.plot(np.sin(np.linspace(0, 2 * np.pi)), 'r-o')
plt.show()
```

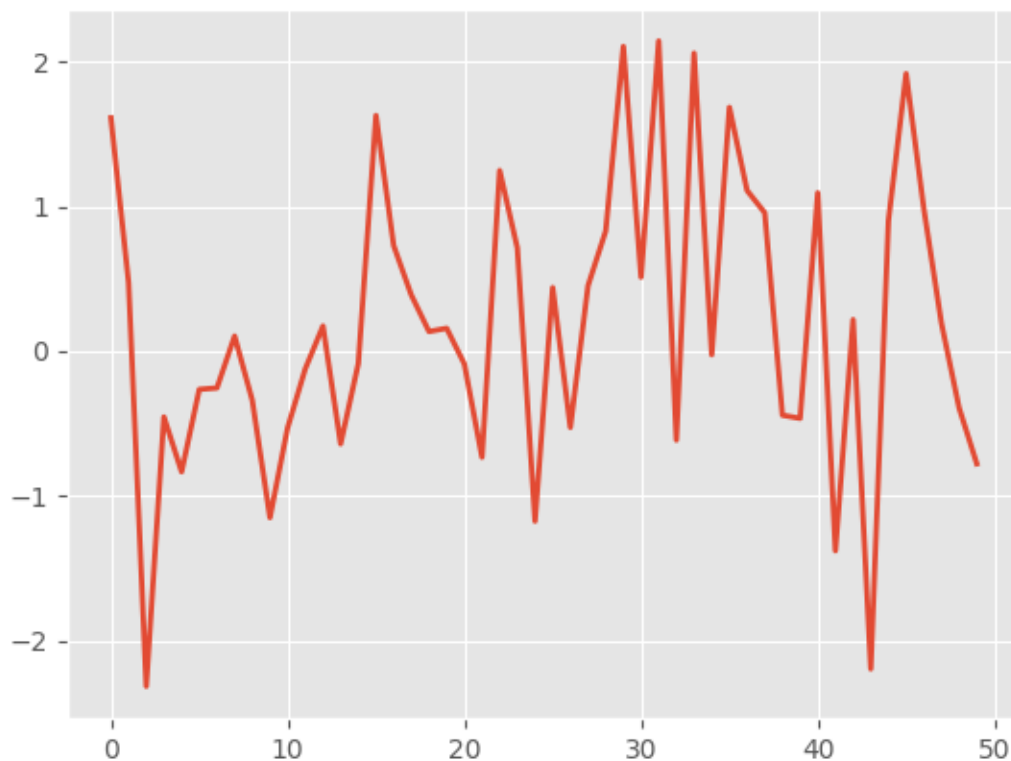


3.1.3 matplotlib rcParams

Dynamic rc settings

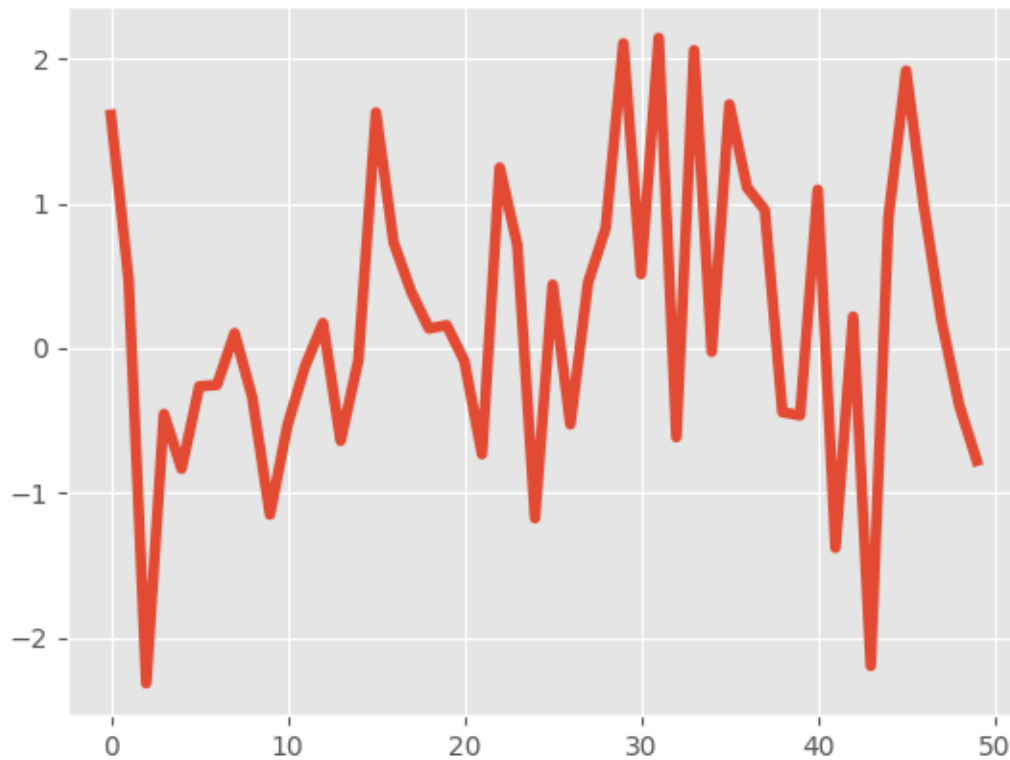
You can also dynamically change the default rc settings in a python script or interactively from the python shell. All of the rc settings are stored in a dictionary-like variable called `matplotlib.rcParams`, which is global to the matplotlib package. `rcParams` can be modified directly, for example:

```
mpl.rcParams['lines.linewidth'] = 2
mpl.rcParams['lines.color'] = 'r'
plt.plot(data)
```



Matplotlib also provides a couple of convenience functions for modifying rc settings. The `matplotlib.rc()` command can be used to modify multiple settings in a single group at once, using keyword arguments:

```
mpl.rc('lines', linewidth=4, color='g')
plt.plot(data)
```



The `matplotlib.rcdefaults()` command will restore the standard matplotlib default settings.

There is some degree of validation when setting the values of `rcParams`, see `matplotlib.rcsetup` for details.

The `matplotlibrc` file

matplotlib uses `matplotlibrc` configuration files to customize all kinds of properties, which we call `rc` settings or `rc` parameters. You can control the defaults of almost every property in matplotlib: figure size and dpi, line width, color and style, axes, axis and grid properties, text and font properties and so on. matplotlib looks for `matplotlibrc` in four locations, in the following order:

1. `matplotlibrc` in the current working directory, usually used for specific customizations that you do not want to apply elsewhere.
2. `$MATPLOTLIBRC` if it is a file, else `$MATPLOTLIBRC/matplotlibrc`.
3. It next looks in a user-specific place, depending on your platform:
 - On Linux and FreeBSD, it looks in `.config/matplotlib/matplotlibrc` (or `$XDG_CONFIG_HOME/matplotlib/matplotlibrc`) if you've customized your environment.
 - On other platforms, it looks in `.matplotlib/matplotlibrc`.

See matplotlib configuration and cache directory locations.

4. `INSTALL/matplotlib/mpl-data/matplotlibrc`, where `INSTALL` is something like `/usr/lib/python3.5/site-packages` on Linux, and maybe `C:\Python35\Lib\site-packages` on Windows. Every time you install matplotlib, this file will be overwritten, so if you want your customizations to be saved, please move this file to your user-specific matplotlib directory.

Once a `matplotlibrc` file has been found, it will not search any of the other paths.

To display where the currently active `matplotlibrc` file was loaded from, one can do the following:

```
>>> import matplotlib
>>> matplotlib.matplotlib_fname()
'/home/foo/.config/matplotlib/matplotlibrc'
```

See below for a sample `matplotlibrc` file. Although all parameters are optional, you should almost always set the backend or else matplotlib will choose Agg, a non-interactive backend. This can lead to unexpected behavior, since if you do not have a `matplotlibrc` file, it would normally fall back to `INSTALL/matplotlib/mpl-data/matplotlibrc`, which is often set to an interactive backend by the package maintainer.

A sample `matplotlibrc` file

```
##### MATPLOTLIBRC FORMAT

## This is a sample matplotlib configuration file - you can find a copy
## of it on your system in
## site-packages/matplotlib/mpl-data/matplotlibrc. If you edit it
## there, please note that it will be overwritten in your next install.
## If you want to keep a permanent local copy that will not be
## overwritten, place it in the following location:
## unix/linux:
##     $HOME/.config/matplotlib/matplotlibrc or
##     $XDG_CONFIG_HOME/matplotlib/matplotlibrc (if $XDG_CONFIG_HOME is set)
## other platforms:
##     $HOME/.matplotlib/matplotlibrc
##
## See http://matplotlib.org/users/customizing.html#the-matplotlibrc-file for
## more details on the paths which are checked for the configuration file.
##
## This file is best viewed in a editor which supports python mode
## syntax highlighting. Blank lines, or lines starting with a comment
## symbol, are ignored, as are trailing comments. Other lines must
## have the format
##     key : val ## optional comment
##
## Colors: for the color values below, you can either use - a
```

(continues on next page)

(continued from previous page)

```

## matplotlib color string, such as r, k, or b - an rgb tuple, such as
## (1.0, 0.5, 0.0) - a hex string, such as ff00ff - a scalar
## grayscale intensity such as 0.75 - a legal html color name, e.g., red,
## blue, darkslategray

##### CONFIGURATION BEGINS HERE

## The default backend; one of GTK GTKAgg GTKCairo GTK3Agg GTK3Cairo
## MacOSX Qt4Agg Qt5Agg TkAgg WX WXAgg Agg Cairo GDK PS PDF SVG
## Template.
## You can also deploy your own backend outside of matplotlib by
## referring to the module name (which must be in the PYTHONPATH) as
## 'module://my_backend'.
##
## If you omit this parameter, it will always default to "Agg", which is a
## non-interactive backend.
backend      : $TEMPLATE_BACKEND

## Note that this can be overridden by the environment variable
## QT_API used by Enthought Tool Suite (ETS); valid values are
## "pyqt" and "pyside". The "pyqt" setting has the side effect of
## forcing the use of Version 2 API for QString and QVariant.

## The port to use for the web server in the WebAgg backend.
#webagg.port : 8988

## The address on which the WebAgg web server should be reachable
#webagg.address : 127.0.0.1

## If webagg.port is unavailable, a number of other random ports will
## be tried until one that is available is found.
#webagg.port_retries : 50

## When True, open the webbrowser to the plot that is shown
#webagg.open_in_browser : True

## if you are running pyplot inside a GUI and your backend choice
## conflicts, we will automatically try to find a compatible one for
## you if backend_fallback is True
#backend_fallback: True

#interactive : False
#toolbar      : toolbar2  ## None | toolbar2 ("classic" is deprecated)
#timezone     : UTC       ## a pytz timezone string, e.g., US/Central or Europe/Paris

```

(continues on next page)

(continued from previous page)

```

## Where your matplotlib data lives if you installed to a non-default
## location. This is where the matplotlib fonts, bitmaps, etc reside
#datapath : /home/jdhunter/mpldata

##### LINES
## See http://matplotlib.org/api/artist\_api.html#module-matplotlib.lines for more
## information on line properties.
#lines.linewidth : 1.5    ## line width in points
#lines.linestyle : -      ## solid line
#lines.color      : C0     ## has no affect on plot(); see axes.prop_cycle
#lines.marker     : None   ## the default marker
#lines.markeredgewidth : 1.0  ## the line width around the marker symbol
#lines.markersize : 6      ## markersize, in points
#lines.dash_joinstyle : round  ## miter|round|bevel
#lines.dash_capstyle : butt   ## butt|round|projecting
#lines.solid_joinstyle : round  ## miter|round|bevel
#lines.solid_capstyle : projecting ## butt|round|projecting
#lines.antialiased : True     ## render lines in antialiased (no jaggies)

## The three standard dash patterns. These are scaled by the linewidth.
#lines.dashed_pattern : 3.7, 1.6
#lines.dashdot_pattern : 6.4, 1.6, 1, 1.6
#lines.dotted_pattern : 1, 1.65
#lines.scale_dashes : True

#markers.fillstyle: full ## full|left|right|bottom|top|none

##### PATCHES
## Patches are graphical objects that fill 2D space, like polygons or
## circles. See
## http://matplotlib.org/api/artist\_api.html#module-matplotlib.patches
## information on patch properties
#patch.linewidth : 1      ## edge width in points.
#patch.facecolor : C0
#patch.edgecolor : k      ## if forced, or patch is not filled
#patch.force_edgecolor : False ## True to always use edgecolor
#patch.antialiased : True  ## render patches in antialiased (no jaggies)

##### HATCHES
#hatch.color : k
#hatch.linewidth : 1.0

##### Boxplot
#boxplot.notch : False

```

(continues on next page)

(continued from previous page)

```

#boxplot.vertical      : True
#boxplot.whiskers      : 1.5
#boxplot.bootstrap    : None
#boxplot.patchartist   : False
#boxplot.showmeans     : False
#boxplot.showcaps      : True
#boxplot.showbox       : True
#boxplot.showfliers    : True
#boxplot.meanline      : False

#boxplot.flierprops.color      : k
#boxplot.flierprops.marker     : o
#boxplot.flierprops.markerfacecolor : none
#boxplot.flierprops.markeredgecolor : k
#boxplot.flierprops.markersize   : 6
#boxplot.flierprops.linestyle    : none
#boxplot.flierprops.linewidth    : 1.0

#boxplot.boxprops.color      : k
#boxplot.boxprops.linewidth  : 1.0
#boxplot.boxprops.linestyle  : -

#boxplot.whiskerprops.color   : k
#boxplot.whiskerprops.linewidth : 1.0
#boxplot.whiskerprops.linestyle : -

#boxplot.capprops.color      : k
#boxplot.capprops.linewidth  : 1.0
#boxplot.capprops.linestyle  : -

#boxplot.medianprops.color   : C1
#boxplot.medianprops.linewidth : 1.0
#boxplot.medianprops.linestyle : -

#boxplot.meanprops.color     : C2
#boxplot.meanprops.marker    : ^
#boxplot.meanprops.markerfacecolor : C2
#boxplot.meanprops.markeredgecolor : C2
#boxplot.meanprops.markersize   : 6
#boxplot.meanprops.linestyle    : --
#boxplot.meanprops.linewidth    : 1.0

#### FONT

```

(continues on next page)

(continued from previous page)

```

## font properties used by text.Text. See
## http://matplotlib.org/api/font_manager_api.html for more
## information on font properties. The 6 font properties used for font
## matching are given below with their default values.
##
## The font.family property has five values: 'serif' (e.g., Times),
## 'sans-serif' (e.g., Helvetica), 'cursive' (e.g., Zapf-Chancery),
## 'fantasy' (e.g., Western), and 'monospace' (e.g., Courier). Each of
## these font families has a default list of font names in decreasing
## order of priority associated with them. When text.usetex is False,
## font.family may also be one or more concrete font names.
##
## The font.style property has three values: normal (or roman), italic
## or oblique. The oblique style will be used for italic, if it is not
## present.
##
## The font.variant property has two values: normal or small-caps. For
## TrueType fonts, which are scalable fonts, small-caps is equivalent
## to using a font size of 'smaller', or about 83%% of the current font
## size.
##
## The font.weight property has effectively 13 values: normal, bold,
## bolder, lighter, 100, 200, 300, ..., 900. Normal is the same as
## 400, and bold is 700. bolder and lighter are relative values with
## respect to the current weight.
##
## The font.stretch property has 11 values: ultra-condensed,
## extra-condensed, condensed, semi-condensed, normal, semi-expanded,
## expanded, extra-expanded, ultra-expanded, wider, and narrower. This
## property is not currently implemented.
##
## The font.size property is the default font size for text, given in pts.
## 10 pt is the standard value.

#font.family      : sans-serif
#font.style       : normal
#font.variant     : normal
#font.weight      : normal
#font.stretch     : normal
## note that font.size controls default text sizes. To configure
## special text sizes tick labels, axes, labels, title, etc, see the rc
## settings for axes and ticks. Special text sizes can be defined
## relative to font.size, using the following values: xx-small, x-small,
## small, medium, large, x-large, xx-large, larger, or smaller
#font.size        : 10.0

```

(continues on next page)

(continued from previous page)

```

#font.serif      : DejaVu Serif, Bitstream Vera Serif, Computer Modern Roman, New Century
↳ Schoolbook, Century Schoolbook L, Utopia, ITC Bookman, Bookman, Nimbus Roman No9 L,
↳ Times New Roman, Times, Palatino, Charter, serif
#font.sans-serif  : DejaVu Sans, Bitstream Vera Sans, Computer Modern Sans Serif, Lucida
↳ Grande, Verdana, Geneva, Lucid, Arial, Helvetica, Avant Garde, sans-serif
#font.cursive     : Apple Chancery, Textile, Zapf Chancery, Sand, Script MT, Felipa, cursive
#font.fantasy     : Comic Sans MS, Chicago, Charcoal, ImpactWestern, Humor Sans, xkcd,
↳ fantasy
#font.monospace   : DejaVu Sans Mono, Bitstream Vera Sans Mono, Computer Modern
↳ Typewriter, Andale Mono, Nimbus Mono L, Courier New, Courier, Fixed, Terminal,
↳ monospace

#### TEXT
## text properties used by text.Text. See
## http://matplotlib.org/api/artist\_api.html#module-matplotlib.text for more
## information on text properties
#text.color       : k

#### LaTeX customizations. See http://wiki.scipy.org/Cookbook/Matplotlib/UsingTex
#text.usetex      : False ## use latex for all text handling. The following fonts
                        ## are supported through the usual rc parameter settings:
                        ## new century schoolbook, bookman, times, palatino,
                        ## zapf chancery, charter, serif, sans-serif, helvetica,
                        ## avant garde, courier, monospace, computer modern roman,
                        ## computer modern sans serif, computer modern typewriter
                        ## If another font is desired which can loaded using the
                        ## LaTeX \usepackage command, please inquire at the
                        ## matplotlib mailing list
#text.latex.unicode : False ## use "ucs" and "inputenc" LaTeX packages for handling
                        ## unicode strings.
#text.latex.preamble : ## IMPROPER USE OF THIS FEATURE WILL LEAD TO
↳ LATEX FAILURES
                        ## AND IS THEREFORE UNSUPPORTED. PLEASE DO NOT ASK
↳ FOR HELP
                        ## IF THIS FEATURE DOES NOT DO WHAT YOU EXPECT IT TO.
                        ## preamble is a comma separated list of LaTeX statements
                        ## that are included in the LaTeX document preamble.
                        ## An example:
                        ## text.latex.preamble : \usepackage{bm},\usepackage{euler}
                        ## The following packages are always loaded with usetex, so
                        ## beware of package collisions: color, geometry, graphicx,
                        ## type1cm, textcomp. Adobe Postscript (PSSNFS) font packages
                        ## may also be loaded, depending on your font settings
#text.latex.preview : False

```

(continues on next page)

(continued from previous page)

```

#text.hinting : auto  ## May be one of the following:
                        ## none: Perform no hinting
                        ## auto: Use FreeType's autohinter
                        ## native: Use the hinting information in the
                        #       font file, if available, and if your
                        #       FreeType library supports it
                        ## either: Use the native hinting information,
                        #       or the autohinter if none is available.
                        ## For backward compatibility, this value may also be
                        ## True === 'auto' or False === 'none'.
#text.hinting_factor : 8 ## Specifies the amount of softness for hinting in the
                        ## horizontal direction. A value of 1 will hint to full
                        ## pixels. A value of 2 will hint to half pixels etc.
#text.antialiased : True ## If True (default), the text will be antialiased.
                        ## This only affects the Agg backend.

## The following settings allow you to select the fonts in math mode.
## They map from a TeX font name to a fontconfig font pattern.
## These settings are only used if mathtext.fontset is 'custom'.
## Note that this "custom" mode is unsupported and may go away in the
## future.
#mathtext.cal : cursive
#mathtext.rm : sans
#mathtext.tt : monospace
#mathtext.it : sans:italic
#mathtext.bf : sans:bold
#mathtext.sf : sans
#mathtext.fontset : dejavusans ## Should be 'dejavusans' (default),
                        ## 'dejavuserif', 'cm' (Computer Modern), 'stix',
                        ## 'stixsans' or 'custom'
#mathtext.fallback_to_cm : True ## When True, use symbols from the Computer Modern
                        ## fonts when a symbol can not be found in one of
                        ## the custom math fonts.
#mathtext.default : it ## The default font to use for math.
                        ## Can be any of the LaTeX font names, including
                        ## the special name "regular" for the same font
                        ## used in regular text.

#### AXES
## default face and edge color, default tick sizes,
## default fontsizes for ticklabels, and so on. See
## http://matplotlib.org/api/axes\_api.html#module-matplotlib.axes
#axes.facecolor : w ## axes background color
#axes.edgecolor : k ## axes edge color
#axes.linewidth : 0.8 ## edge linewidth

```

(continues on next page)

(continued from previous page)

```

#axes.grid      : False  ## display grid or not
#axes.grid.axis : both   ## which axis the grid should apply to
#axes.grid.which : major  ## gridlines at major, minor or both ticks
#axes.titlesize : large   ## fontsize of the axes title
#axes.titleweight : normal ## font weight of title
#axes.titlepad   : 6.0    ## pad between axes and title in points
#axes.labelsize  : medium  ## fontsize of the x any y labels
#axes.labelpad   : 4.0    ## space between label and axis
#axes.labelweight : normal ## weight of the x and y labels
#axes.labelcolor : k
#axes.axisbelow  : line   ## draw axis gridlines and ticks below
                        ## patches (True); above patches but below
                        ## lines ('line'); or above all (False)
#axes.formatter.limits : -7, 7 ## use scientific notation if log10
                        ## of the axis range is smaller than the
                        ## first or larger than the second
#axes.formatter.use_locale : False ## When True, format tick labels
                        ## according to the user's locale.
                        ## For example, use ',' as a decimal
                        ## separator in the fr_FR locale.
#axes.formatter.use_mathtext : False ## When True, use mathtext for scientific
                        ## notation.
#axes.formatter.min_exponent: 0 ## minimum exponent to format in scientific notation
#axes.formatter.useoffset   : True  ## If True, the tick label formatter
                        ## will default to labeling ticks relative
                        ## to an offset when the data range is
                        ## small compared to the minimum absolute
                        ## value of the data.
#axes.formatter.offset_threshold : 4 ## When useoffset is True, the offset
                        ## will be used when it can remove
                        ## at least this number of significant
                        ## digits from tick labels.
#axes.spines.left  : True  ## display axis spines
#axes.spines.bottom : True
#axes.spines.top   : True
#axes.spines.right  : True
#axes.unicode_minus : True  ## use unicode for the minus symbol
                        ## rather than hyphen. See
                        ## http://en.wikipedia.org/wiki/Plus\_and\_minus\_signs#Character\_codes
#axes.prop_cycle    : cycler('color', ['1f77b4', 'ff7f0e', '2ca02c', 'd62728', '9467bd', '8c564b',
    ↪ 'e377c2', '7f7f7f', 'bcbd22', '17becf'])
                        ## color cycle for plot lines as list of string
                        ## colorspecs: single letter, long name, or web-style hex
                        ## Note the use of string escapes here ('1f77b4', instead of ↪
    ↪ 1f77b4)

```

(continues on next page)

(continued from previous page)

```

    ## as opposed to the rest of this file.
#axes.autolimit_mode : data ## How to scale axes limits to the data.
    ## Use "data" to use data limits, plus some margin
    ## Use "round_number" move to the nearest "round" number
#axes.xmargin       : .05 ## x margin. See `axes.Axes.margins`
#axes.ymargin       : .05 ## y margin See `axes.Axes.margins`
#polaraxes.grid     : True  ## display grid on polar axes
#axes3d.grid        : True  ## display grid on 3d axes

##### DATES
## These control the default format strings used in AutoDateFormatter.
## Any valid format datetime format string can be used (see the python
## `datetime` for details). For example using '%%x' will use the locale date representation
## '%%X' will use the locale time representation and '%%c' will use the full locale datetime
## representation.
## These values map to the scales:
##   {'year': 365, 'month': 30, 'day': 1, 'hour': 1/24, 'minute': 1 / (24 * 60)}

#date.autoformatter.year      : %Y
#date.autoformatter.month    : %Y-%m
#date.autoformatter.day      : %Y-%m-%d
#date.autoformatter.hour     : %m-%d %H
#date.autoformatter.minute   : %d %H:%M
#date.autoformatter.second    : %H:%M:%S
#date.autoformatter.microsecond : %M:%S.%f

##### TICKS
## see http://matplotlib.org/api/axis\_api.html#matplotlib.axis.Tick
#xtick.top                 : False ## draw ticks on the top side
#xtick.bottom              : True  ## draw ticks on the bottom side
#xtick.labeltop            : False ## draw label on the top
#xtick.labelbottom         : True  ## draw label on the bottom
#xtick.major.size          : 3.5   ## major tick size in points
#xtick.minor.size          : 2     ## minor tick size in points
#xtick.major.width         : 0.8   ## major tick width in points
#xtick.minor.width         : 0.6   ## minor tick width in points
#xtick.major.pad           : 3.5   ## distance to major tick label in points
#xtick.minor.pad           : 3.4   ## distance to the minor tick label in points
#xtick.color               : k     ## color of the tick labels
#xtick.labelsize           : medium ## fontsize of the tick labels
#xtick.direction           : out    ## direction: in, out, or inout
#xtick.minor.visible       : False  ## visibility of minor ticks on x-axis
#xtick.major.top           : True   ## draw x axis top major ticks
#xtick.major.bottom        : True   ## draw x axis bottom major ticks
#xtick.minor.top           : True   ## draw x axis top minor ticks

```

(continues on next page)

(continued from previous page)

```

#xtick.minor.bottom : True  ## draw x axis bottom minor ticks
#xtick.alignment    : center ## alignment of xticks

#ytick.left         : True  ## draw ticks on the left side
#ytick.right        : False ## draw ticks on the right side
#ytick.lableleft    : True  ## draw tick labels on the left side
#ytick.lablelright  : False ## draw tick labels on the right side
#ytick.major.size   : 3.5   ## major tick size in points
#ytick.minor.size    : 2     ## minor tick size in points
#ytick.major.width   : 0.8   ## major tick width in points
#ytick.minor.width   : 0.6   ## minor tick width in points
#ytick.major.pad     : 3.5   ## distance to major tick label in points
#ytick.minor.pad     : 3.4   ## distance to the minor tick label in points
#ytick.color         : k     ## color of the tick labels
#ytick.labelsize     : medium ## fontsize of the tick labels
#ytick.direction     : out   ## direction: in, out, or inout
#ytick.minor.visible : False ## visibility of minor ticks on y-axis
#ytick.major.left    : True  ## draw y axis left major ticks
#ytick.major.right   : True  ## draw y axis right major ticks
#ytick.minor.left     : True  ## draw y axis left minor ticks
#ytick.minor.right    : True  ## draw y axis right minor ticks
#ytick.alignment     : center_baseline ## alignment of yticks

#### GRIDS
#grid.color          : b0b0b0 ## grid color
#grid.linestyle       : -      ## solid
#grid.linewidth       : 0.8    ## in points
#grid.alpha           : 1.0    ## transparency, between 0.0 and 1.0

#### Legend
#legend.loc           : best
#legend.frameon       : True   ## if True, draw the legend on a background patch
#legend.framealpha     : 0.8   ## legend patch transparency
#legend.facecolor      : inherit ## inherit from axes.facecolor; or color spec
#legend.edgecolor      : 0.8   ## background patch boundary color
#legend.fancybox       : True   ## if True, use a rounded box for the
                                ## legend background, else a rectangle
#legend.shadow         : False  ## if True, give background a shadow effect
#legend.numpoints      : 1      ## the number of marker points in the legend line
#legend.scatterpoints  : 1      ## number of scatter points
#legend.markerscale    : 1.0    ## the relative size of legend markers vs. original
#legend.fontsize       : medium
## Dimensions as fraction of fontsize:
#legend.borderpad      : 0.4    ## border whitespace
#legend.labelspacing   : 0.5    ## the vertical space between the legend entries

```

(continues on next page)

(continued from previous page)

```

#legend.handlelength : 2.0    ## the length of the legend lines
#legend.handleheight : 0.7    ## the height of the legend handle
#legend.handletextpad : 0.8    ## the space between the legend line and legend text
#legend.borderaxespadd : 0.5    ## the border between the axes and legend edge
#legend.columnspacing : 2.0    ## column separation

#### FIGURE
## See http://matplotlib.org/api/figure\_api.html#matplotlib.figure.Figure
#figure.titlesize : large    ## size of the figure title (Figure.suptitle())
#figure.titleweight : normal  ## weight of the figure title
#figure.figsize : 6.4, 4.8    ## figure size in inches
#figure.dpi : 100            ## figure dots per inch
#figure.facecolor : w        ## figure facecolor; 0.75 is scalar gray
#figure.edgecolor : w        ## figure edgecolor
#figure.frameon : True       ## enable figure frame
#figure.max_open_warning : 20 ## The maximum number of figures to open through
                             ## the pyplot interface before emitting a warning.
                             ## If less than one this feature is disabled.

## The figure subplot parameters. All dimensions are a fraction of the
#figure.subplot.left : 0.125  ## the left side of the subplots of the figure
#figure.subplot.right : 0.9    ## the right side of the subplots of the figure
#figure.subplot.bottom : 0.11  ## the bottom of the subplots of the figure
#figure.subplot.top : 0.88     ## the top of the subplots of the figure
#figure.subplot.wspace : 0.2   ## the amount of width reserved for space between subplots,
                             ## expressed as a fraction of the average axis width
#figure.subplot.hspace : 0.2   ## the amount of height reserved for space between subplots,
                             ## expressed as a fraction of the average axis height

## Figure layout
#figure.autolayout : False    ## When True, automatically adjust subplot
                             ## parameters to make the plot fit the figure
                             ## using `tight_layout`
#figure.constrained_layout.use: False ## When True, automatically make plot
                             ## elements fit on the figure. (Not compatible
                             ## with `autolayout`, above).
#figure.constrained_layout.h_pad : 0.04167 ## Padding around axes objects. Float
→representing
#figure.constrained_layout.w_pad : 0.04167 ## inches. Default is 3./72. inches (3 pts)
#figure.constrained_layout.hspace : 0.02   ## Space between subplot groups. Float representing
#figure.constrained_layout.wspace : 0.02   ## a fraction of the subplot widths being separated.

#### IMAGES
#image.aspect : equal        ## equal | auto | a number
#image.interpolation : nearest ## see help(imshow) for options
#image.cmap : viridis        ## A colormap name, gray etc...

```

(continues on next page)

(continued from previous page)

```

#image.lut : 256          ## the size of the colormap lookup table
#image.origin : upper      ## lower | upper
#image.resample : True
#image.composite_image : True  ## When True, all the images on a set of axes are
                                ## combined into a single composite image before
                                ## saving a figure as a vector graphics file,
                                ## such as a PDF.

##### CONTOUR PLOTS
#contour.negative_linestyle : dashed ## string or on-off ink sequence
#contour.corner_mask : True  ## True | False | legacy

##### ERRORBAR PLOTS
#errorbar.capsize : 0        ## length of end cap on error bars in pixels

##### HISTOGRAM PLOTS
#hist.bins : 10              ## The default number of histogram bins.
                                ## If Numpy 1.11 or later is
                                ## installed, may also be `auto`

##### SCATTER PLOTS
#scatter.marker : o          ## The default marker type for scatter plots.

##### Agg rendering
##### Warning: experimental, 2008/10/10
#agg.path.chunksize : 0      ## 0 to disable; values in the range
                                ## 10000 to 100000 can improve speed slightly
                                ## and prevent an Agg rendering failure
                                ## when plotting very large data sets,
                                ## especially if they are very gappy.
                                ## It may cause minor artifacts, though.
                                ## A value of 20000 is probably a good
                                ## starting point.

##### PATHS
#path.simplify : True  ## When True, simplify paths by removing "invisible"
                                ## points to reduce file size and increase rendering
                                ## speed
#path.simplify_threshold : 0.111111111111 ## The threshold of similarity below which
                                ## vertices will be removed in the
                                ## simplification process
#path.snap : True  ## When True, rectilinear axis-aligned paths will be snapped to
                                ## the nearest pixel when certain criteria are met. When False,
                                ## paths will never be snapped.
#path.sketch : None ## May be none, or a 3-tuple of the form (scale, length,
                                ## randomness).

```

(continues on next page)

(continued from previous page)

```

    ## *scale* is the amplitude of the wiggle
    ## perpendicular to the line (in pixels). *length*
    ## is the length of the wiggle along the line (in
    ## pixels). *randomness* is the factor by which
    ## the length is randomly scaled.
#path.effects : [] ##

#### SAVING FIGURES
## the default savefig params can be different from the display params
## e.g., you may want a higher resolution, or to make the figure
## background white
#savefig.dpi      : figure  ## figure dots per inch or 'figure'
#savefig.facecolor : w      ## figure facecolor when saving
#savefig.edgecolor : w      ## figure edgecolor when saving
#savefig.format   : png     ## png, ps, pdf, svg
#savefig.bbox     : standard ## 'tight' or 'standard'.
                        ## 'tight' is incompatible with pipe-based animation
                        ## backends but will workd with temporary file based ones:
                        ## e.g. setting animation.writer to ffmpeg will not work,
                        ## use ffmpeg_file instead
#savefig.pad_inches : 0.1    ## Padding to be used when bbox is set to 'tight'
#savefig.jpeg_quality: 95    ## when a jpeg is saved, the default quality parameter.
#savefig.directory : ~      ## default directory in savefig dialog box,
                        ## leave empty to always use current working directory
#savefig.transparent : False ## setting that controls whether figures are saved with a
                        ## transparent background by default
#savefig.frameon : True      ## enable frame of figure when saving
#savefig.orientation : portrait ## Orientation of saved figure

#### tk backend params
#tk.window_focus : False    ## Maintain shell focus for TkAgg

#### ps backend params
#ps.papersize     : letter   ## auto, letter, legal, ledger, A0-A10, B0-B10
#ps.useafm        : False    ## use of afm fonts, results in small files
#ps.usedistiller  : False    ## can be: None, ghostscript or xpdf
                        ## Experimental: may produce smaller files.
                        ## xpdf intended for production of publication quality files,
                        ## but requires ghostscript, xpdf and ps2eps
#ps.distiller.res : 6000     ## dpi
#ps.fonttype      : 3        ## Output Type 3 (Type3) or Type 42 (TrueType)

#### pdf backend params
#pdf.compression  : 6        ## integer from 0 to 9
                        ## 0 disables compression (good for debugging)

```

(continues on next page)

(continued from previous page)

```

#pdf.fonttype      : 3      ## Output Type 3 (Type3) or Type 42 (TrueType)
#pdf.use14corefonts : False
#pdf.inheritcolor  : False

### svg backend params
#svg.image_inline : True    ## write raster image data directly into the svg file
#svg.fonttype : path      ## How to handle SVG fonts:
    ## none: Assume fonts are installed on the machine where the SVG will be viewed.
    ## path: Embed characters as paths -- supported by most SVG renderers
    ## svgfont: Embed characters as SVG fonts -- supported only by Chrome,
    ##           Opera and Safari
#svg.hashsalt : None      ## if not None, use this string as hash salt
                        ## instead of uuid4

### pgf parameter
#pgf.rcfonts : True
#pgf.preamble :
#pgf.texsystem : xelatex
#pgf.debug : False

### docstring params
##docstring.hardcopy = False ## set this when you want to generate hardcopy docstring

## Event keys to interact with figures/plots via keyboard.
## Customize these settings according to your needs.
## Leave the field(s) empty if you don't need a key-map. (i.e., fullscreen : '')
#keymap.fullscreen : f, ctrl+f      ## toggling
#keymap.home : h, r, home          ## home or reset mnemonic
#keymap.back : left, c, backspace   ## forward / backward keys to enable
#keymap.forward : right, v          ## left handed quick navigation
#keymap.pan : p                    ## pan mnemonic
#keymap.zoom : o                   ## zoom mnemonic
#keymap.save : s, ctrl+s           ## saving current figure
#keymap.quit : ctrl+w, cmd+w, q     ## close the current figure
#keymap.quit_all : W, cmd+W, Q      ## close all figures
#keymap.grid : g                   ## switching on/off major grids in current axes
#keymap.grid_minor : G             ## switching on/off minor grids in current axes
#keymap.yscale : l                 ## toggle scaling of y-axes ('log'/'linear')
#keymap.xscale : k, L              ## toggle scaling of x-axes ('log'/'linear')
#keymap.all_axes : a               ## enable all axes

## Control location of examples data files
#examples.directory :              ## directory to look in for custom installation

###ANIMATION settings
#animation.html : none             ## How to display the animation as HTML in

```

(continues on next page)

(continued from previous page)

```
## the IPython notebook. 'html5' uses
## HTML5 video tag; 'jshtml' creates a
## Javascript animation
#animation.writer : ffmpeg      ## MovieWriter 'backend' to use
#animation.codec : h264        ## Codec to use for writing movie
#animation.bitrate: -1         ## Controls size/quality tradeoff for movie.
## -1 implies let utility auto-determine
#animation.frame_format: png   ## Controls frame format used by temp files
#animation.html_args:         ## Additional arguments to pass to html writer
#animation.ffmpeg_path: ffmpeg ## Path to ffmpeg binary. Without full path
## $PATH is searched
#animation.ffmpeg_args:       ## Additional arguments to pass to ffmpeg
#animation.avconv_path: avconv ## Path to avconv binary. Without full path
## $PATH is searched
#animation.avconv_args:       ## Additional arguments to pass to avconv
#animation.convert_path: convert ## Path to ImageMagick's convert binary.
## On Windows use the full path since convert
## is also the name of a system tool.
#animation.convert_args:      ## Additional arguments to pass to convert
#animation.embed_limit : 20.0
```

Note: [Click here to download the full example code](#)

3.1.4 Image tutorial

A short tutorial on plotting images with Matplotlib.

Startup commands

First, let's start IPython. It is a most excellent enhancement to the standard Python prompt, and it ties in especially well with Matplotlib. Start IPython either at a shell, or the IPython Notebook now.

With IPython started, we now need to connect to a GUI event loop. This tells IPython where (and how) to display plots. To connect to a GUI loop, execute the `%matplotlib` magic at your IPython prompt. There's more detail on exactly what this does at [IPython's documentation on GUI event loops](#).

If you're using IPython Notebook, the same commands are available, but people commonly use a specific argument to the `%matplotlib` magic:

```
In [1]: %matplotlib inline
```

This turns on inline plotting, where plot graphics will appear in your notebook. This has important implications for interactivity. For inline plotting, commands in cells below the cell that outputs a plot will not affect the plot. For example, changing the color map is not possible from cells below the cell that creates a plot. However, for other backends, such as Qt5, that open a separate window, cells below those that create the plot will change the plot - it is a live object in memory.

This tutorial will use matplotlib's imperative-style plotting interface, pyplot. This interface maintains global state, and is very useful for quickly and easily experimenting with various plot settings. The alternative is the object-oriented interface, which is also very powerful, and generally more suitable for large application development. If you'd like to learn about the object-oriented interface, a great place to start is our [FAQ on usage](#). For now, let's get on with the imperative-style approach:

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
```

Importing image data into Numpy arrays

Loading image data is supported by the [Pillow](#) library. Natively, Matplotlib only supports PNG images. The commands shown below fall back on Pillow if the native read fails.

The image used in this example is a PNG file, but keep that Pillow requirement in mind for your own data.

Here's the image we're going to play with:



It's a 24-bit RGB PNG image (8 bits for each of R, G, B). Depending on where you get your data, the other kinds of image that you'll most likely encounter are RGBA images, which allow for transparency, or single-channel grayscale (luminosity) images. You can right click on it and choose "Save image as" to download it to your computer for the rest of this tutorial.

And here we go...

```
img = mpimg.imread('../doc/_static/stinkbug.png')
print(img)
```

Out:

```
[[[0.40784314 0.40784314 0.40784314]
 [0.40784314 0.40784314 0.40784314]
 [0.40784314 0.40784314 0.40784314]
 ...
 [0.42745098 0.42745098 0.42745098]
 [0.42745098 0.42745098 0.42745098]
 [0.42745098 0.42745098 0.42745098]]

 [[0.4117647 0.4117647 0.4117647 ]
```

(continues on next page)

(continued from previous page)

```

[0.4117647 0.4117647 0.4117647 ]
[0.4117647 0.4117647 0.4117647 ]
...
[0.42745098 0.42745098 0.42745098]
[0.42745098 0.42745098 0.42745098]
[0.42745098 0.42745098 0.42745098]]

[[0.41960785 0.41960785 0.41960785]
 [0.41568628 0.41568628 0.41568628]
 [0.41568628 0.41568628 0.41568628]
 ...
 [0.43137255 0.43137255 0.43137255]
 [0.43137255 0.43137255 0.43137255]
 [0.43137255 0.43137255 0.43137255]]

...

[[0.4392157 0.4392157 0.4392157 ]
 [0.43529412 0.43529412 0.43529412]
 [0.43137255 0.43137255 0.43137255]
 ...
 [0.45490196 0.45490196 0.45490196]
 [0.4509804 0.4509804 0.4509804 ]
 [0.4509804 0.4509804 0.4509804 ]]

[[0.44313726 0.44313726 0.44313726]
 [0.44313726 0.44313726 0.44313726]
 [0.4392157 0.4392157 0.4392157 ]
 ...
 [0.4509804 0.4509804 0.4509804 ]
 [0.44705883 0.44705883 0.44705883]
 [0.44705883 0.44705883 0.44705883]]

[[0.44313726 0.44313726 0.44313726]
 [0.4509804 0.4509804 0.4509804 ]
 [0.4509804 0.4509804 0.4509804 ]
 ...
 [0.44705883 0.44705883 0.44705883]
 [0.44705883 0.44705883 0.44705883]
 [0.44313726 0.44313726 0.44313726]]]

```

Note the dtype there - float32. Matplotlib has rescaled the 8 bit data from each channel to floating point data between 0.0 and 1.0. As a side note, the only datatype that Pillow can work with is uint8. Matplotlib plotting can handle float32 and uint8, but image reading/writing for any format other than PNG is limited to uint8 data. Why 8 bits? Most displays can only render 8 bits per channel worth of color gradation. Why can they only render 8 bits/channel? Because that's about

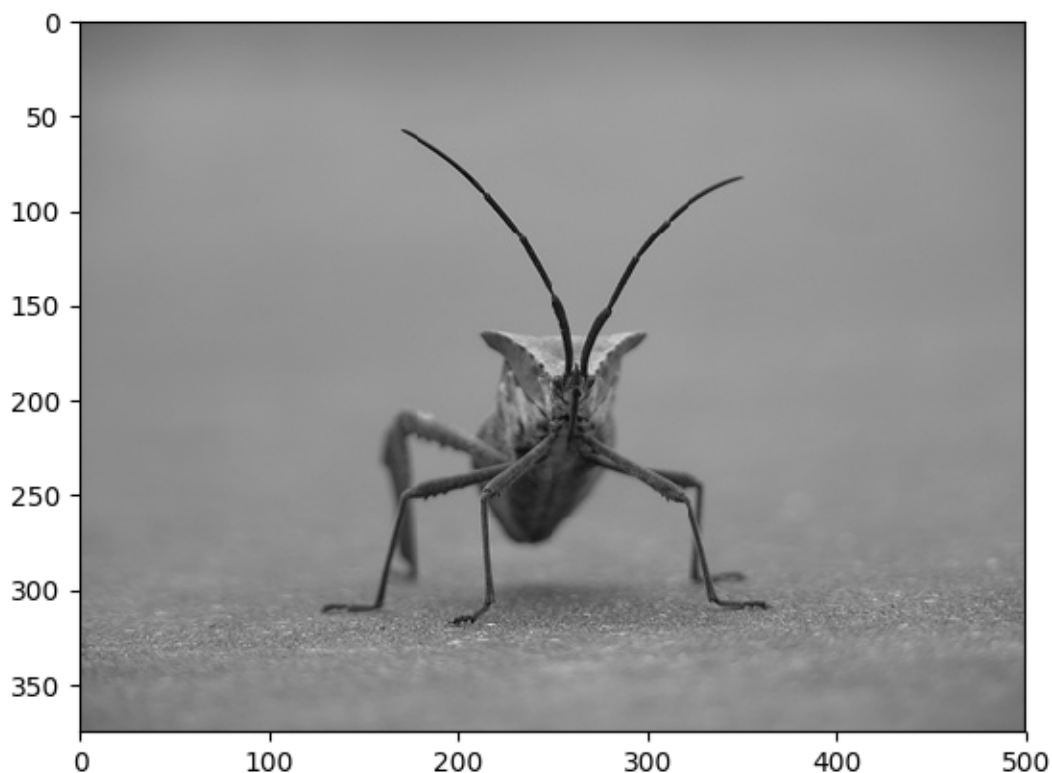
all the human eye can see. More here (from a photography standpoint): [Luminous Landscape bit depth tutorial](#).

Each inner list represents a pixel. Here, with an RGB image, there are 3 values. Since it's a black and white image, R, G, and B are all similar. An RGBA (where A is alpha, or transparency), has 4 values per inner list, and a simple luminance image just has one value (and is thus only a 2-D array, not a 3-D array). For RGB and RGBA images, matplotlib supports float32 and uint8 data types. For grayscale, matplotlib supports only float32. If your array data does not meet one of these descriptions, you need to rescale it.

Plotting numpy arrays as images

So, you have your data in a numpy array (either by importing it, or by generating it). Let's render it. In Matplotlib, this is performed using the `imshow()` function. Here we'll grab the plot object. This object gives you an easy way to manipulate the plot from the prompt.

```
imgplot = plt.imshow(img)
```



You can also plot any numpy array.

Applying pseudocolor schemes to image plots

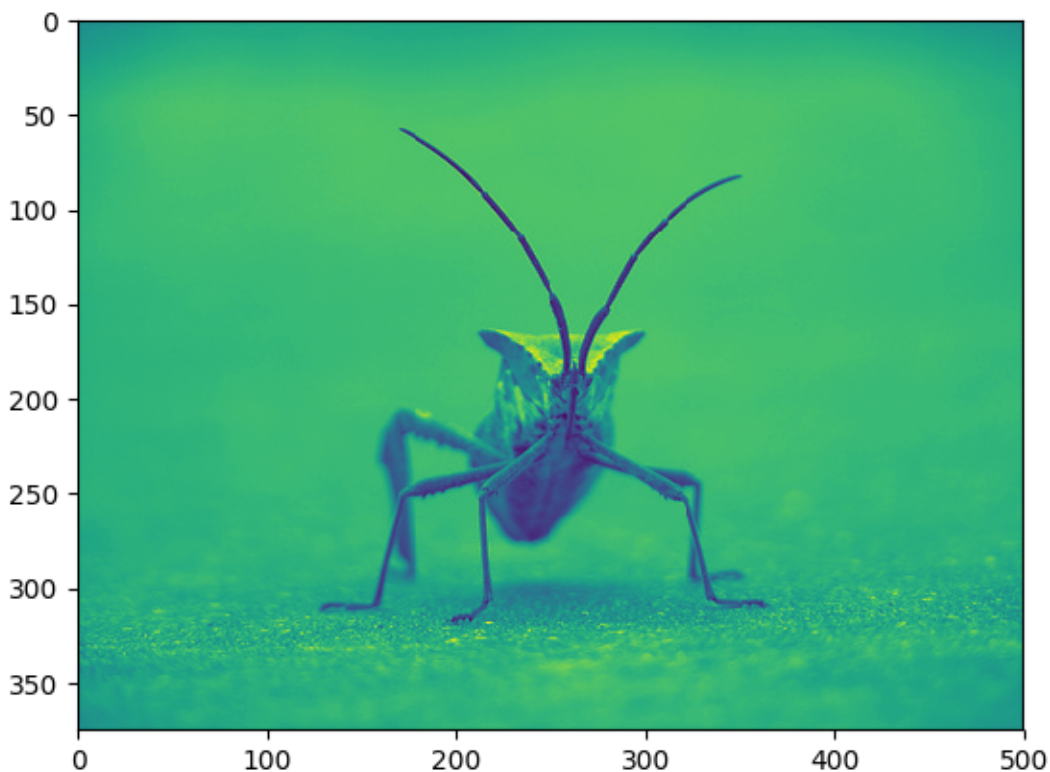
Pseudocolor can be a useful tool for enhancing contrast and visualizing your data more easily. This is especially useful when making presentations of your data using projectors - their contrast is typically quite poor.

Pseudocolor is only relevant to single-channel, grayscale, luminosity images. We currently have an RGB image. Since R, G, and B are all similar (see for yourself above or in your data), we can just pick one channel of our data:

```
lum_img = img[:, :, 0]

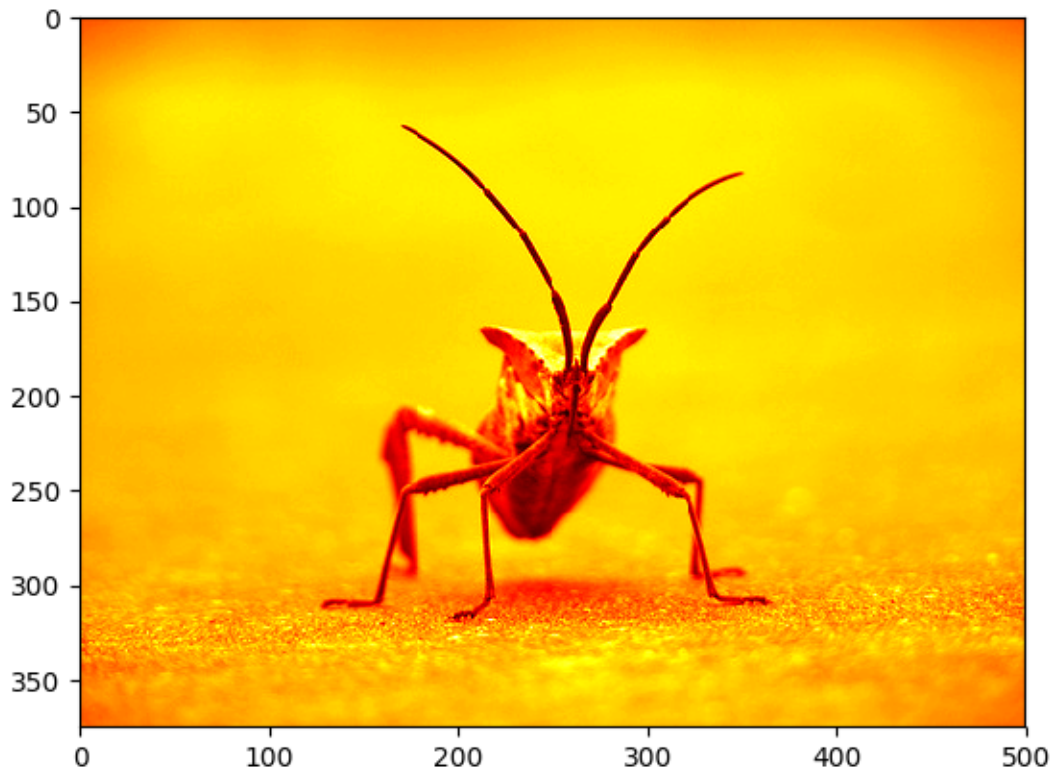
# This is array slicing. You can read more in the `Numpy tutorial
# <https://docs.scipy.org/doc/numpy/user/quickstart.html>`_.

plt.imshow(lum_img)
```



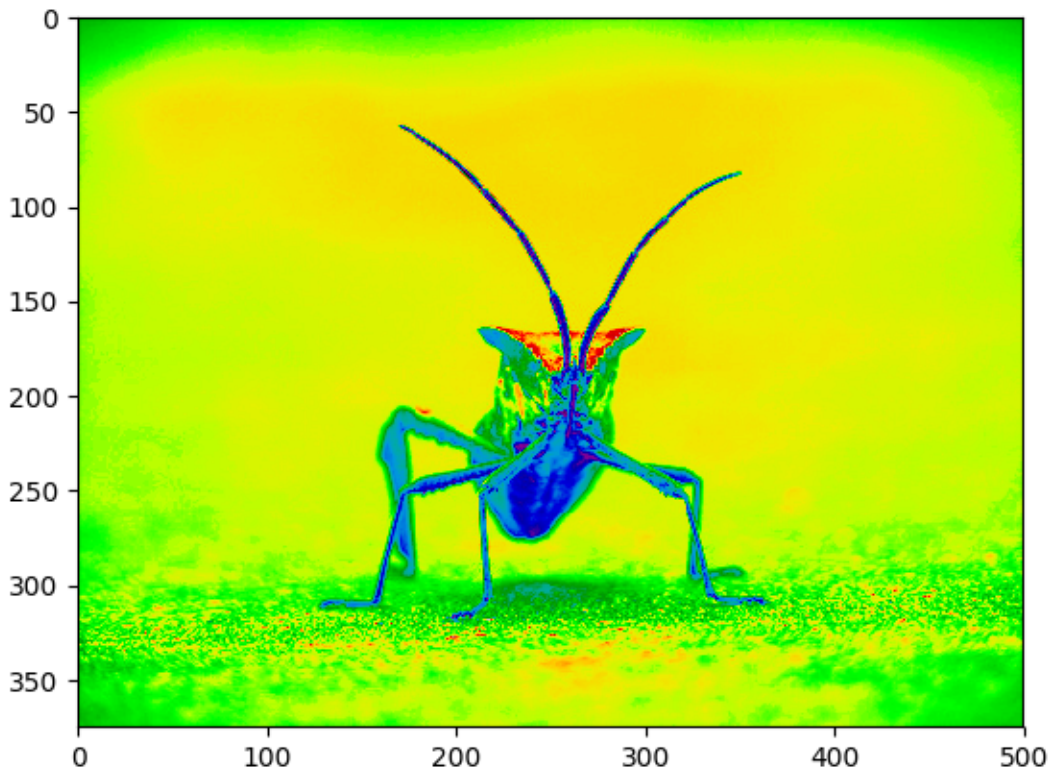
Now, with a luminosity (2D, no color) image, the default colormap (aka lookup table, LUT), is applied. The default is called viridis. There are plenty of others to choose from.

```
plt.imshow(lum_img, cmap="hot")
```



Note that you can also change colormaps on existing plot objects using the `set_cmap()` method:

```
imgplot = plt.imshow(lum_img)
imgplot.set_cmap('nipy_spectral')
```



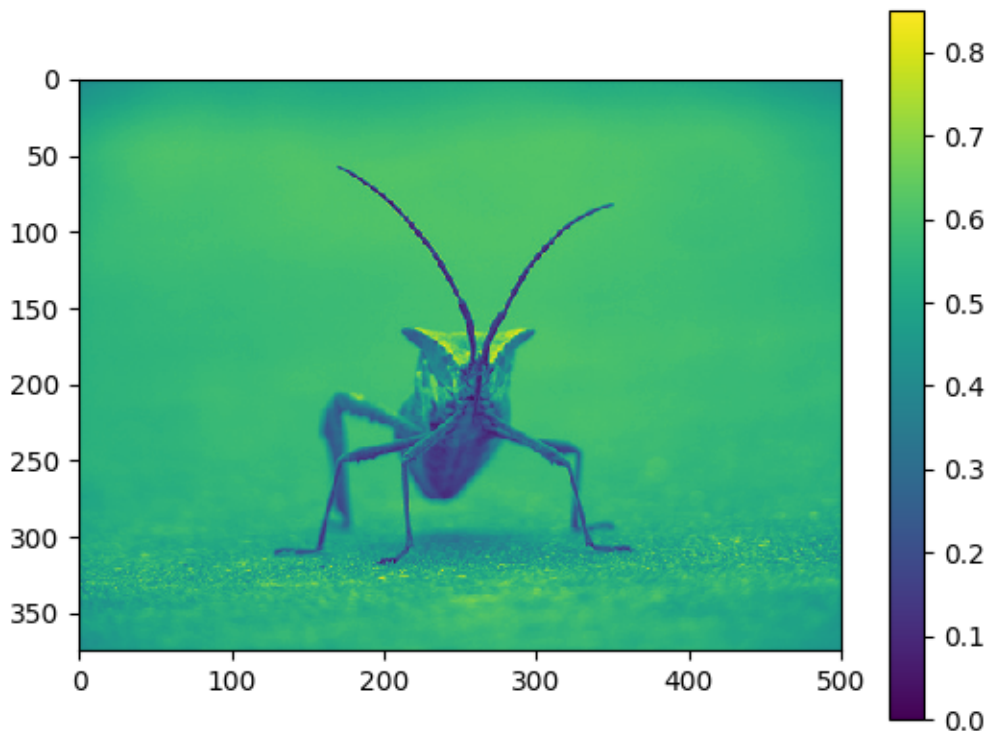
Note: However, remember that in the IPython notebook with the inline backend, you can't make changes to plots that have already been rendered. If you create `imgplot` here in one cell, you cannot call `set_cmap()` on it in a later cell and expect the earlier plot to change. Make sure that you enter these commands together in one cell. `plt` commands will not change plots from earlier cells.

There are many other colormap schemes available. See the [list](#) and [images of the colormaps](#).

Color scale reference

It's helpful to have an idea of what value a color represents. We can do that by adding color bars.

```
imgplot = plt.imshow(lum_img)
plt.colorbar()
```

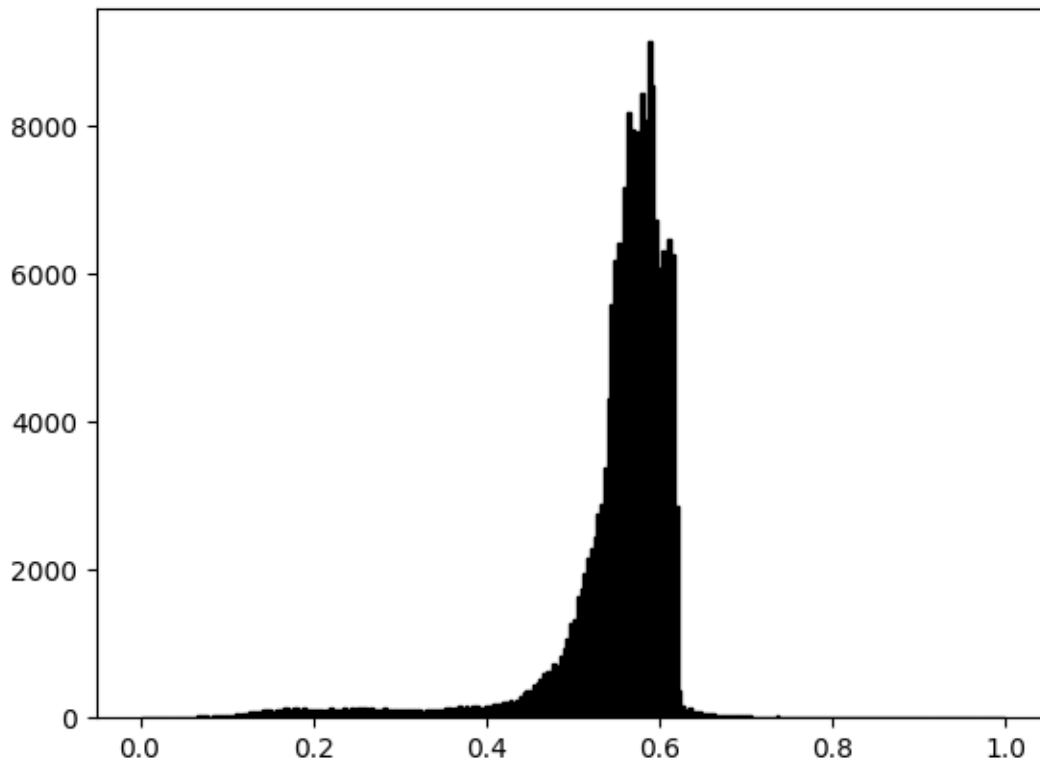


This adds a colorbar to your existing figure. This won't automatically change if you change you switch to a different colormap - you have to re-create your plot, and add in the colorbar again.

Examining a specific data range

Sometimes you want to enhance the contrast in your image, or expand the contrast in a particular region while sacrificing the detail in colors that don't vary much, or don't matter. A good tool to find interesting regions is the histogram. To create a histogram of our image data, we use the `hist()` function.

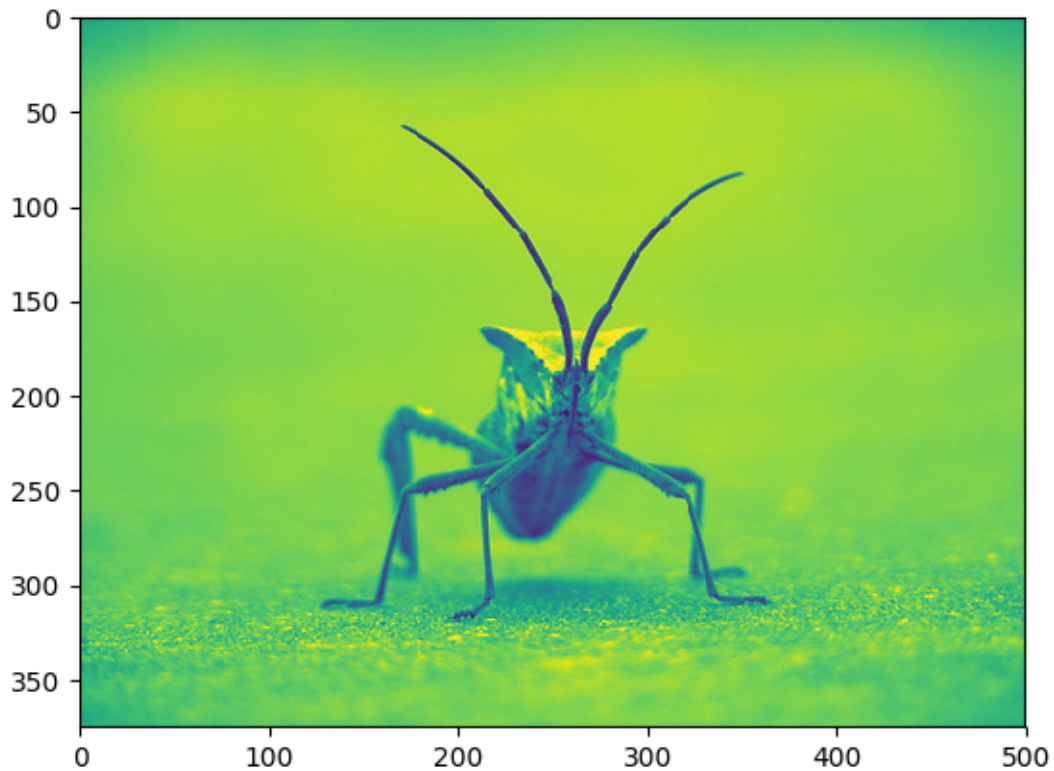
```
plt.hist(lum_img.ravel(), bins=256, range=(0.0, 1.0), fc='k', ec='k')
```



Most often, the "interesting" part of the image is around the peak, and you can get extra contrast by clipping the regions above and/or below the peak. In our histogram, it looks like there's not much useful information in the high end (not many white things in the image). Let's adjust the upper limit, so that we effectively "zoom in on" part of the histogram. We do this by passing the `clim` argument to `imshow`. You could also do this by calling the `set_clim()` method of the image plot object, but make sure that you do so in the same cell as your plot command when working with the IPython Notebook - it will not change plots from earlier cells.

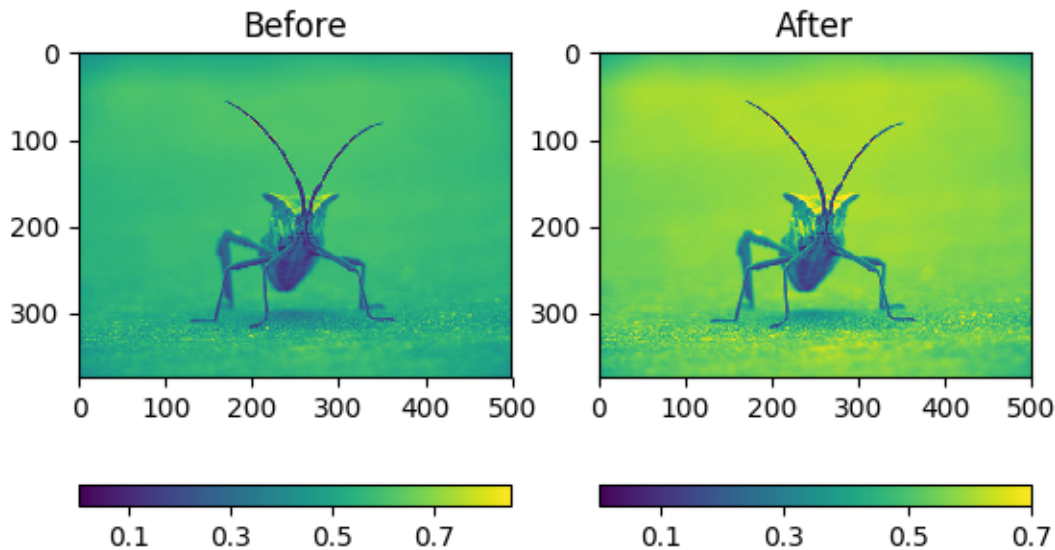
You can specify the `clim` in the call to plot.

```
imgplot = plt.imshow(lum_img, clim=(0.0, 0.7))
```

You can also specify the clim using the returned object

```
fig = plt.figure()
a = fig.add_subplot(1, 2, 1)
imgplot = plt.imshow(lum_img)
a.set_title('Before')
plt.colorbar(ticks=[0.1, 0.3, 0.5, 0.7], orientation='horizontal')
a = fig.add_subplot(1, 2, 2)
imgplot = plt.imshow(lum_img)
imgplot.set_clim(0.0, 0.7)
a.set_title('After')
plt.colorbar(ticks=[0.1, 0.3, 0.5, 0.7], orientation='horizontal')
```



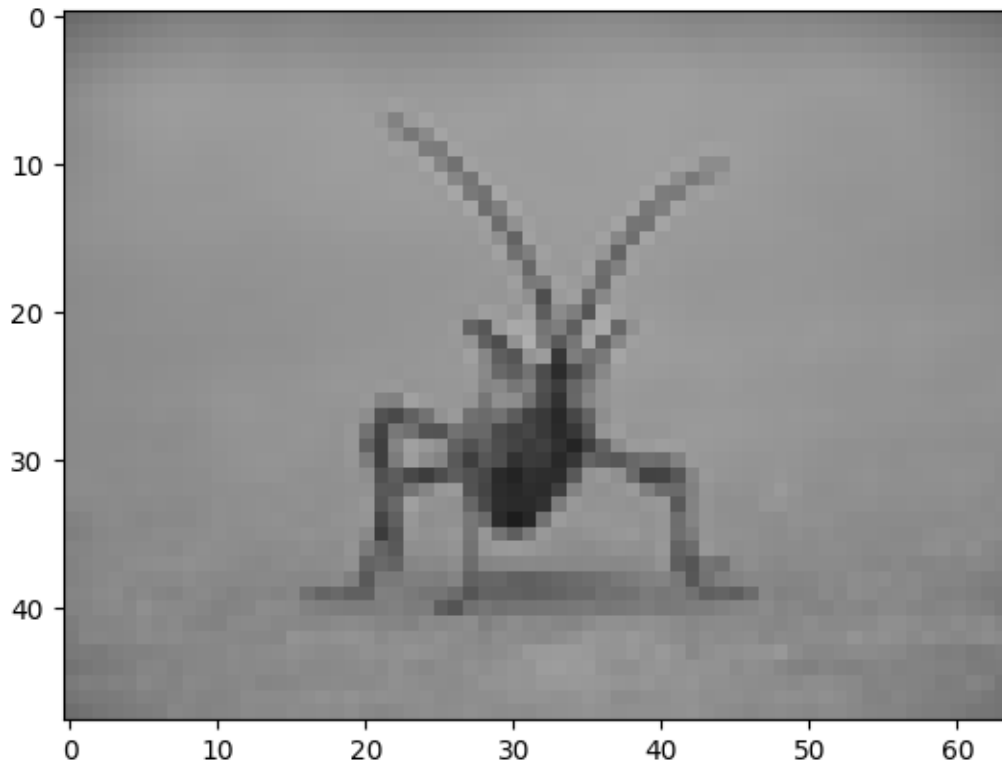
Array Interpolation schemes

Interpolation calculates what the color or value of a pixel "should" be, according to different mathematical schemes. One common place that this happens is when you resize an image. The number of pixels change, but you want the same information. Since pixels are discrete, there's missing space. Interpolation is how you fill that space. This is why your images sometimes come out looking pixelated when you blow them up. The effect is more pronounced when the difference between the original image and the expanded image is greater. Let's take our image and shrink it. We're effectively discarding pixels, only keeping a select few. Now when we plot it, that data gets blown up to the size on your screen. The old pixels aren't there anymore, and the computer has to draw in pixels to fill that space.

We'll use the Pillow library that we used to load the image also to resize the image.

```
from PIL import Image

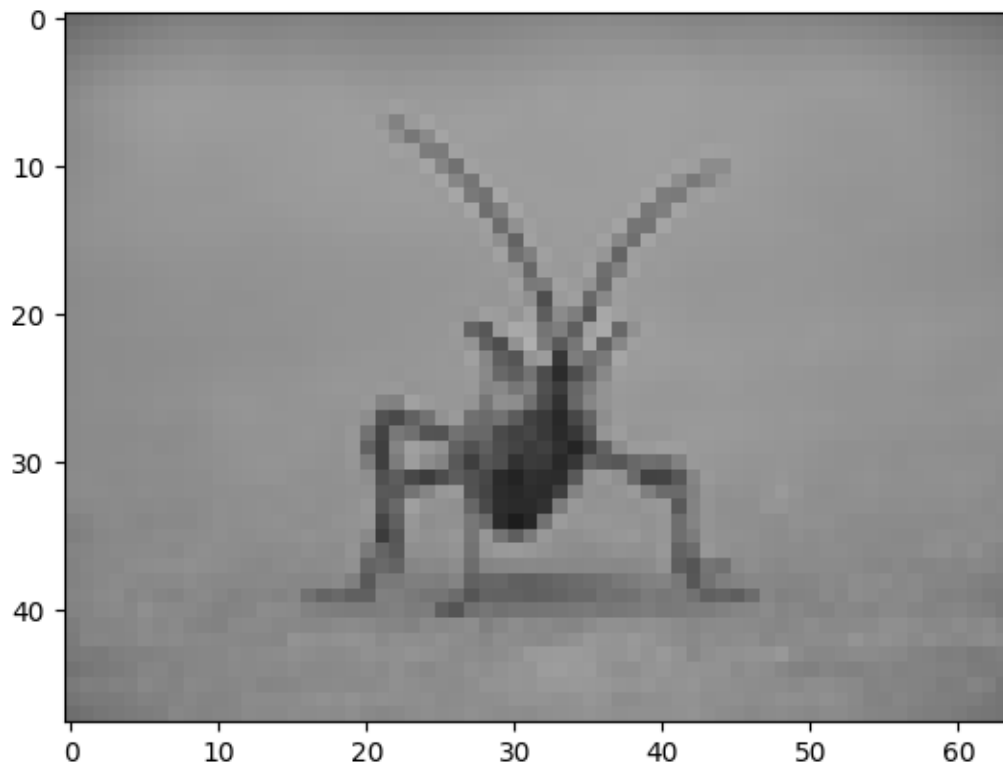
img = Image.open('../doc/_static/stinkbug.png')
img.thumbnail((64, 64), Image.ANTIALIAS) # resizes image in-place
imgplot = plt.imshow(img)
```



Here we have the default interpolation, bilinear, since we did not give `imshow()` any interpolation argument.

Let's try some others. Here's "nearest", which does no interpolation.

```
imgplot = plt.imshow(img, interpolation="nearest")
```



and bicubic:

```
imgplot = plt.imshow(img, interpolation="bicubic")
```



Bicubic interpolation is often used when blowing up photos - people tend to prefer blurry over pixelated.

Note: [Click here to download the full example code](#)

3.1.5 Usage Guide

This tutorial covers some basic usage patterns and best-practices to help you get started with Matplotlib.

General Concepts

matplotlib has an extensive codebase that can be daunting to many new users. However, most of matplotlib can be understood with a fairly simple conceptual framework and knowledge of a few important points.

Plotting requires action on a range of levels, from the most general (e.g., 'contour this 2-D array') to the most specific (e.g., 'color this screen pixel red'). The purpose of a plotting package is to assist you in visualizing your data as easily as possible, with all the necessary control – that is, by

using relatively high-level commands most of the time, and still have the ability to use the low-level commands when needed.

Therefore, everything in matplotlib is organized in a hierarchy. At the top of the hierarchy is the matplotlib "state-machine environment" which is provided by the matplotlib.pyplot module. At this level, simple functions are used to add plot elements (lines, images, text, etc.) to the current axes in the current figure.

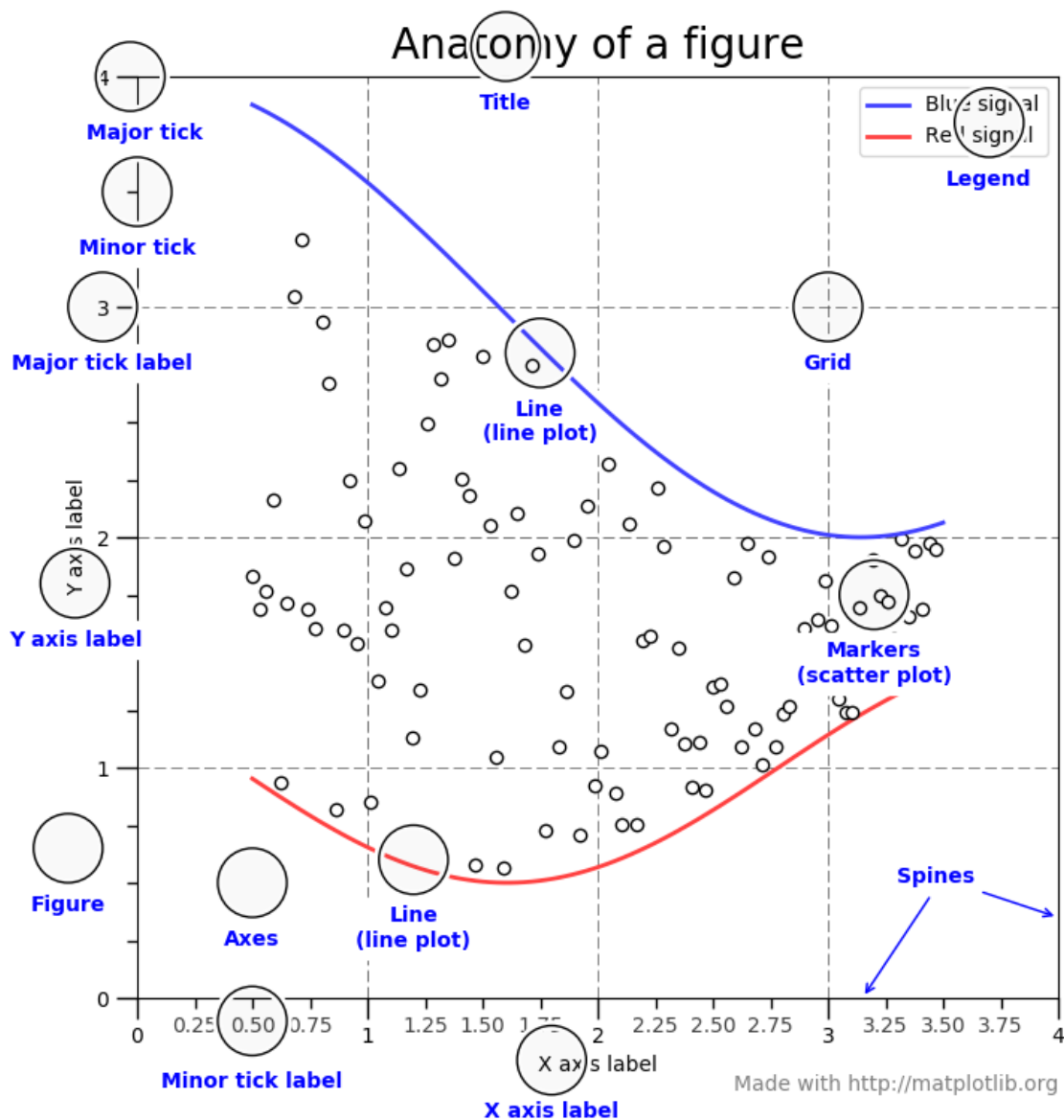
Note: Pyplot's state-machine environment behaves similarly to MATLAB and should be most familiar to users with MATLAB experience.

The next level down in the hierarchy is the first level of the object-oriented interface, in which pyplot is used only for a few functions such as figure creation, and the user explicitly creates and keeps track of the figure and axes objects. At this level, the user uses pyplot to create figures, and through those figures, one or more axes objects can be created. These axes objects are then used for most plotting actions.

For even more control – which is essential for things like embedding matplotlib plots in GUI applications – the pyplot level may be dropped completely, leaving a purely object-oriented approach.

```
# sphinx_gallery_thumbnail_number = 3
import matplotlib.pyplot as plt
import numpy as np
```

Parts of a Figure



Figure

The whole figure. The figure keeps track of all the child Axes, a smattering of 'special' artists (titles, figure legends, etc), and the canvas. (Don't worry too much about the canvas, it is crucial as it is the object that actually does the drawing to get you your plot, but as the user it is more-or-less invisible to you). A figure can have any number of Axes, but to be useful should have at least one.

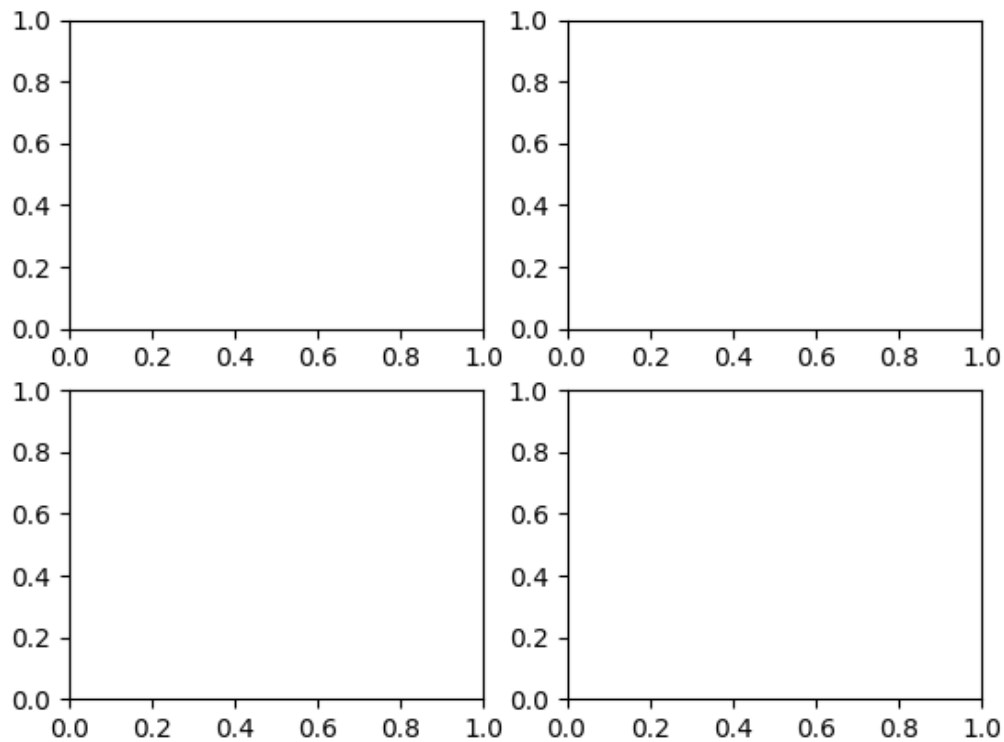
The easiest way to create a new figure is with pyplot:

```
fig = plt.figure() # an empty figure with no axes
fig.suptitle('No axes on this figure') # Add a title so we know which it is

fig, ax_lst = plt.subplots(2, 2) # a figure with a 2x2 grid of Axes
```

No axes on this figure

•



•

Axes

This is what you think of as 'a plot', it is the region of the image with the data space. A given figure can contain many Axes, but a given Axes object can only be in one Figure. The Axes contains two (or three in the case of 3D) Axis objects (be aware of the difference between Axes and Axis) which take care of the data limits (the data limits can also be controlled via set via the `set_xlim()` and `set_ylim()` Axes methods). Each Axes has a title (set via `set_title()`), an x-label (set via `set_xlabel()`), and a y-label set via `set_ylabel()`.

The Axes class and its member functions are the primary entry point to working with the OO interface.

Axis

These are the number-line-like objects. They take care of setting the graph limits and generating the ticks (the marks on the axis) and ticklabels (strings labeling the ticks). The location of the ticks is determined by a Locator object and the ticklabel strings are formatted by a Formatter. The combination of the correct Locator and Formatter gives very fine control over the tick locations and labels.

Artist

Basically everything you can see on the figure is an artist (even the Figure, Axes, and Axis objects). This includes Text objects, Line2D objects, collection objects, Patch objects ... (you get the idea). When the figure is rendered, all of the artists are drawn to the canvas. Most Artists are tied to an Axes; such an Artist cannot be shared by multiple Axes, or moved from one to another.

Types of inputs to plotting functions

All of plotting functions expect `np.array` or `np.ma.masked_array` as input. Classes that are 'array-like' such as `pandas` data objects and `np.matrix` may or may not work as intended. It is best to convert these to `np.array` objects prior to plotting.

For example, to convert a `pandas.DataFrame`

```
a = pandas.DataFrame(np.random.rand(4,5), columns = list('abcde'))
a_asndarray = a.values
```

and to convert a `np.matrix`

```
b = np.matrix([[1,2],[3,4]])
b_asarray = np.asarray(b)
```

Matplotlib, pyplot and pylab: how are they related?

Matplotlib is the whole package; `matplotlib.pyplot` is a module in `matplotlib`; and `pylab` is a module that gets installed alongside `matplotlib`.

Pyplot provides the state-machine interface to the underlying object-oriented plotting library. The state-machine implicitly and automatically creates figures and axes to achieve the desired plot. For example:

```
x = np.linspace(0, 2, 100)

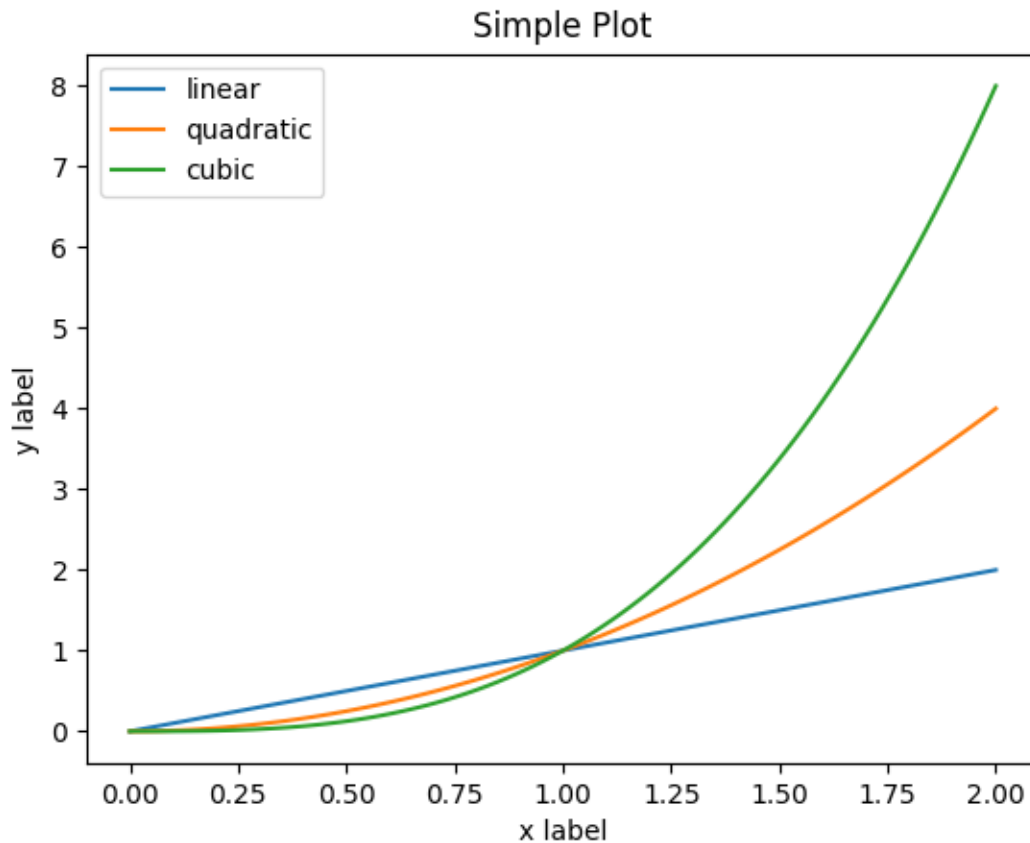
plt.plot(x, x, label='linear')
plt.plot(x, x**2, label='quadratic')
plt.plot(x, x**3, label='cubic')

plt.xlabel('x label')
plt.ylabel('y label')

plt.title("Simple Plot")

plt.legend()

plt.show()
```



The first call to `plt.plot` will automatically create the necessary figure and axes to achieve the desired plot. Subsequent calls to `plt.plot` re-use the current axes and each add another line. Setting the title, legend, and axis labels also automatically use the current axes and set the title, create the legend, and label the axis respectively.

`pylab` is a convenience module that bulk imports `matplotlib.pyplot` (for plotting) and `numpy` (for mathematics and working with arrays) in a single name space. Although many examples use `pylab`, it is no longer recommended.

For non-interactive plotting it is suggested to use `pyplot` to create the figures and then the OO interface for plotting.

Coding Styles

When viewing this documentation and examples, you will find different coding styles and usage patterns. These styles are perfectly valid and have their pros and cons. Just about all of the examples can be converted into another style and achieve the same results. The only caveat is to avoid mixing the coding styles for your own code.

Note: Developers for `matplotlib` have to follow a specific style and guidelines. See [The Matplotlib](#)

Developers' Guide.

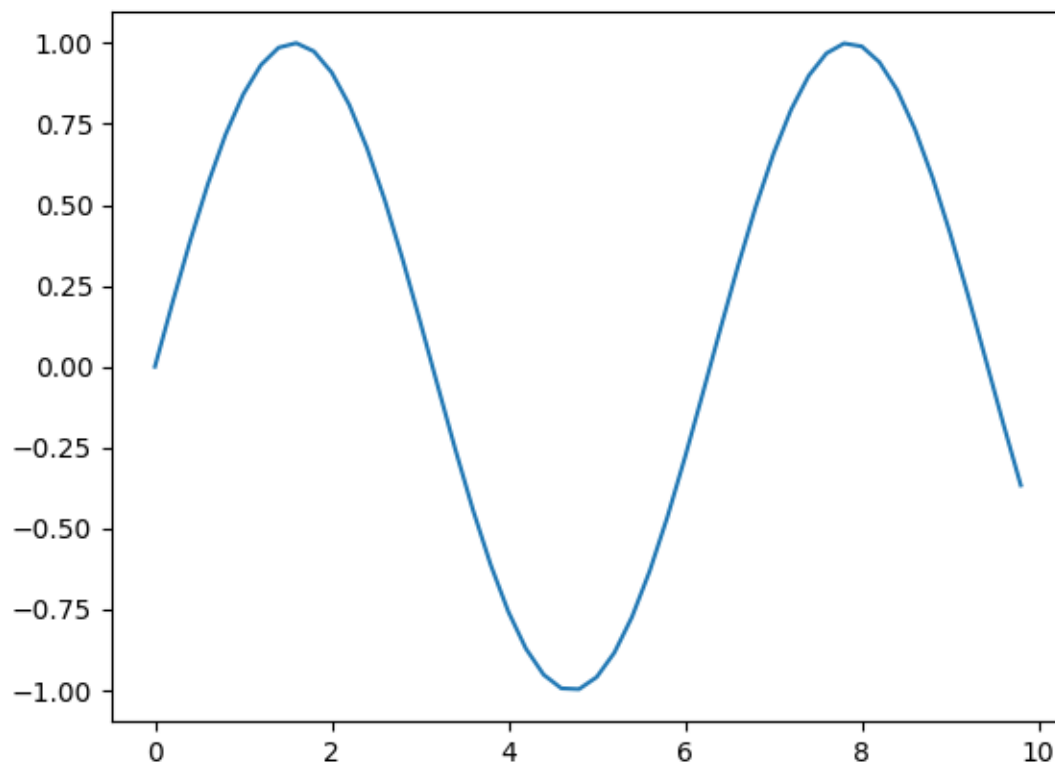
Of the different styles, there are two that are officially supported. Therefore, these are the preferred ways to use matplotlib.

For the pyplot style, the imports at the top of your scripts will typically be:

```
import matplotlib.pyplot as plt
import numpy as np
```

Then one calls, for example, `np.arange`, `np.zeros`, `np.pi`, `plt.figure`, `plt.plot`, `plt.show`, etc. Use the pyplot interface for creating figures, and then use the object methods for the rest:

```
x = np.arange(0, 10, 0.2)
y = np.sin(x)
fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```



So, why all the extra typing instead of the MATLAB-style (which relies on global state and a flat namespace)? For very simple things like this example, the only advantage is academic: the wordier styles are more explicit, more clear as to where things come from and what is going on.

For more complicated applications, this explicitness and clarity becomes increasingly valuable, and the richer and more complete object-oriented interface will likely make the program easier to write and maintain.

Typically one finds oneself making the same plots over and over again, but with different data sets, which leads to needing to write specialized functions to do the plotting. The recommended function signature is something like:

```
def my__plotter(ax, data1, data2, param__dict):  
    """
```

```
    A helper function to make a graph
```

```
    Parameters
```

```
    -----
```

```
    ax : Axes
```

```
        The axes to draw to
```

```
    data1 : array
```

```
        The x data
```

```
    data2 : array
```

```
        The y data
```

```
    param__dict : dict
```

```
        Dictionary of kwargs to pass to ax.plot
```

```
    Returns
```

```
    -----
```

```
    out : list
```

```
        list of artists added
```

```
    """
```

```
    out = ax.plot(data1, data2, **param__dict)
```

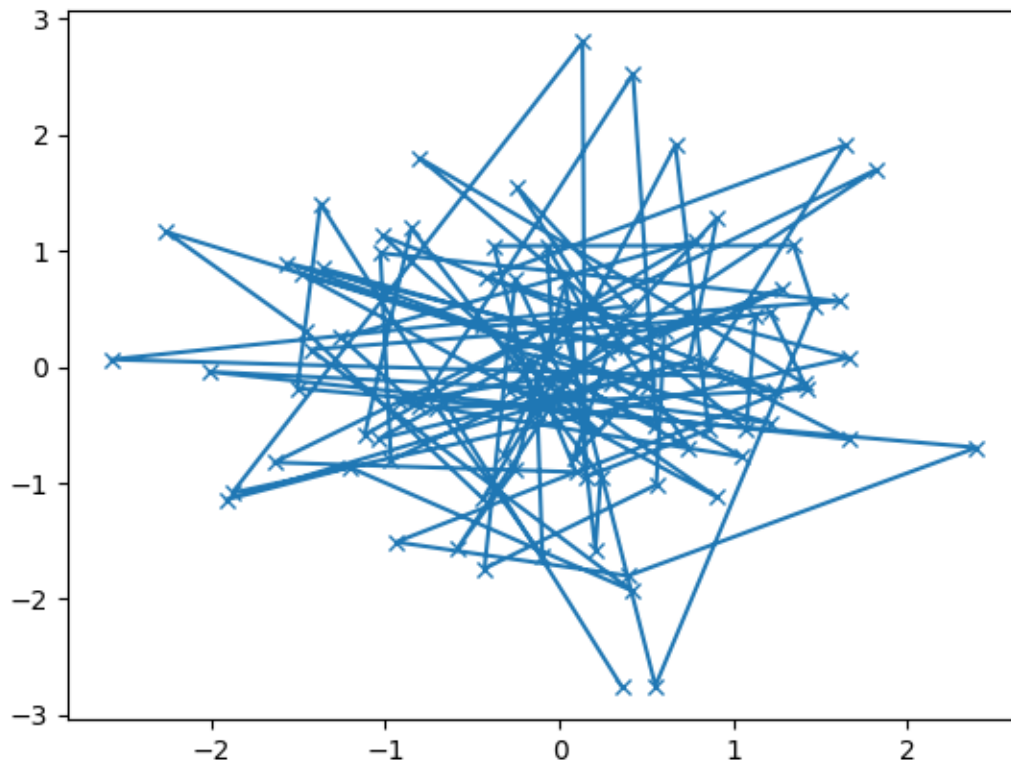
```
    return out
```

```
# which you would then use as:
```

```
data1, data2, data3, data4 = np.random.randn(4, 100)
```

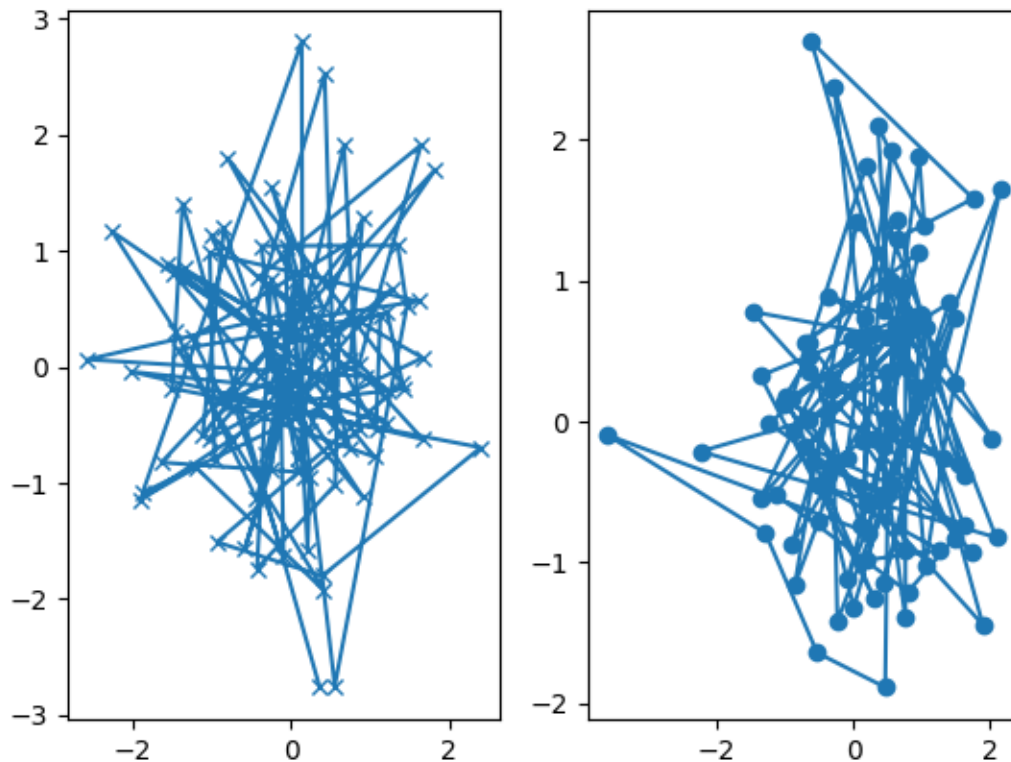
```
fig, ax = plt.subplots(1, 1)
```

```
my__plotter(ax, data1, data2, {'marker': 'x'})
```



or if you wanted to have 2 sub-plots:

```
fig, (ax1, ax2) = plt.subplots(1, 2)
my_plotter(ax1, data1, data2, {'marker': 'x'})
my_plotter(ax2, data3, data4, {'marker': 'o'})
```



Again, for these simple examples this style seems like overkill, however once the graphs get slightly more complex it pays off.

Backends

What is a backend?

A lot of documentation on the website and in the mailing lists refers to the "backend" and many new users are confused by this term. `matplotlib` targets many different use cases and output formats. Some people use `matplotlib` interactively from the python shell and have plotting windows pop up when they type commands. Some people embed `matplotlib` into graphical user interfaces like `wxpython` or `pygtk` to build rich applications. Others use `matplotlib` in batch scripts to generate postscript images from some numerical simulations, and still others in web application servers to dynamically serve up graphs.

To support all of these use cases, `matplotlib` can target different outputs, and each of these capabilities is called a backend; the "frontend" is the user facing code, i.e., the plotting code, whereas the "backend" does all the hard work behind-the-scenes to make the figure. There are two types of backends: user interface backends (for use in `pygtk`, `wxpython`, `tkinter`, `qt4`, or `macosx`; also referred to as "interactive backends") and hardcopy backends to make image files (PNG, SVG, PDF, PS; also referred to as "non-interactive backends").

There are four ways to configure your backend. If they conflict each other, the method mentioned last in the following list will be used, e.g. calling `use()` will override the setting in your `matplotlibrc`.

1. The backend parameter in your `matplotlibrc` file (see [Customizing Matplotlib with style sheets and rcParams](#)):

```
backend : WXAgg  # use wxpython with antigrain (agg) rendering
```

2. Setting the `MPLBACKEND` environment variable, either for your current shell or for a single script. On Unix:

```
> export MPLBACKEND=module://my_backend
> python simple_plot.py

> MPLBACKEND="module://my_backend" python simple_plot.py
```

On Windows, only the former is possible:

```
> set MPLBACKEND=module://my_backend
> python simple_plot.py
```

Setting this environment variable will override the backend parameter in any `matplotlibrc`, even if there is a `matplotlibrc` in your current working directory. Therefore setting `MPLBACKEND` globally, e.g. in your `.bashrc` or `.profile`, is discouraged as it might lead to counter-intuitive behavior.

3. If your script depends on a specific backend you can use the `use()` function:

```
import matplotlib
matplotlib.use('PS')  # generate postscript output by default
```

If you use the `use()` function, this must be done before importing `matplotlib.pyplot`. Calling `use()` after `pyplot` has been imported will have no effect. Using `use()` will require changes in your code if users want to use a different backend. Therefore, you should avoid explicitly calling `use()` unless absolutely necessary.

Note: Backend name specifications are not case-sensitive; e.g., `'GTKAgg'` and `'gtkagg'` are equivalent.

With a typical installation of `matplotlib`, such as from a binary installer or a linux distribution package, a good default backend will already be set, allowing both interactive work and plotting from scripts, with output to the screen and/or to a file, so at least initially you will not need to use any of the methods given above.

If, however, you want to write graphical user interfaces, or a web application server (`Matplotlib` in a web application server), or need a better understanding of what is going on, read on. To make things a little more customizable for graphical user interfaces, `matplotlib` separates the concept of the renderer (the thing that actually does the drawing) from the canvas (the place where the drawing goes). The canonical renderer for user interfaces is `Agg` which uses the [Anti-Grain Geometry C++](#)

library to make a raster (pixel) image of the figure. All of the user interfaces except macosx can be used with agg rendering, e.g., WXAgg, GTKAgg, QT4Agg, QT5Agg, TkAgg. In addition, some of the user interfaces support other rendering engines. For example, with GTK, you can also select GDK rendering (backend GTK deprecated in 2.0) or Cairo rendering (backend GTKCairo).

For the rendering engines, one can also distinguish between [vector](#) or [raster](#) renderers. Vector graphics languages issue drawing commands like "draw a line from this point to this point" and hence are scale free, and raster backends generate a pixel representation of the line whose accuracy depends on a DPI setting.

Here is a summary of the matplotlib renderers (there is an eponymous backend for each; these are non-interactive backends, capable of writing to a file):

Renderer	Filetypes	Description
AGG	png	raster graphics – high quality images using the Anti-Grain Geometry engine
PS	ps eps	vector graphics – Postscript output
PDF	pdf	vector graphics – Portable Document Format
SVG	svg	vector graphics – Scalable Vector Graphics
Cairo	png ps pdf svg	raster graphics and vector graphics – using the Cairo graphics library

And here are the user interfaces and renderer combinations supported; these are interactive backends, capable of displaying to the screen and of using appropriate renderers from the table above to write to a file:

Back-end	Description
Qt5Agg	Agg rendering in a Qt5 canvas (requires PyQt5). This backend can be activated in IPython with <code>%matplotlib qt5</code> .
ipympl	Agg rendering embedded in a Jupyter widget. (requires <code>ipympl</code>). This backend can be enabled in a Jupyter notebook with <code>%matplotlib ipympl</code> .
GTK3Agg	Agg rendering to a GTK 3.x canvas (requires PyGObject , and pycairo or cairocffi). This backend can be activated in IPython with <code>%matplotlib gtk3</code> .
macosx	Agg rendering into a Cocoa canvas in OSX. This backend can be activated in IPython with <code>%matplotlib osx</code> .
TkAgg	Agg rendering to a Tk canvas (requires TkInter). This backend can be activated in IPython with <code>%matplotlib tk</code> .
nbAgg	Embed an interactive figure in a Jupyter classic notebook. This backend can be enabled in Jupyter notebooks via <code>%matplotlib notebook</code> .
WebAgg	On <code>show()</code> will start a tornado server with an interactive figure.
GTK3Cairo	Cairo rendering to a GTK 3.x canvas (requires PyGObject , and pycairo or cairocffi).
Qt4Agg	Agg rendering to a Qt4 canvas (requires PyQt4 or <code>pyside</code>). This backend can be activated in IPython with <code>%matplotlib qt4</code> .
GTK-Agg	Agg rendering to a GTK 2.x canvas (requires PyGTK , and pycairo or cairocffi ; Python2 only). This backend can be activated in IPython with <code>%matplotlib gtk</code> .
GTK-Cairo	Cairo rendering to a GTK 2.x canvas (requires PyGTK , and pycairo or cairocffi ; Python2 only).
WX-Agg	Agg rendering to a wxWidgets canvas (requires wxPython ; v4.0 (in beta) is required for Python3). This backend can be activated in IPython with <code>%matplotlib wx.#</code>

ipympl

The Jupyter widget ecosystem is moving too fast to support directly in Matplotlib. To install `ipympl`

```
pip install ipympl
jupyter nbextension enable --py --sys-prefix ipympl
```

or

```
conda install ipympl -c conda-forge
```

See [jupyter-matplotlib](#) for more details.

GTK and Cairo

Both GTK2 and GTK3 have implicit dependencies on PyCairo regardless of the specific Matplotlib backend used. Unfortunately the latest release of PyCairo for Python3 does not implement the

Python wrappers needed for the GTK3Agg backend. Cairoffi can be used as a replacement which implements the correct wrapper.

How do I select PyQt4 or PySide?

The `QT_API` environment variable can be set to either `pyqt` or `pyside` to use PyQt4 or PySide, respectively.

Since the default value for the bindings to be used is PyQt4, matplotlib first tries to import it, if the import fails, it tries to import PySide.

What is interactive mode?

Use of an interactive backend (see [What is a backend?](#)) permits—but does not by itself require or ensure—plotting to the screen. Whether and when plotting to the screen occurs, and whether a script or shell session continues after a plot is drawn on the screen, depends on the functions and methods that are called, and on a state variable that determines whether matplotlib is in “interactive mode”. The default Boolean value is set by the `matplotlibrc` file, and may be customized like any other configuration parameter (see [Customizing Matplotlib with style sheets and rcParams](#)). It may also be set via `matplotlib.interactive()`, and its value may be queried via `matplotlib.is_interactive()`. Turning interactive mode on and off in the middle of a stream of plotting commands, whether in a script or in a shell, is rarely needed and potentially confusing, so in the following we will assume all plotting is done with interactive mode either on or off.

Note: Major changes related to interactivity, and in particular the role and behavior of `show()`, were made in the transition to matplotlib version 1.0, and bugs were fixed in 1.0.1. Here we describe the version 1.0.1 behavior for the primary interactive backends, with the partial exception of `macosx`.

Interactive mode may also be turned on via `matplotlib.pyplot.ion()`, and turned off via `matplotlib.pyplot.ioff()`.

Note: Interactive mode works with suitable backends in `ipython` and in the ordinary python shell, but it does not work in the IDLE IDE. If the default backend does not support interactivity, an interactive backend can be explicitly activated using any of the methods discussed in [What is a backend?](#).

Interactive example

From an ordinary python prompt, or after invoking `ipython` with no options, try this:

```
import matplotlib.pyplot as plt
plt.ion()
plt.plot([1.6, 2.7])
```

Assuming you are running version 1.0.1 or higher, and you have an interactive backend installed and selected by default, you should see a plot, and your terminal prompt should also be active; you can type additional commands such as:

```
plt.title("interactive test")
plt.xlabel("index")
```

and you will see the plot being updated after each line. Since version 1.5, modifying the plot by other means should also automatically update the display on most backends. Get a reference to the Axes instance, and call a method of that instance:

```
ax = plt.gca()
ax.plot([3.1, 2.2])
```

If you are using certain backends (like macosx), or an older version of matplotlib, you may not see the new line added to the plot immediately. In this case, you need to explicitly call `draw()` in order to update the plot:

```
plt.draw()
```

Non-interactive example

Start a fresh session as in the previous example, but now turn interactive mode off:

```
import matplotlib.pyplot as plt
plt.ioff()
plt.plot([1.6, 2.7])
```

Nothing happened—or at least nothing has shown up on the screen (unless you are using macosx backend, which is anomalous). To make the plot appear, you need to do this:

```
plt.show()
```

Now you see the plot, but your terminal command line is unresponsive; the `show()` command blocks the input of additional commands until you manually kill the plot window.

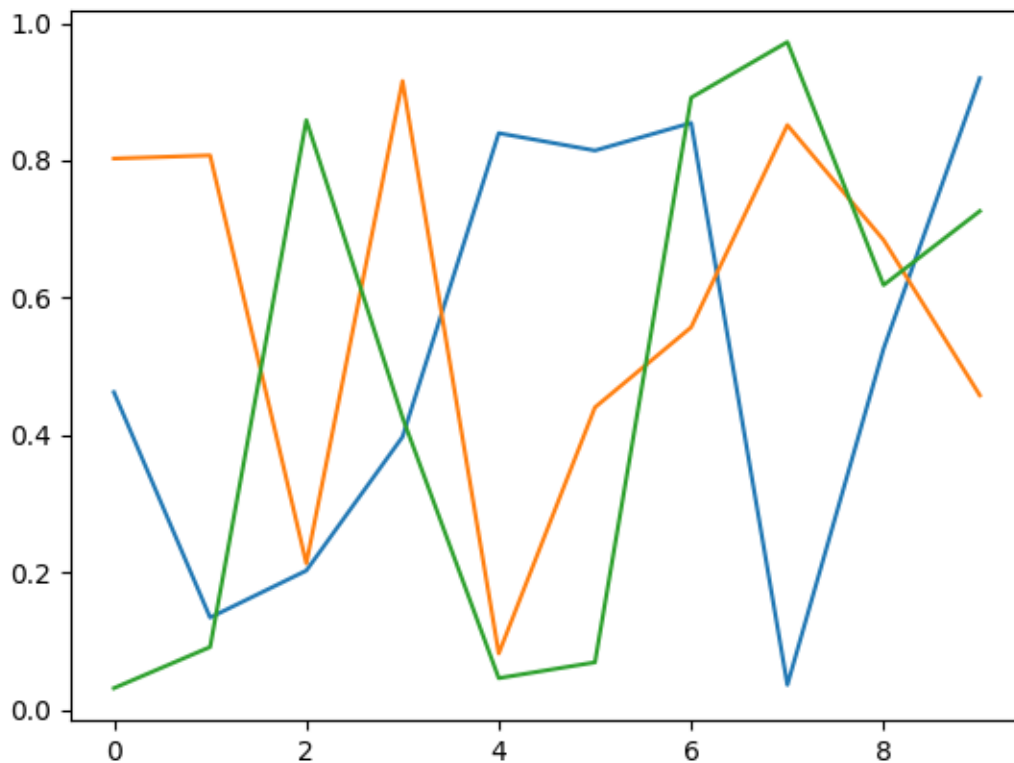
What good is this—being forced to use a blocking function? Suppose you need a script that plots the contents of a file to the screen. You want to look at that plot, and then end the script. Without some blocking command such as `show()`, the script would flash up the plot and then end immediately, leaving nothing on the screen.

In addition, non-interactive mode delays all drawing until `show()` is called; this is more efficient than redrawing the plot each time a line in the script adds a new feature.

Prior to version 1.0, `show()` generally could not be called more than once in a single script (although sometimes one could get away with it); for version 1.0.1 and above, this restriction is lifted, so one can write a script like this:

```
import numpy as np
import matplotlib.pyplot as plt

plt.ioff()
for i in range(3):
    plt.plot(np.random.rand(10))
    plt.show()
```



which makes three plots, one at a time.

Summary

In interactive mode, pyplot functions automatically draw to the screen.

When plotting interactively, if using object method calls in addition to pyplot functions, then call `draw()` whenever you want to refresh the plot.

Use non-interactive mode in scripts in which you want to generate one or more figures and display

them before ending or generating a new set of figures. In that case, use `show()` to display the figure(s) and to block execution until you have manually destroyed them.

Performance

Whether exploring data in interactive mode or programmatically saving lots of plots, rendering performance can be a painful bottleneck in your pipeline. Matplotlib provides a couple ways to greatly reduce rendering time at the cost of a slight change (to a settable tolerance) in your plot's appearance. The methods available to reduce rendering time depend on the type of plot that is being created.

Line segment simplification

For plots that have line segments (e.g. typical line plots, outlines of polygons, etc.), rendering performance can be controlled by the `path.simplify` and `path.simplify_threshold` parameters in your `matplotlibrc` file (see [Customizing Matplotlib with style sheets and rcParams](#) for more information about the `matplotlibrc` file). The `path.simplify` parameter is a boolean indicating whether or not line segments are simplified at all. The `path.simplify_threshold` parameter controls how much line segments are simplified; higher thresholds result in quicker rendering.

The following script will first display the data without any simplification, and then display the same data with simplification. Try interacting with both of them:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl

# Setup, and create the data to plot
y = np.random.rand(100000)
y[50000:] *= 2
y[np.logspace(1, np.log10(50000), 400).astype(int)] = -1
mpl.rcParams['path.simplify'] = True

mpl.rcParams['path.simplify_threshold'] = 0.0
plt.plot(y)
plt.show()

mpl.rcParams['path.simplify_threshold'] = 1.0
plt.plot(y)
plt.show()
```

Matplotlib currently defaults to a conservative simplification threshold of $1/9$. If you want to change your default settings to use a different value, you can change your `matplotlibrc` file. Alternatively, you could create a new style for interactive plotting (with maximal simplification) and another style for publication quality plotting (with minimal simplification) and activate them as necessary. See [Customizing Matplotlib with style sheets and rcParams](#) for instructions on how to perform these actions.

The simplification works by iteratively merging line segments into a single vector until the next line segment's perpendicular distance to the vector (measured in display-coordinate space) is greater than the `path.simplify_threshold` parameter.

Note: Changes related to how line segments are simplified were made in version 2.1. Rendering time will still be improved by these parameters prior to 2.1, but rendering time for some kinds of data will be vastly improved in versions 2.1 and greater.

Marker simplification

Markers can also be simplified, albeit less robustly than line segments. Marker simplification is only available to Line2D objects (through the `markevery` property). Wherever Line2D construction parameter are passed through, such as `matplotlib.pyplot.plot()` and `matplotlib.axes.Axes.plot()`, the `markevery` parameter can be used:

```
plt.plot(x, y, markevery=10)
```

The `markevery` argument allows for naive subsampling, or an attempt at evenly spaced (along the x axis) sampling. See the [/gallery/lines_bars_and_markers/markevery_demo](#) for more information.

Splitting lines into smaller chunks

If you are using the Agg backend (see [What is a backend?](#)), then you can make use of the `agg.path.chunksize` rc parameter. This allows you to specify a chunk size, and any lines with greater than that many vertices will be split into multiple lines, each of which have no more than `agg.path.chunksize` many vertices. (Unless `agg.path.chunksize` is zero, in which case there is no chunking.) For some kind of data, chunking the line up into reasonable sizes can greatly decrease rendering time.

The following script will first display the data without any chunk size restriction, and then display the same data with a chunk size of 10,000. The difference can best be seen when the figures are large, try maximizing the GUI and then interacting with them:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['path.simplify_threshold'] = 1.0

# Setup, and create the data to plot
y = np.random.rand(100000)
y[50000:] *= 2
y[np.logspace(1, np.log10(50000), 400).astype(int)] = -1
mpl.rcParams['path.simplify'] = True
```

(continues on next page)

(continued from previous page)

```
mpl.rcParams['agg.path.chunksize'] = 0
plt.plot(y)
plt.show()

mpl.rcParams['agg.path.chunksize'] = 10000
plt.plot(y)
plt.show()
```

Using the fast style

The fast style can be used to automatically set simplification and chunking parameters to reasonable settings to speed up plotting large amounts of data. It can be used simply by running:

```
import matplotlib.style as mplstyle
mplstyle.use('fast')
```

It is very light weight, so it plays nicely with other styles, just make sure the fast style is applied last so that other styles do not overwrite the settings:

```
mplstyle.use(['dark_background', 'ggplot', 'fast'])
```

Note: [Click here to download the full example code](#)

3.1.6 Pyplot tutorial

An introduction to the pyplot interface.

Intro to pyplot

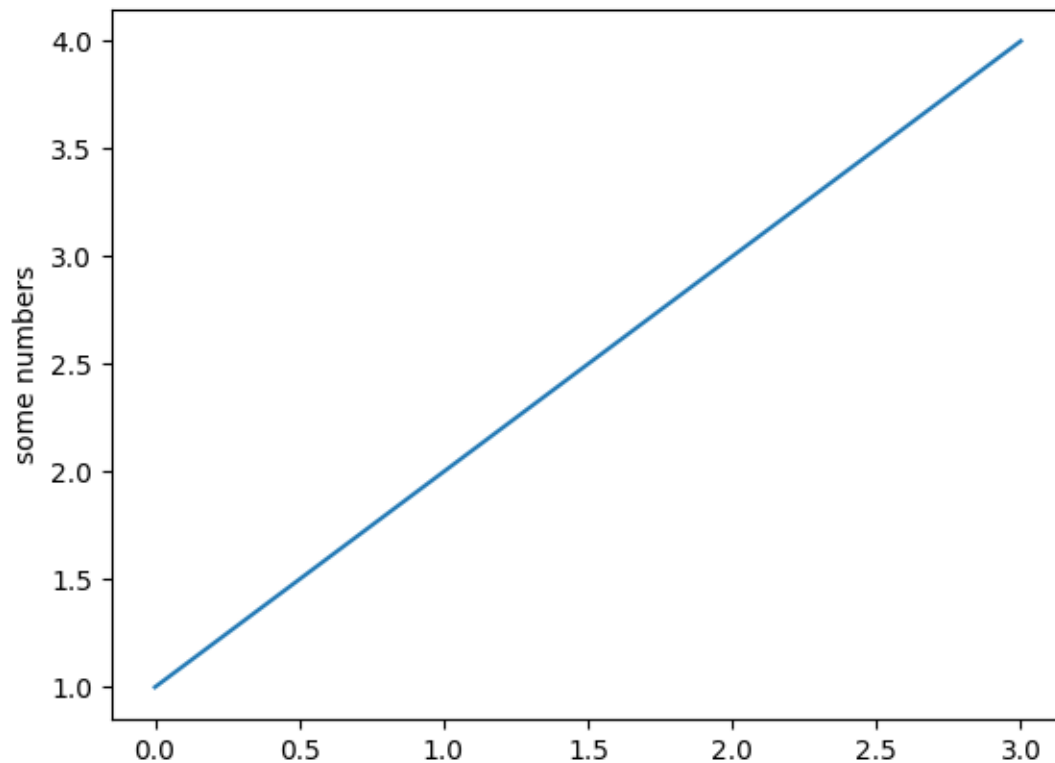
matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that "axes" here and in most places in the documentation refers to the axes *part of a figure* and not the strict mathematical term for more than one axis).

Note: the pyplot API is generally less-flexible than the object-oriented API. Most of the function calls you see here can also be called as methods from an Axes object. We recommend browsing the tutorials and examples to see how this works.

Generating visualizations with pyplot is very quick:

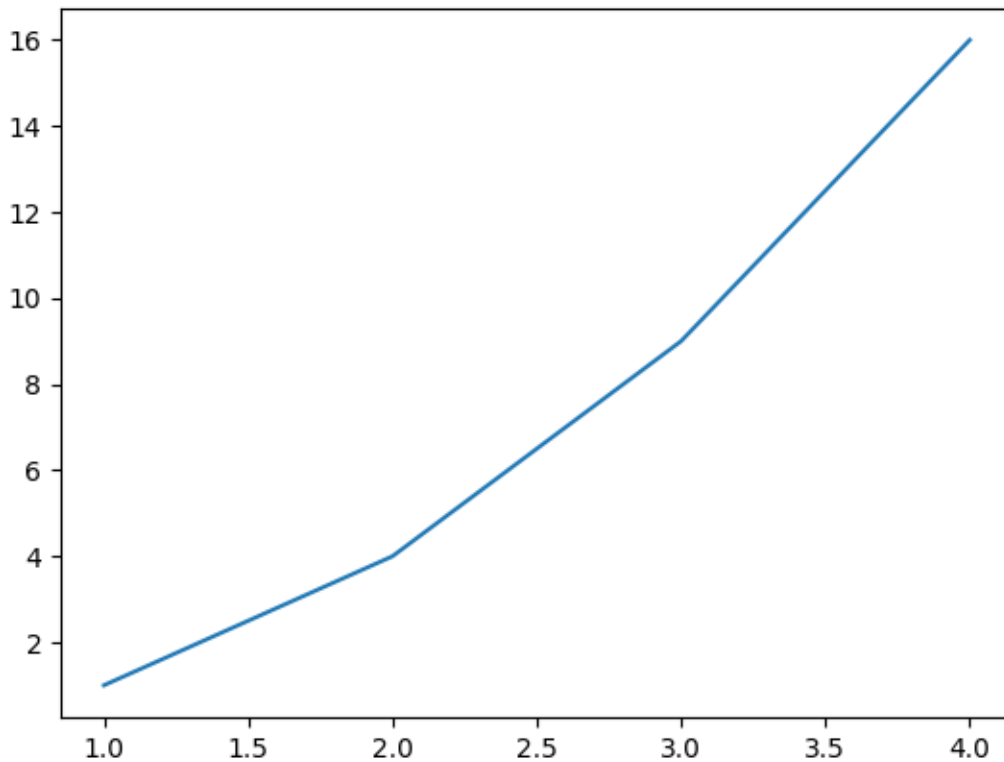
```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```



You may be wondering why the x-axis ranges from 0-3 and the y-axis from 1-4. If you provide a single list or array to the `plot()` command, matplotlib assumes it is a sequence of y values, and automatically generates the x values for you. Since python ranges start with 0, the default x vector has the same length as y but starts with 0. Hence the x data are `[0,1,2,3]`.

`plot()` is a versatile command, and will take an arbitrary number of arguments. For example, to plot x versus y, you can issue the command:

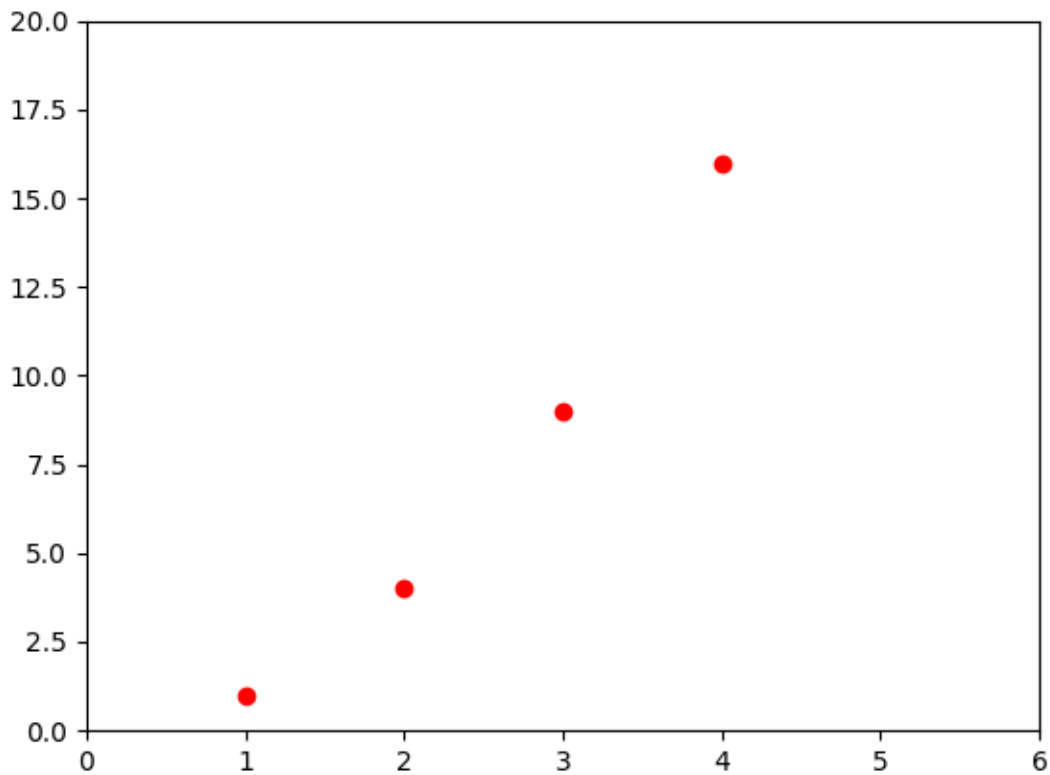
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
```



Formatting the style of your plot

For every x, y pair of arguments, there is an optional third argument which is the format string that indicates the color and line type of the plot. The letters and symbols of the format string are from MATLAB, and you concatenate a color string with a line style string. The default format string is 'b-', which is a solid blue line. For example, to plot the above with red circles, you would issue

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```



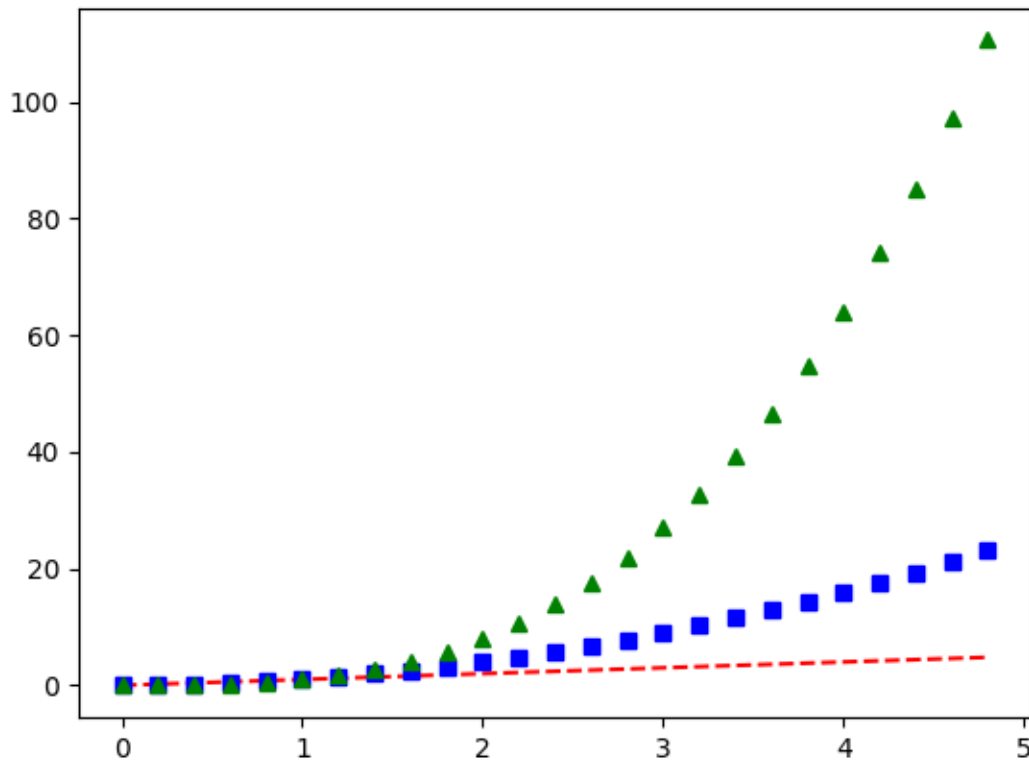
See the `plot()` documentation for a complete list of line styles and format strings. The `axis()` command in the example above takes a list of `[xmin, xmax, ymin, ymax]` and specifies the viewport of the axes.

If matplotlib were limited to working with lists, it would be fairly useless for numeric processing. Generally, you will use `numpy` arrays. In fact, all sequences are converted to numpy arrays internally. The example below illustrates a plotting several lines with different format styles in one command using arrays.

```
import numpy as np

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



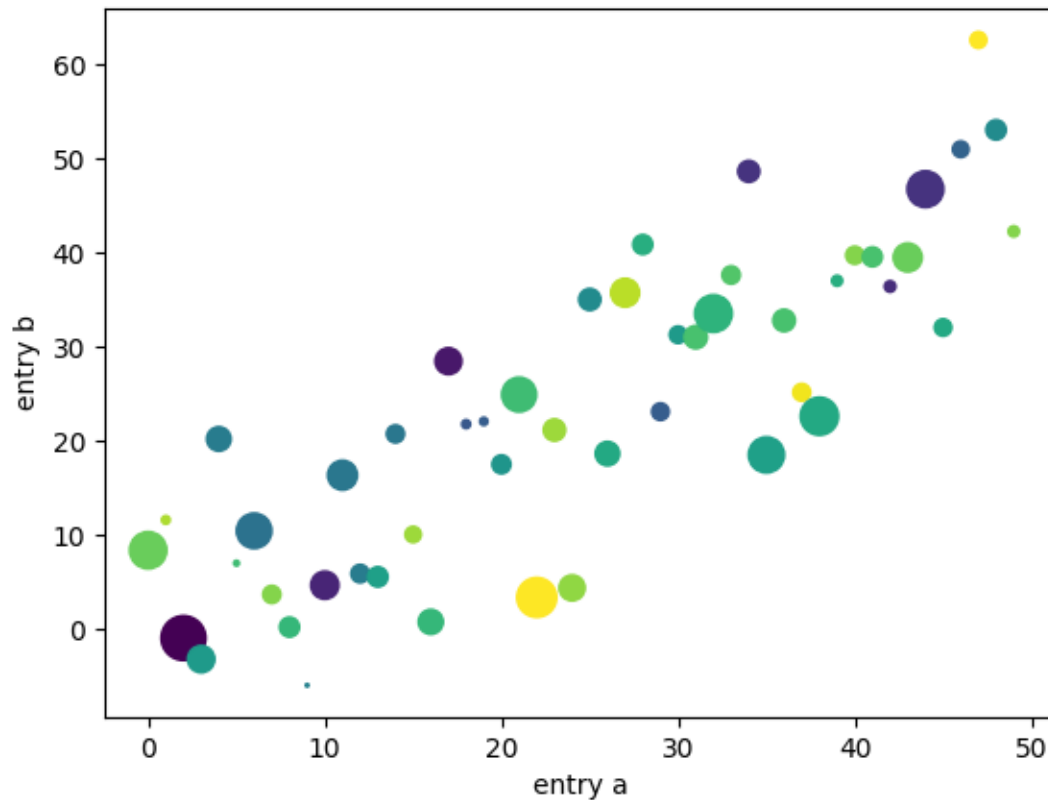
Plotting with keyword strings

There are some instances where you have data in a format that lets you access particular variables with strings. For example, with `numpy.recarray` or `pandas.DataFrame`.

Matplotlib allows you provide such an object with the `data` keyword argument. If provided, then you may generate plots with the strings corresponding to these variables.

```
data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100

plt.scatter('a', 'b', c='c', s='d', data=data)
plt.xlabel('entry a')
plt.ylabel('entry b')
plt.show()
```



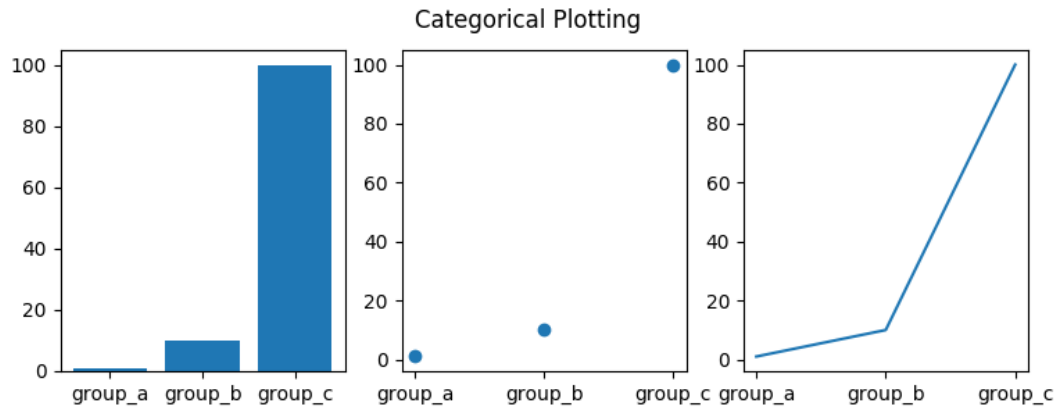
Plotting with categorical variables

It is also possible to create a plot using categorical variables. Matplotlib allows you to pass categorical variables directly to many plotting functions. For example:

```
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]

plt.figure(1, figsize=(9, 3))

plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show()
```



Controlling line properties

Lines have many attributes that you can set: linewidth, dash style, antialiased, etc; see `matplotlib.lines.Line2D`. There are several ways to set line properties

- Use keyword args:

```
plt.plot(x, y, linewidth=2.0)
```

- Use the setter methods of a `Line2D` instance. `plot` returns a list of `Line2D` objects; e.g., `line1, line2 = plot(x1, y1, x2, y2)`. In the code below we will suppose that we have only one line so that the list returned is of length 1. We use tuple unpacking with `line`, to get the first element of that list:

```
line, = plt.plot(x, y, '-')
line.set_antialiased(False) # turn off antialiasing
```

- Use the `setp()` command. The example below uses a MATLAB-style command to set multiple properties on a list of lines. `setp` works transparently with a list of objects or a single object. You can either use python keyword arguments or MATLAB-style string/value pairs:

```
lines = plt.plot(x1, y1, x2, y2)
# use keyword args
plt.setp(lines, color='r', linewidth=2.0)
# or MATLAB style string value pairs
plt.setp(lines, 'color', 'r', 'linewidth', 2.0)
```

Here are the available `Line2D` properties.

Property	Value Type
<code>alpha</code>	float
<code>animated</code>	[True False]
<code>antialiased</code> or <code>aa</code>	[True False]
<code>clip_box</code>	a <code>matplotlib.transform.Bbox</code> instance

Continued on next page

Table 1 – continued from previous page

Property	Value Type
clip_on	[True False]
clip_path	a Path instance and a Transform instance, a Patch
color or c	any matplotlib color
contains	the hit testing function
dash_capstyle	['butt' 'round' 'projecting']
dash_joinstyle	['miter' 'round' 'bevel']
dashes	sequence of on/off ink in points
data	(np.array xdata, np.array ydata)
figure	a matplotlib.figure.Figure instance
label	any string
linestyle or ls	['-' '--' '-.' ':' 'steps' ...]
linewidth or lw	float value in points
lod	[True False]
marker	['+' ',' '.' '1' '2' '3' '4']
markeredgecolor or mec	any matplotlib color
markeredgewidth or mew	float value in points
markerfacecolor or mfc	any matplotlib color
markersize or ms	float
markevery	[None integer (startind, stride)]
picker	used in interactive line selection
pickradius	the line pick selection radius
solid_capstyle	['butt' 'round' 'projecting']
solid_joinstyle	['miter' 'round' 'bevel']
transform	a matplotlib.transforms.Transform instance
visible	[True False]
xdata	np.array
ydata	np.array
zorder	any number

To get a list of settable line properties, call the `setp()` function with a line or lines as argument

```
In [69]: lines = plt.plot([1, 2, 3])
```

```
In [70]: plt.setp(lines)
```

```
alpha: float
```

```
animated: [True | False]
```

```
antialiased or aa: [True | False]
```

```
...snip
```

Working with multiple figures and axes

MATLAB, and pyplot, have the concept of the current figure and the current axes. All plotting commands apply to the current axes. The function `gca()` returns the current axes (a `matplotlib.axes.Axes` instance), and `gcf()` returns the current figure (`matplotlib.figure.Figure` instance). Normally,

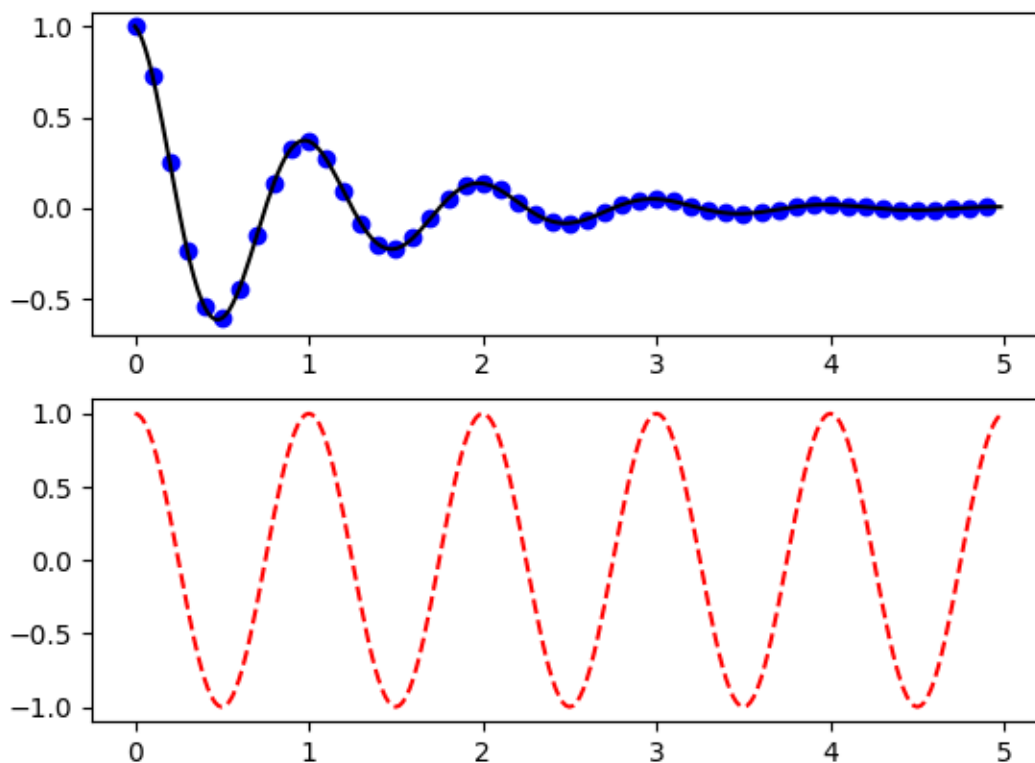
you don't have to worry about this, because it is all taken care of behind the scenes. Below is a script to create two subplots.

```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```



The `figure()` command here is optional because `figure(1)` will be created by default, just as a `subplot(111)` will be created by default if you don't manually specify any axes. The `subplot()` command specifies `numrows`, `numcols`, `plot_number` where `plot_number` ranges from 1 to `numrows*numcols`. The commas in the subplot command are optional if `numrows*numcols < 10`. So `subplot(211)` is identical to `subplot(2, 1, 1)`.

You can create an arbitrary number of subplots and axes. If you want to place an axes manually, i.e., not on a rectangular grid, use the `axes()` command, which allows you to specify the location as `axes([left, bottom, width, height])` where all values are in fractional (0 to 1) coordinates. See [/gallery/subplots_axes_and_figures/axes_demo](#) for an example of placing axes manually and [/gallery/subplots_axes_and_figures/subplot_demo](#) for an example with lots of subplots.

You can create multiple figures by using multiple `figure()` calls with an increasing figure number. Of course, each figure can contain as many axes and subplots as your heart desires:

```
import matplotlib.pyplot as plt
plt.figure(1)           # the first figure
plt.subplot(211)        # the first subplot in the first figure
plt.plot([1, 2, 3])
plt.subplot(212)        # the second subplot in the first figure
plt.plot([4, 5, 6])

plt.figure(2)           # a second figure
plt.plot([4, 5, 6])     # creates a subplot(111) by default

plt.figure(1)           # figure 1 current; subplot(212) still current
plt.subplot(211)        # make subplot(211) in figure1 current
plt.title('Easy as 1, 2, 3') # subplot 211 title
```

You can clear the current figure with `clf()` and the current axes with `cla()`. If you find it annoying that states (specifically the current image, figure and axes) are being maintained for you behind the scenes, don't despair: this is just a thin stateful wrapper around an object oriented API, which you can use instead (see [Artist tutorial](#))

If you are making lots of figures, you need to be aware of one more thing: the memory required for a figure is not completely released until the figure is explicitly closed with `close()`. Deleting all references to the figure, and/or using the window manager to kill the window in which the figure appears on the screen, is not enough, because pyplot maintains internal references until `close()` is called.

Working with text

The `text()` command can be used to add text in an arbitrary location, and the `xlabel()`, `ylabel()` and `title()` are used to add text in the indicated locations (see [Text in Matplotlib Plots](#) for a more detailed example)

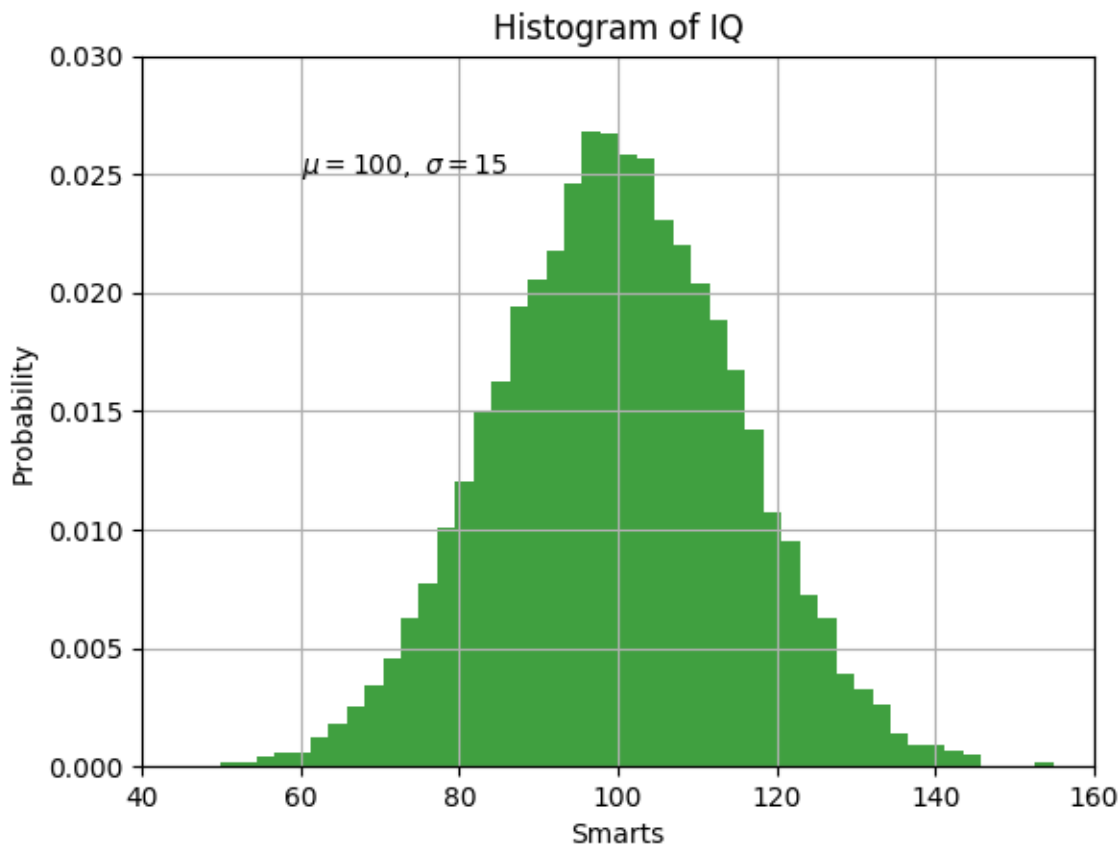
```
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, density=1, facecolor='g', alpha=0.75)
```

(continues on next page)

(continued from previous page)

```
plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```



All of the `text()` commands return an `matplotlib.text.Text` instance. Just as with with lines above, you can customize the properties by passing keyword arguments into the text functions or using `setp()`:

```
t = plt.xlabel('my data', fontsize=14, color='red')
```

These properties are covered in more detail in [Text properties and layout](#).

Using mathematical expressions in text

matplotlib accepts TeX equation expressions in any text expression. For example to write the expression $\sigma_i = 15$ in the title, you can write a TeX expression surrounded by dollar signs:

```
plt.title(r'$\sigma_i=15$')
```

The `r` preceding the title string is important – it signifies that the string is a raw string and not to treat backslashes as python escapes. matplotlib has a built-in TeX expression parser and layout engine, and ships its own math fonts – for details see [Writing mathematical expressions](#). Thus you can use mathematical text across platforms without requiring a TeX installation. For those who have LaTeX and dvipng installed, you can also use LaTeX to format your text and incorporate the output directly into your display figures or saved postscript – see [Text rendering With LaTeX](#).

Annotating text

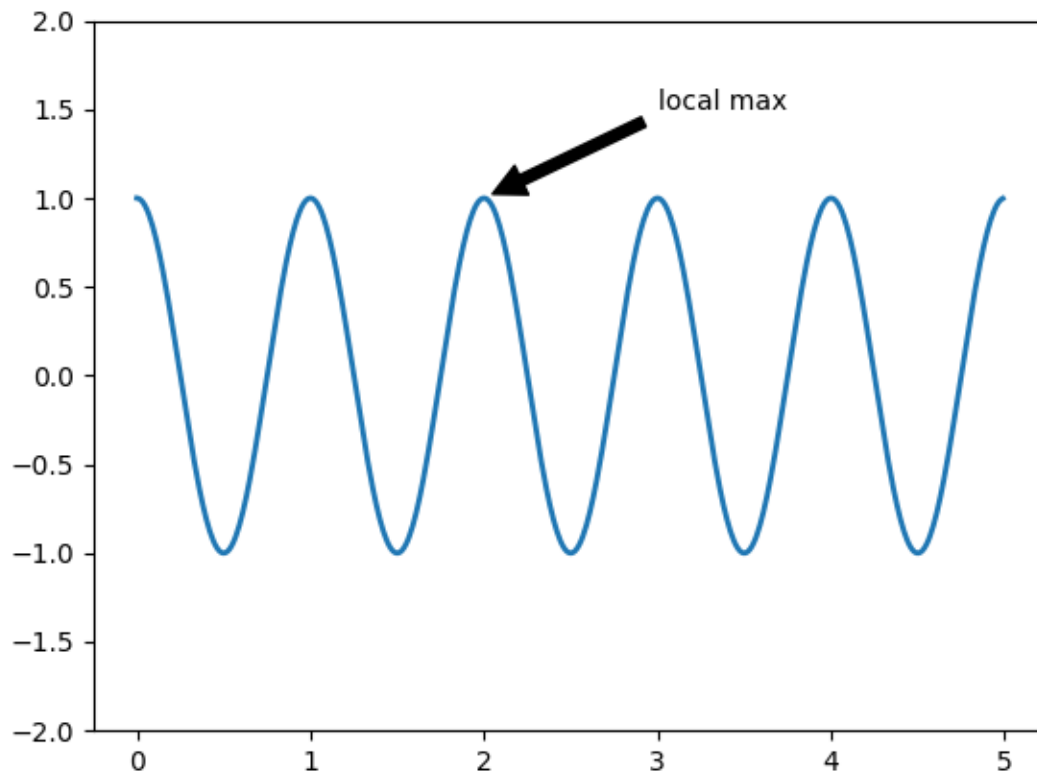
The uses of the basic `text()` command above place text at an arbitrary position on the Axes. A common use for text is to annotate some feature of the plot, and the `annotate()` method provides helper functionality to make annotations easy. In an annotation, there are two points to consider: the location being annotated represented by the argument `xy` and the location of the text `xytext`. Both of these arguments are `(x,y)` tuples.

```
ax = plt.subplot(111)

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = plt.plot(t, s, lw=2)

plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )

plt.ylim(-2, 2)
plt.show()
```



In this basic example, both the `xy` (arrow tip) and `xytext` locations (text location) are in data coordinates. There are a variety of other coordinate systems one can choose – see [Basic annotation](#) and [Advanced Annotation](#) for details. More examples can be found in [/gallery/text_labels_and_annotations/annotation_demo](#).

Logarithmic and other nonlinear axes

`matplotlib.pyplot` supports not only linear axis scales, but also logarithmic and logit scales. This is commonly used if data spans many orders of magnitude. Changing the scale of an axis is easy:

```
plt.xscale('log')
```

An example of four plots with the same data and different scales for the y axis is shown below.

```
from matplotlib.ticker import NullFormatter # useful for `logit` scale

# Fixing random state for reproducibility
np.random.seed(19680801)

# make up some data in the interval ]0, 1[
y = np.random.normal(loc=0.5, scale=0.4, size=1000)
```

(continues on next page)

(continued from previous page)

```
y = y[(y > 0) & (y < 1)]
y.sort()
x = np.arange(len(y))

# plot with various axes scales
plt.figure(1)

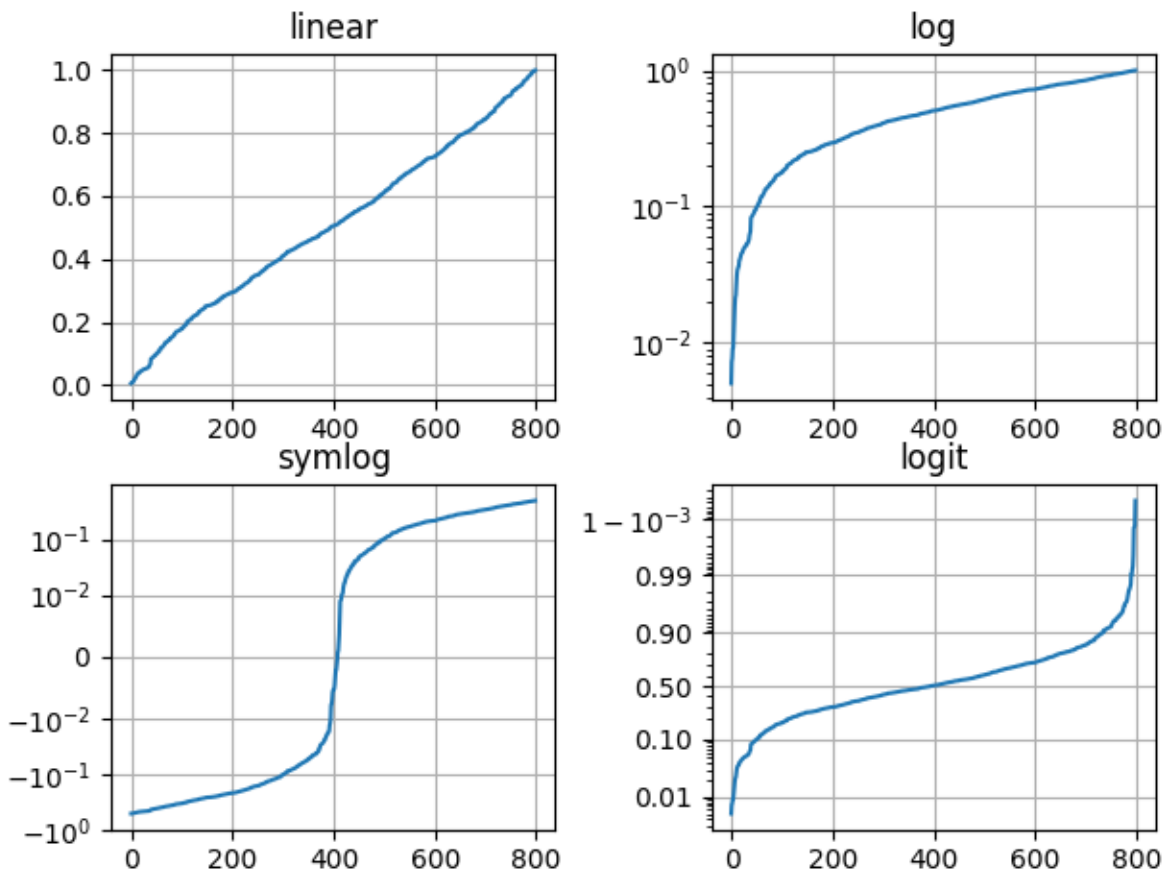
# linear
plt.subplot(221)
plt.plot(x, y)
plt.yscale('linear')
plt.title('linear')
plt.grid(True)

# log
plt.subplot(222)
plt.plot(x, y)
plt.yscale('log')
plt.title('log')
plt.grid(True)

# symmetric log
plt.subplot(223)
plt.plot(x, y - y.mean())
plt.yscale('symlog', linthreshy=0.01)
plt.title('symlog')
plt.grid(True)

# logit
plt.subplot(224)
plt.plot(x, y)
plt.yscale('logit')
plt.title('logit')
plt.grid(True)
# Format the minor tick labels of the y-axis into empty strings with
# `NullFormatter`, to avoid cumbering the axis with too many labels.
plt.gca().yaxis.set_minor_formatter(NullFormatter())
# Adjust the subplot layout, because the logit one may take more space
# than usual, due to y-tick labels like "1 - 10^{-3}"
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.25,
                    wspace=0.35)

plt.show()
```



It is also possible to add your own scale, see Developer's guide for creating scales and transformations for details.

Note: [Click here to download the full example code](#)

3.1.7 The Lifecycle of a Plot

This tutorial aims to show the beginning, middle, and end of a single visualization using Matplotlib. We'll begin with some raw data and end by saving a figure of a customized visualization. Along the way we'll try to highlight some neat features and best-practices using Matplotlib.

Note: This tutorial is based off of [this excellent blog post](#) by Chris Moffitt. It was transformed into this tutorial by Chris Holdgraf.

A note on the Object-Oriented API vs Pyplot

Matplotlib has two interfaces. The first is an object-oriented (OO) interface. In this case, we utilize an instance of `axes.Axes` in order to render visualizations on an instance of `figure.Figure`.

The second is based on MATLAB and uses a state-based interface. This is encapsulated in the `pyplot` module. See the [pyplot tutorials](#) for a more in-depth look at the `pyplot` interface.

Most of the terms are straightforward but the main thing to remember is that:

- The `Figure` is the final image that may contain 1 or more `Axes`.
- The `Axes` represent an individual plot (don't confuse this with the word "axis", which refers to the x/y axis of a plot).

We call methods that do the plotting directly from the `Axes`, which gives us much more flexibility and power in customizing our plot. See the object-oriented examples for many examples of how this approach is used.

Note: In general, try to use the object-oriented interface over the `pyplot` interface.

Our data

We'll use the data from the post from which this tutorial was derived. It contains sales information for a number of companies.

```
# sphinx_gallery_thumbnail_number = 10
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter

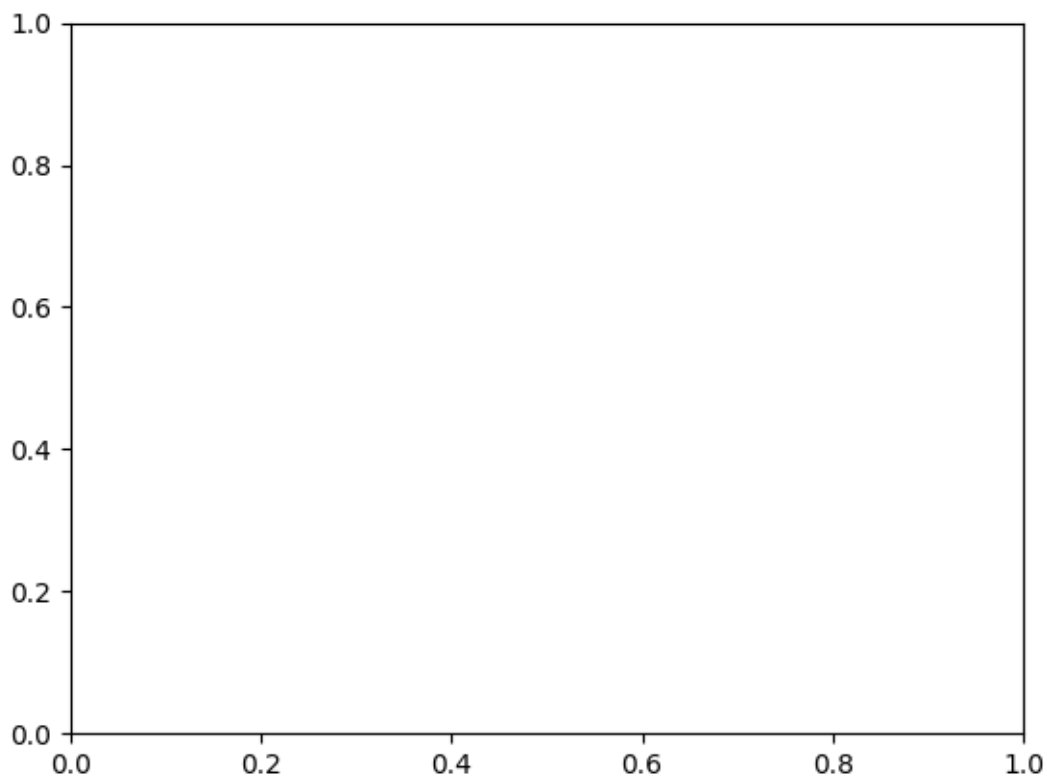
data = {'Barton LLC': 109438.50,
        'Frami, Hills and Schmidt': 103569.59,
        'Fritsch, Russel and Anderson': 112214.71,
        'Jerde-Hilpert': 112591.43,
        'Keeling LLC': 100934.30,
        'Koepp Ltd': 103660.54,
        'Kulas Inc': 137351.96,
        'Trantow-Barrows': 123381.38,
        'White-Trantow': 135841.99,
        'Will LLC': 104437.60}
group_data = list(data.values())
group_names = list(data.keys())
group_mean = np.mean(group_data)
```

Getting started

This data is naturally visualized as a barplot, with one bar per group. To do this with the object-oriented approach, we'll first generate an instance of `figure.Figure` and `axes.Axes`. The `Figure` is like a canvas, and the `Axes` is a part of that canvas on which we will make a particular visualization.

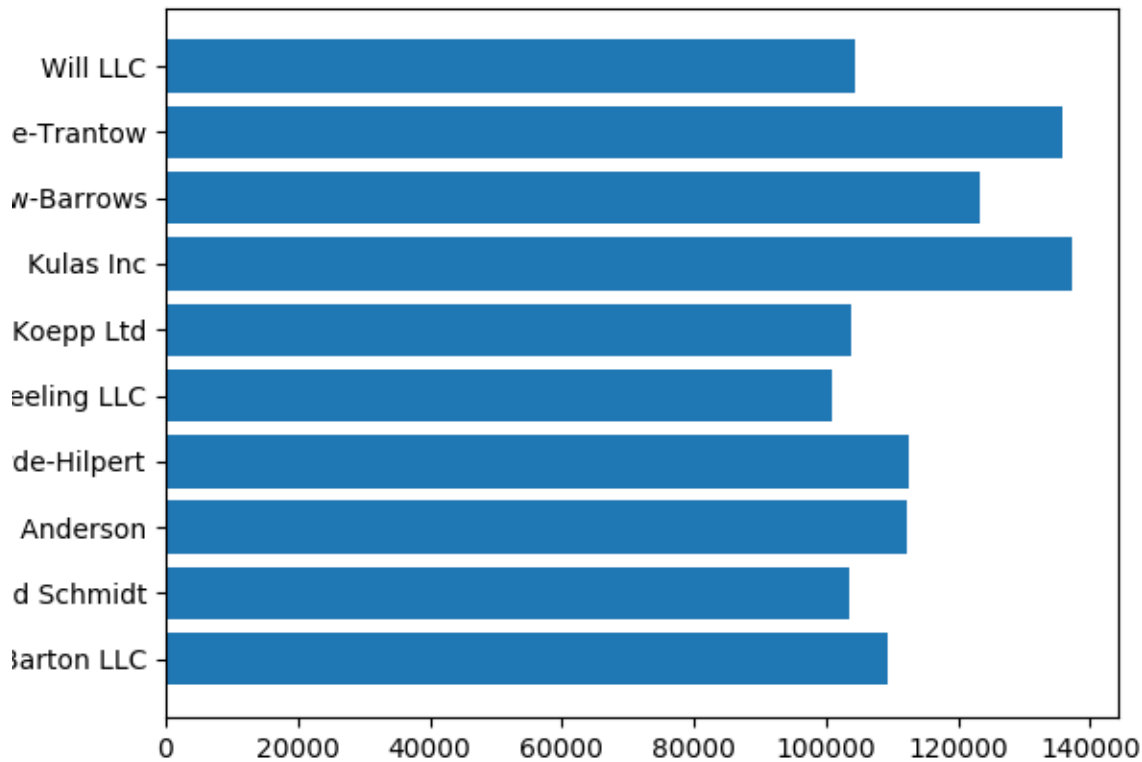
Note: Figures can have multiple axes on them. For information on how to do this, see the [Tight Layout tutorial](#).

```
fig, ax = plt.subplots()
```



Now that we have an `Axes` instance, we can plot on top of it.

```
fig, ax = plt.subplots()
ax.barh(group_names, group_data)
```

Controlling the style

There are many styles available in Matplotlib in order to let you tailor your visualization to your needs. To see a list of styles, we can use `pyplot.style`.

```
print(plt.style.available)
```

Out:

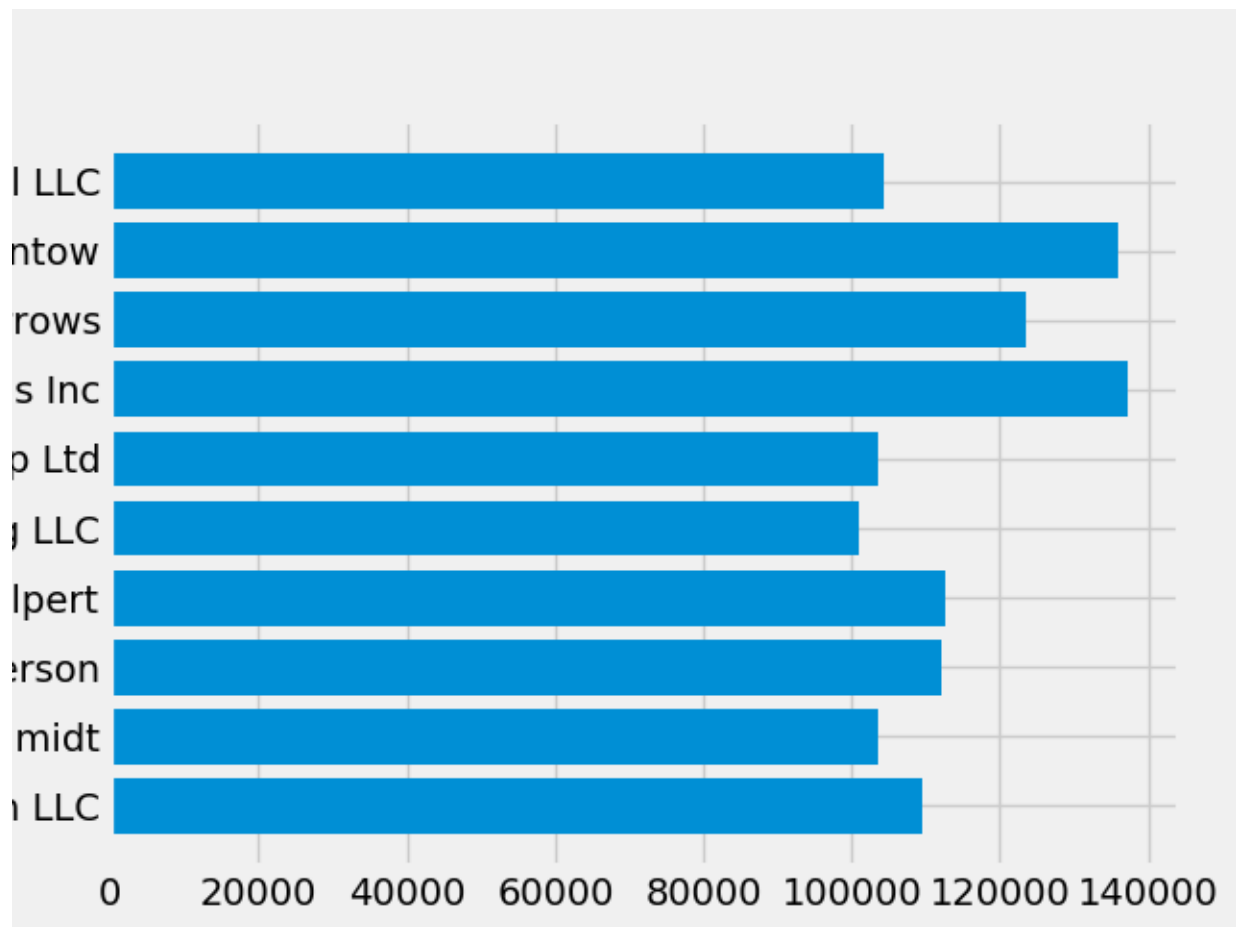
```
['seaborn-paper', 'seaborn-poster', 'seaborn-darkgrid', 'fast', 'ggplot', 'seaborn-pastel', 'bmh',  
↪ 'seaborn-deep', 'fivethirtyeight', 'grayscale', 'seaborn', 'seaborn-talk', 'seaborn-bright',  
↪ 'seaborn-white', 'seaborn-ticks', 'seaborn-dark-palette', 'seaborn-dark', '__classic_test',  
↪ 'seaborn-colorblind', 'dark_background', 'seaborn-whitegrid', 'seaborn-notebook', 'tableau-  
↪ colorblind10', 'Solarize_Light2', 'seaborn-muted', 'classic']
```

You can activate a style with the following:

```
plt.style.use('fivethirtyeight')
```

Now let's remake the above plot to see how it looks:

```
fig, ax = plt.subplots()
ax.barh(group_names, group_data)
```

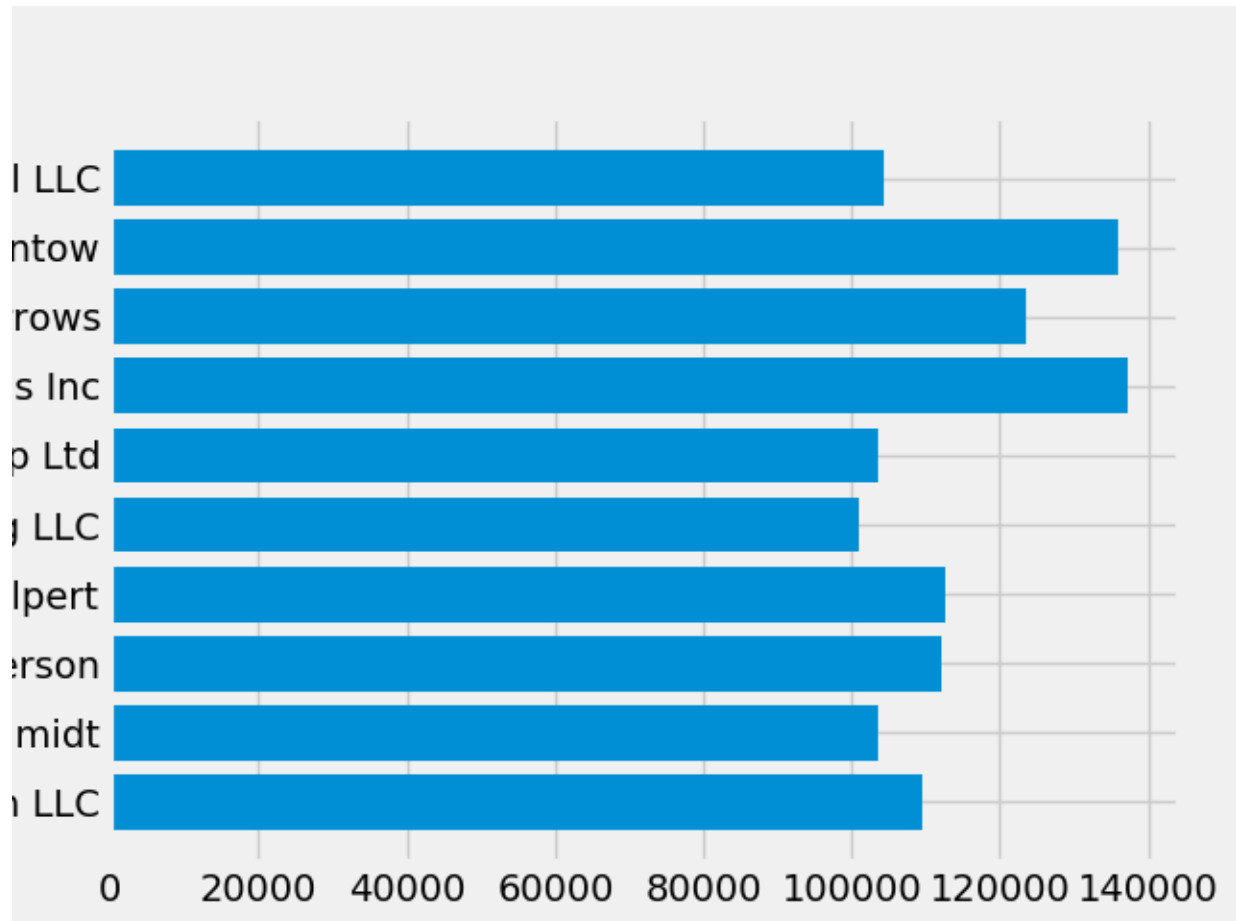


The style controls many things, such as color, linewidths, backgrounds, etc.

Customizing the plot

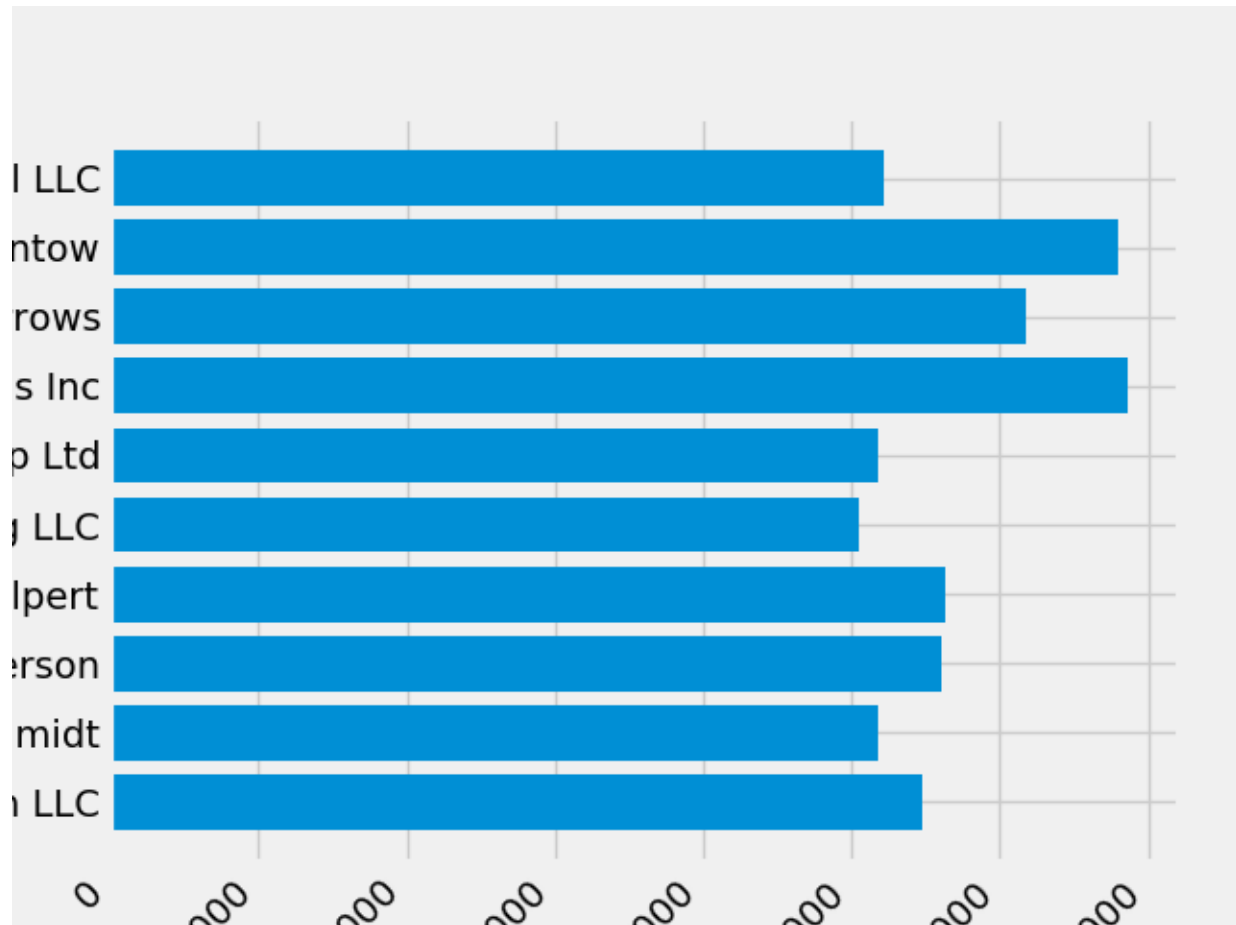
Now we've got a plot with the general look that we want, so let's fine-tune it so that it's ready for print. First let's rotate the labels on the x-axis so that they show up more clearly. We can gain access to these labels with the `axes.Axes.get_xticklabels()` method:

```
fig, ax = plt.subplots()
ax.barh(group_names, group_data)
labels = ax.get_xticklabels()
```



If we'd like to set the property of many items at once, it's useful to use the `pyplot.setp()` function. This will take a list (or many lists) of Matplotlib objects, and attempt to set some style element of each one.

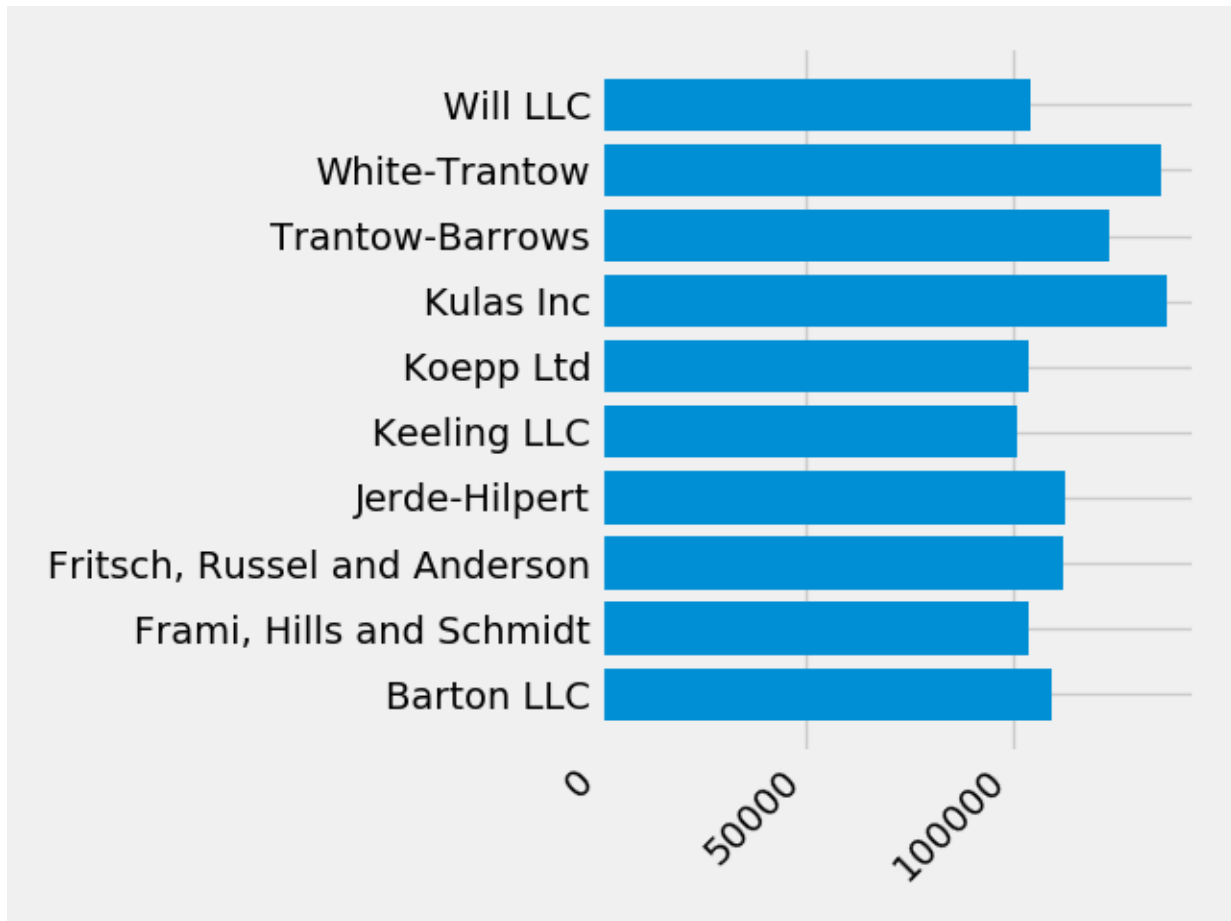
```
fig, ax = plt.subplots()
ax.barh(group_names, group_data)
labels = ax.get_xticklabels()
plt.setp(labels, rotation=45, horizontalalignment='right')
```



It looks like this cut off some of the labels on the bottom. We can tell Matplotlib to automatically make room for elements in the figures that we create. To do this we'll set the `autolayout` value of our `rcParams`. For more information on controlling the style, layout, and other features of plots with `rcParams`, see [Customizing Matplotlib with style sheets and `rcParams`](#).

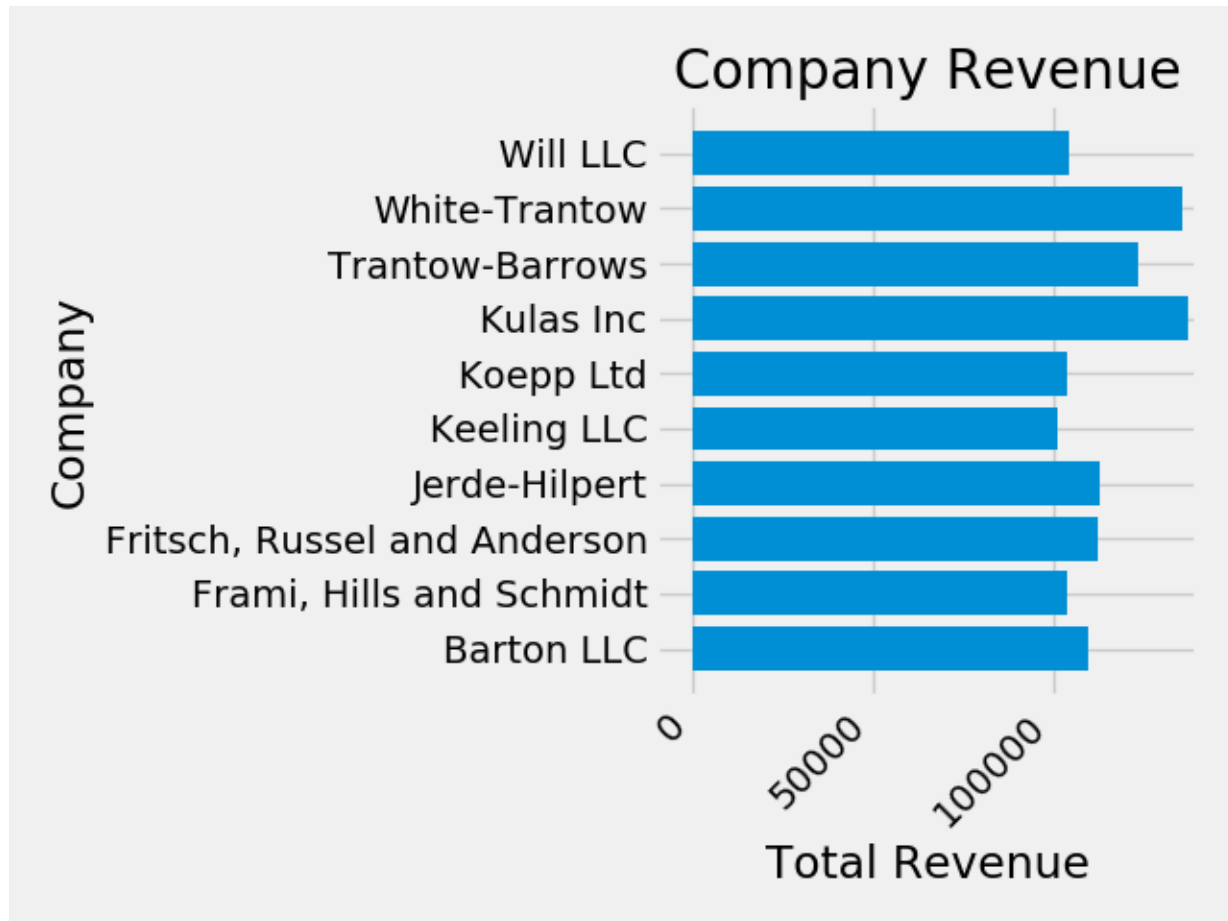
```
plt.rcParams.update({'figure.autolayout': True})

fig, ax = plt.subplots()
ax.barh(group_names, group_data)
labels = ax.get_xticklabels()
plt.setp(labels, rotation=45, horizontalalignment='right')
```



Next, we'll add labels to the plot. To do this with the OO interface, we can use the `axes.Axes.set()` method to set properties of this Axes object.

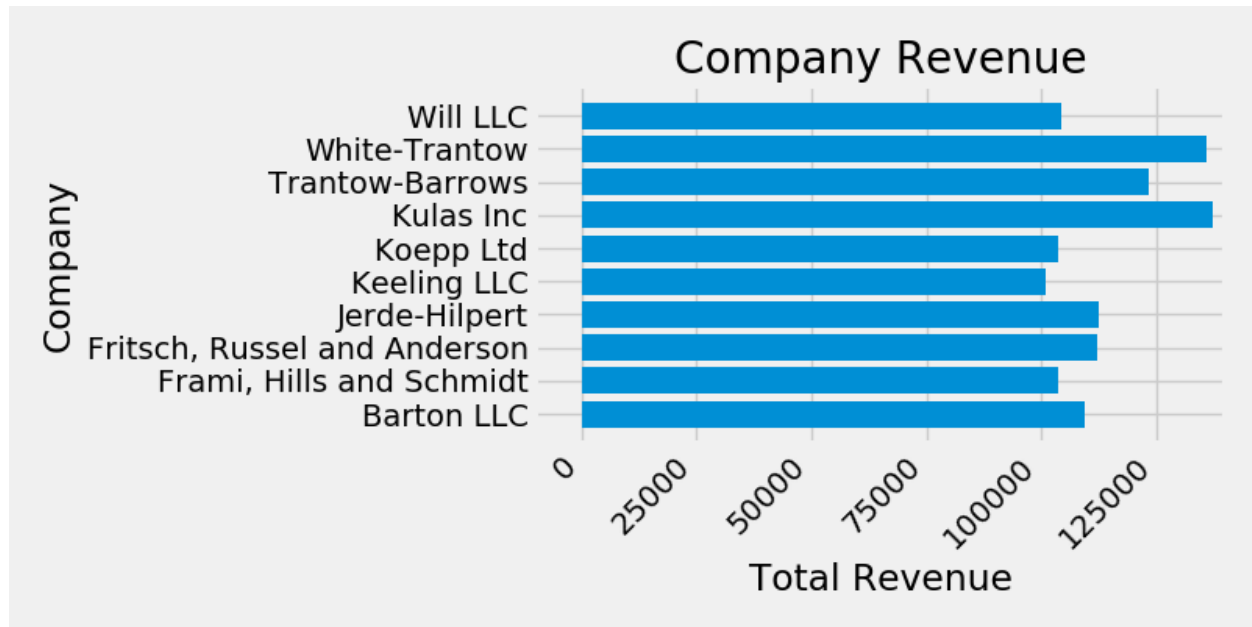
```
fig, ax = plt.subplots()
ax.barh(group_names, group_data)
labels = ax.get_xticklabels()
plt.setp(labels, rotation=45, horizontalalignment='right')
ax.set(xlim=[-10000, 140000], xlabel='Total Revenue', ylabel='Company',
       title='Company Revenue')
```



We can also adjust the size of this plot using the `pyplot.subplots()` function. We can do this with the `figsize` kwarg.

Note: While indexing in NumPy follows the form (row, column), the `figsize` kwarg follows the form (width, height). This follows conventions in visualization, which unfortunately are different from those of linear algebra.

```
fig, ax = plt.subplots(figsize=(8, 4))
ax.barh(group_names, group_data)
labels = ax.get_xticklabels()
plt.setp(labels, rotation=45, horizontalalignment='right')
ax.set(xlim=[-10000, 140000], xlabel='Total Revenue', ylabel='Company',
       title='Company Revenue')
```



For labels, we can specify custom formatting guidelines in the form of functions by using the `ticker.FuncFormatter` class. Below we'll define a function that takes an integer as input, and returns a string as an output.

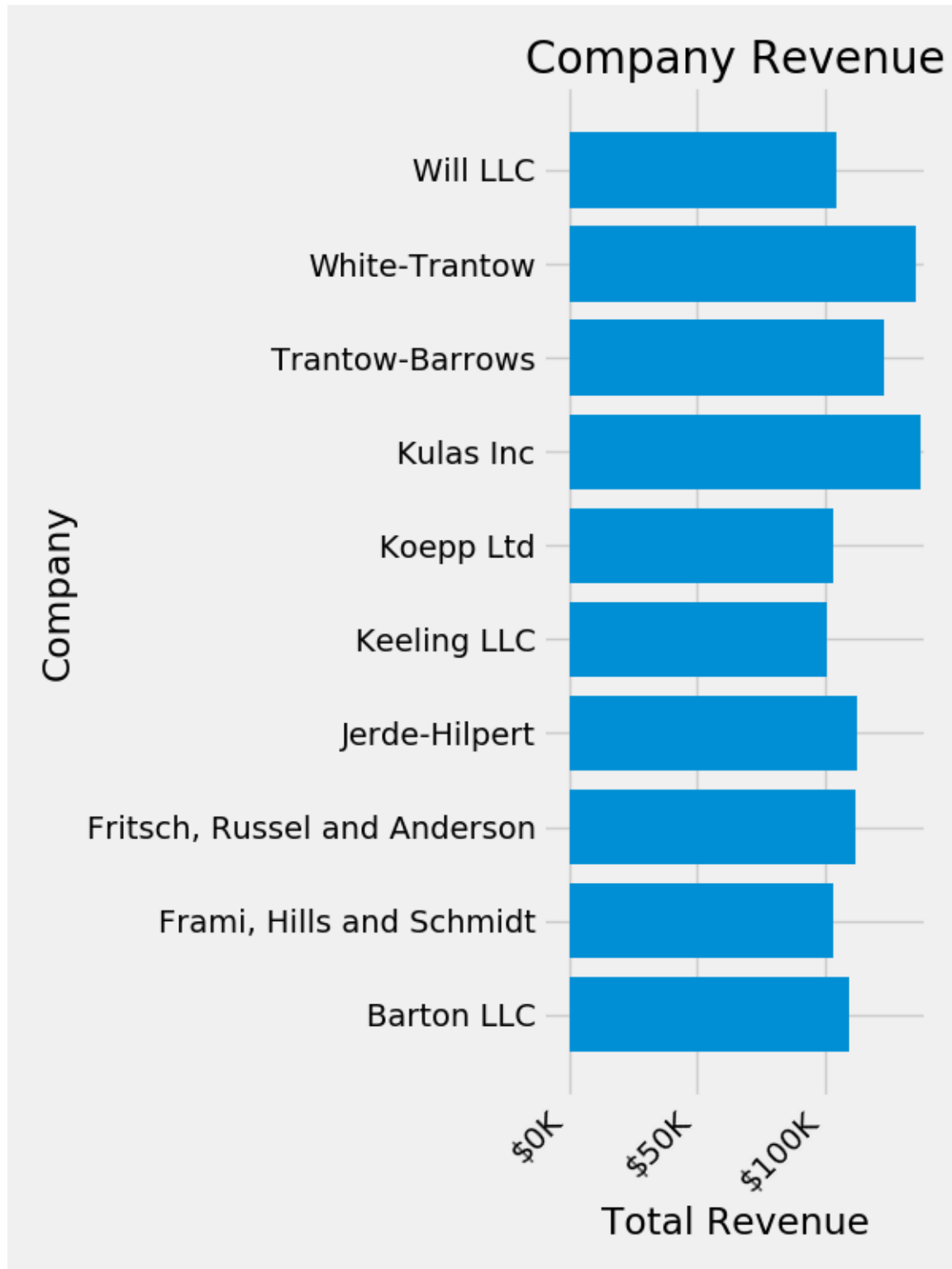
```
def currency(x, pos):
    """The two args are the value and tick position"""
    if x >= 1e6:
        s = '${:1.1f}M'.format(x*1e-6)
    else:
        s = '${:1.0f}K'.format(x*1e-3)
    return s

formatter = FuncFormatter(currency)
```

We can then apply this formatter to the labels on our plot. To do this, we'll use the `xaxis` attribute of our axis. This lets you perform actions on a specific axis on our plot.

```
fig, ax = plt.subplots(figsize=(6, 8))
ax.barh(group_names, group_data)
labels = ax.get_xticklabels()
plt.setp(labels, rotation=45, horizontalalignment='right')

ax.set(xlim=[-10000, 140000], xlabel='Total Revenue', ylabel='Company',
       title='Company Revenue')
ax.xaxis.set_major_formatter(formatter)
```



Combining multiple visualizations

It is possible to draw multiple plot elements on the same instance of `axes.Axes`. To do this we simply need to call another one of the plot methods on that axes object.

```
fig, ax = plt.subplots(figsize=(8, 8))
ax.barh(group_names, group_data)
labels = ax.get_xticklabels()
plt.setp(labels, rotation=45, horizontalalignment='right')

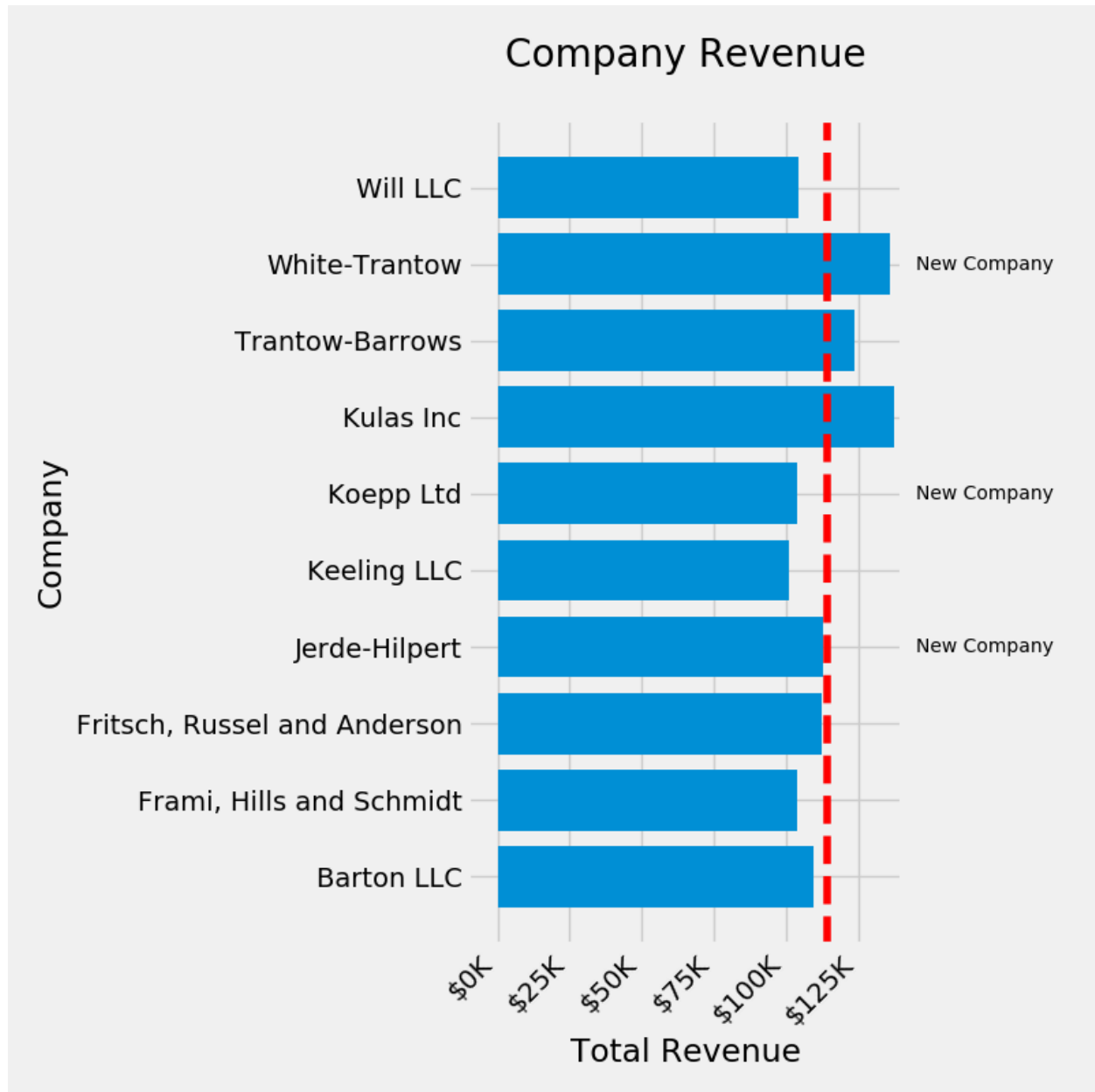
# Add a vertical line, here we set the style in the function call
ax.axvline(group_mean, ls='--', color='r')

# Annotate new companies
for group in [3, 5, 8]:
    ax.text(145000, group, "New Company", fontsize=10,
           verticalalignment="center")

# Now we'll move our title up since it's getting a little cramped
ax.title.set(y=1.05)

ax.set(xlim=[-10000, 140000], xlabel='Total Revenue', ylabel='Company',
       title='Company Revenue')
ax.xaxis.set_major_formatter(formatter)
ax.set_xticks([0, 25e3, 50e3, 75e3, 100e3, 125e3])
fig.subplots_adjust(right=.1)

plt.show()
```



Saving our plot

Now that we're happy with the outcome of our plot, we want to save it to disk. There are many file formats we can save to in Matplotlib. To see a list of available options, use:

```
print(fig.canvas.get_supported_filetypes())
```

Out:

```
{'ps': 'Postscript', 'eps': 'Encapsulated Postscript', 'pdf': 'Portable Document Format', 'pgf':
  ↳ 'PGF code for LaTeX', 'png': 'Portable Network Graphics', 'raw': 'Raw RGBA bitmap', 'rgba
  ↳ ': 'Raw RGBA bitmap', 'svg': 'Scalable Vector Graphics', 'svgz': 'Scalable Vector Graphics'
  ↳ 'jpg': 'Joint Photographic Experts Group', 'jpeg': 'Joint Photographic Experts Group', 'tif':
  ↳ 'Tagged Image File Format', 'tiff': 'Tagged Image File Format'}
```

(continued from previous page)

We can then use the `figure.Figure.savefig()` in order to save the figure to disk. Note that there are several useful flags we'll show below:

- `transparent=True` makes the background of the saved figure transparent if the format supports it.
- `dpi=80` controls the resolution (dots per square inch) of the output.
- `bbox_inches="tight"` fits the bounds of the figure to our plot.

```
# Uncomment this line to save the figure.  
# fig.savefig('sales.png', transparent=False, dpi=80, bbox_inches="tight")
```

3.2 Intermediate

These tutorials cover some of the more complicated classes and functions in Matplotlib. They can be useful for particular custom and complex visualizations.

Note: Click [here](#) to download the full example code

3.2.1 Styling with cycler

Demo of custom property-cycle settings to control colors and other style properties for multi-line plots.

Note: More complete documentation of the cycler API can be found [here](#).

This example demonstrates two different APIs:

1. Setting the default `rc` parameter specifying the property cycle. This affects all subsequent axes (but not axes already created).
2. Setting the property cycle for a single pair of axes.

```
from cycler import cycler  
import numpy as np  
import matplotlib.pyplot as plt
```

First we'll generate some sample data, in this case, four offset sine curves.

```
x = np.linspace(0, 2 * np.pi, 50)  
offsets = np.linspace(0, 2 * np.pi, 4, endpoint=False)  
yy = np.transpose([np.sin(x + phi) for phi in offsets])
```

Now `yy` has shape

```
print(yy.shape)
```

Out:

```
(50, 4)
```

So `yy[:, i]` will give you the *i*-th offset sine curve. Let's set the default `prop_cycle` using `matplotlib.pyplot.rc()`. We'll combine a color cyler and a linestyle cyler by adding (+) two cyler's together. See the bottom of this tutorial for more information about combining different cyclers.

```
default_cycler = (cyler(color=['r', 'g', 'b', 'y']) +
                  cyler(linestyle=['-', '--', ':', '-.']))

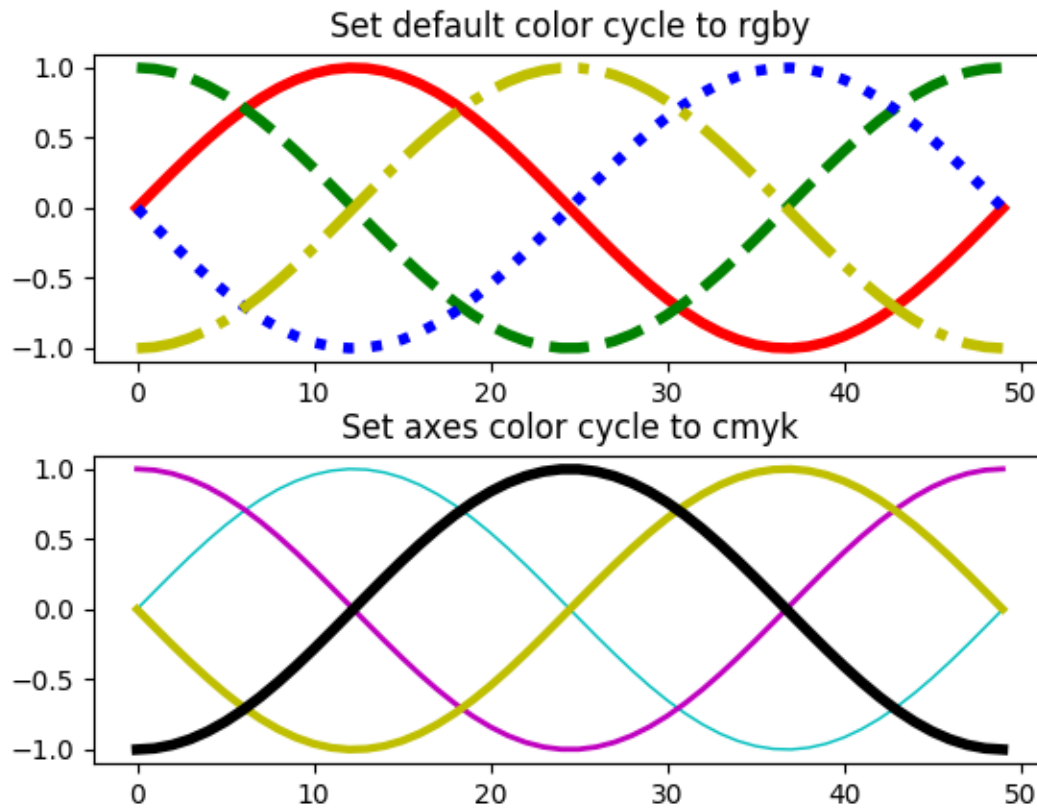
plt.rc('lines', linewidth=4)
plt.rc('axes', prop_cycle=default_cycler)
```

Now we'll generate a figure with two axes, one on top of the other. On the first axis, we'll plot with the default cyler. On the second axis, we'll set the `prop_cycler` using `matplotlib.axes.Axes.set_prop_cycle()` which will only set the `prop_cycle` for this `matplotlib.axes.Axes` instance. We'll use a second cyler that combines a color cyler and a linewidth cyler.

```
custom_cycler = (cyler(color=['c', 'm', 'y', 'k']) +
                  cyler(lw=[1, 2, 3, 4]))

fig, (ax0, ax1) = plt.subplots(nrows=2)
ax0.plot(yy)
ax0.set_title('Set default color cycle to rgby')
ax1.set_prop_cycle(custom_cycler)
ax1.plot(yy)
ax1.set_title('Set axes color cycle to cmyk')

# Add a bit more space between the two plots.
fig.subplots_adjust(hspace=0.3)
plt.show()
```



Setting `prop_cycler` in the `matplotlibrc` file or style files

Remember, if you want to set a custom `prop_cycler` in your `.matplotlibrc` file or a style file (`style.mplstyle`), you can set the `axes.prop_cycle` property:

```
axes.prop_cycle : cycler(color='bgrcmyk')
```

Cycling through multiple properties

You can add cyclers:

```
from cycler import cycler
cc = (cycler(color=list('rgb')) +
      cycler(linestyle=['-', '--', '-.']))
for d in cc:
    print(d)
```

Results in:

```
{'color': 'r', 'linestyle': '-'}  
{'color': 'g', 'linestyle': '--'}  
{'color': 'b', 'linestyle': '-.'}
```

You can multiply cyclers:

```
from cycler import cycler  
cc = (cycler(color=list('rgb')) *  
      cycler(linestyle=['-', '--', '-.']))  
for d in cc:  
    print(d)
```

Results in:

```
{'color': 'r', 'linestyle': '-'}  
{'color': 'r', 'linestyle': '--'}  
{'color': 'r', 'linestyle': '-.'}  
{'color': 'g', 'linestyle': '-'}  
{'color': 'g', 'linestyle': '--'}  
{'color': 'g', 'linestyle': '-.'}  
{'color': 'b', 'linestyle': '-'}  
{'color': 'b', 'linestyle': '--'}  
{'color': 'b', 'linestyle': '-.'}
```

Note: [Click here to download the full example code](#)

3.2.2 Legend guide

Generating legends flexibly in Matplotlib.

This legend guide is an extension of the documentation available at `legend()` - please ensure you are familiar with contents of that documentation before proceeding with this guide.

This guide makes use of some common terms, which are documented here for clarity:

legend entry A legend is made up of one or more legend entries. An entry is made up of exactly one key and one label.

legend key The colored/patterned marker to the left of each legend label.

legend label The text which describes the handle represented by the key.

legend handle The original object which is used to generate an appropriate entry in the legend.

Controlling the legend entries

Calling `legend()` with no arguments automatically fetches the legend handles and their associated labels. This functionality is equivalent to:

```
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, labels)
```

The `get_legend_handles_labels()` function returns a list of handles/artists which exist on the Axes which can be used to generate entries for the resulting legend - it is worth noting however that not all artists can be added to a legend, at which point a "proxy" will have to be created (see [Creating artists specifically for adding to the legend \(aka. Proxy artists\)](#) for further details).

For full control of what is being added to the legend, it is common to pass the appropriate handles directly to `legend()`:

```
line_up, = plt.plot([1,2,3], label='Line 2')
line_down, = plt.plot([3,2,1], label='Line 1')
plt.legend(handles=[line_up, line_down])
```

In some cases, it is not possible to set the label of the handle, so it is possible to pass through the list of labels to `legend()`:

```
line_up, = plt.plot([1,2,3], label='Line 2')
line_down, = plt.plot([3,2,1], label='Line 1')
plt.legend([line_up, line_down], ['Line Up', 'Line Down'])
```

Creating artists specifically for adding to the legend (aka. Proxy artists)

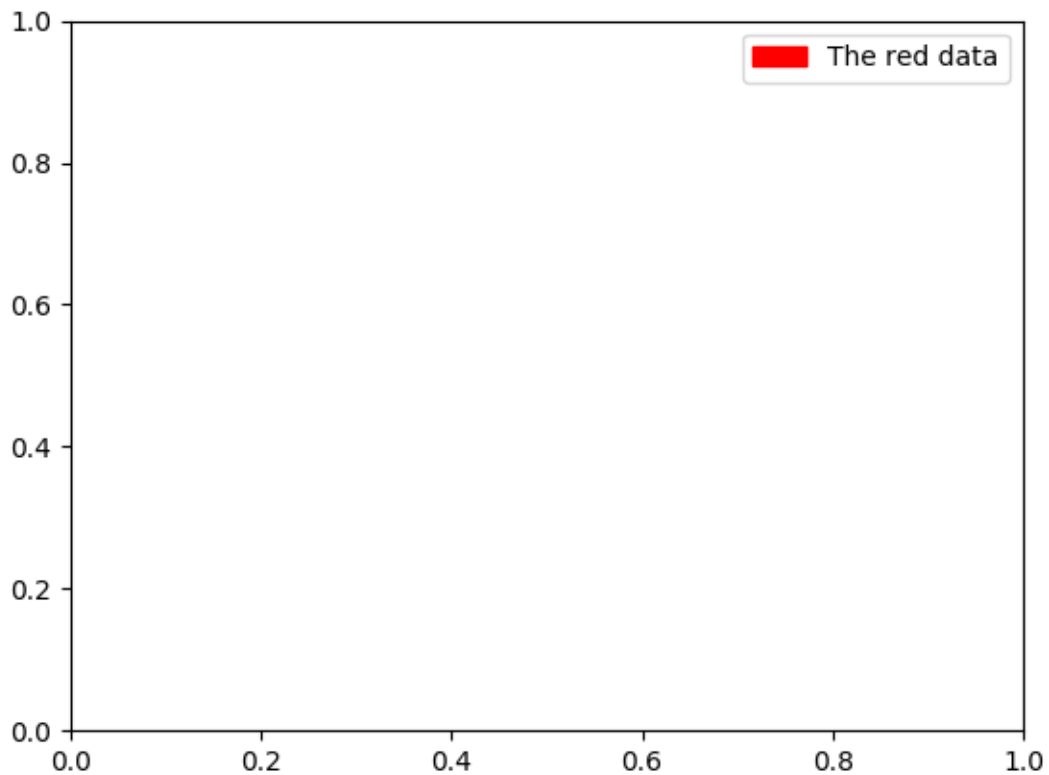
Not all handles can be turned into legend entries automatically, so it is often necessary to create an artist which can. Legend handles don't have to exist on the Figure or Axes in order to be used.

Suppose we wanted to create a legend which has an entry for some data which is represented by a red color:

```
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt

red_patch = mpatches.Patch(color='red', label='The red data')
plt.legend(handles=[red_patch])

plt.show()
```

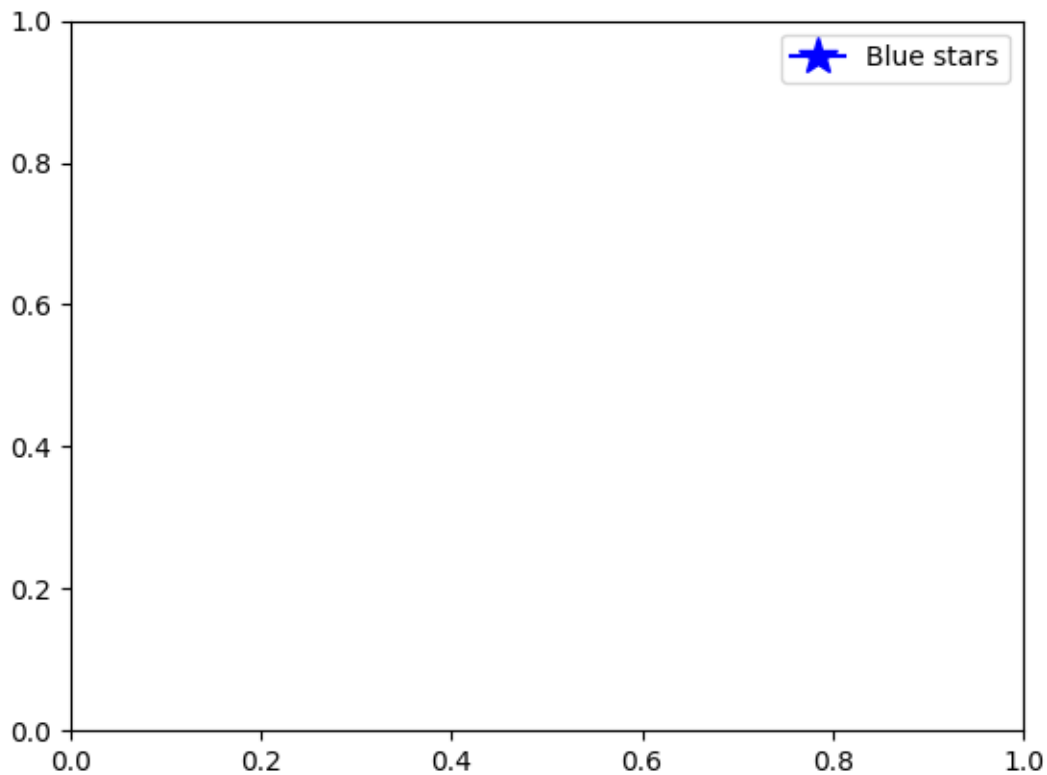


There are many supported legend handles, instead of creating a patch of color we could have created a line with a marker:

```
import matplotlib.lines as mlines

blue_line = mlines.Line2D([], [], color='blue', marker='*',
                           markersize=15, label='Blue stars')
plt.legend(handles=[blue_line])

plt.show()
```

Legend location

The location of the legend can be specified by the keyword argument `loc`. Please see the documentation at `legend()` for more details.

The `bbox_to_anchor` keyword gives a great degree of control for manual legend placement. For example, if you want your axes legend located at the figure's top right-hand corner instead of the axes' corner, simply specify the corner's location, and the coordinate system of that location:

```
plt.legend(bbox_to_anchor=(1, 1),
           bbox_transform=plt.gcf().transFigure)
```

More examples of custom legend placement:

```
plt.subplot(211)
plt.plot([1, 2, 3], label="test1")
plt.plot([3, 2, 1], label="test2")

# Place a legend above this subplot, expanding itself to
# fully use the given bounding box.
```

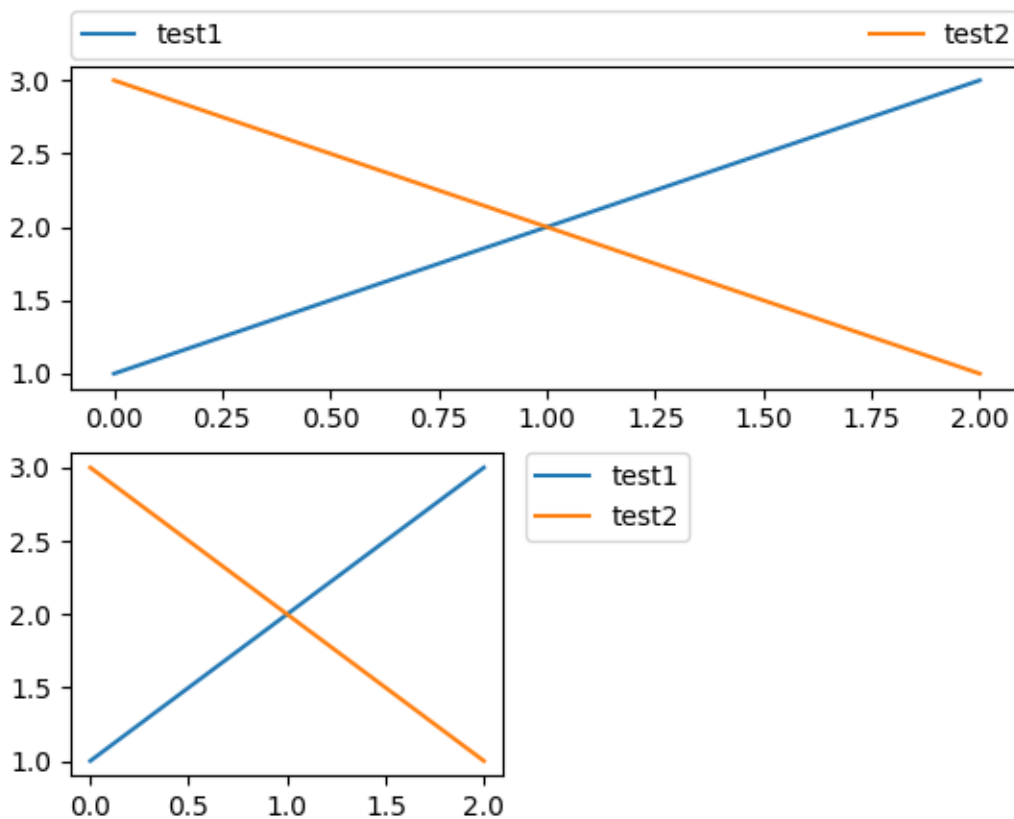
(continues on next page)

(continued from previous page)

```
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
           ncol=2, mode="expand", borderaxespad=0.)

plt.subplot(223)
plt.plot([1, 2, 3], label="test1")
plt.plot([3, 2, 1], label="test2")
# Place a legend to the right of this smaller subplot.
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()
```



Multiple legends on the same Axes

Sometimes it is more clear to split legend entries across multiple legends. Whilst the instinctive approach to doing this might be to call the `legend()` function multiple times, you will find that only one legend ever exists on the Axes. This has been done so that it is possible to call `legend()` repeatedly to update the legend to the latest handles on the Axes, so to persist old legend instances, we must add them manually to the Axes:

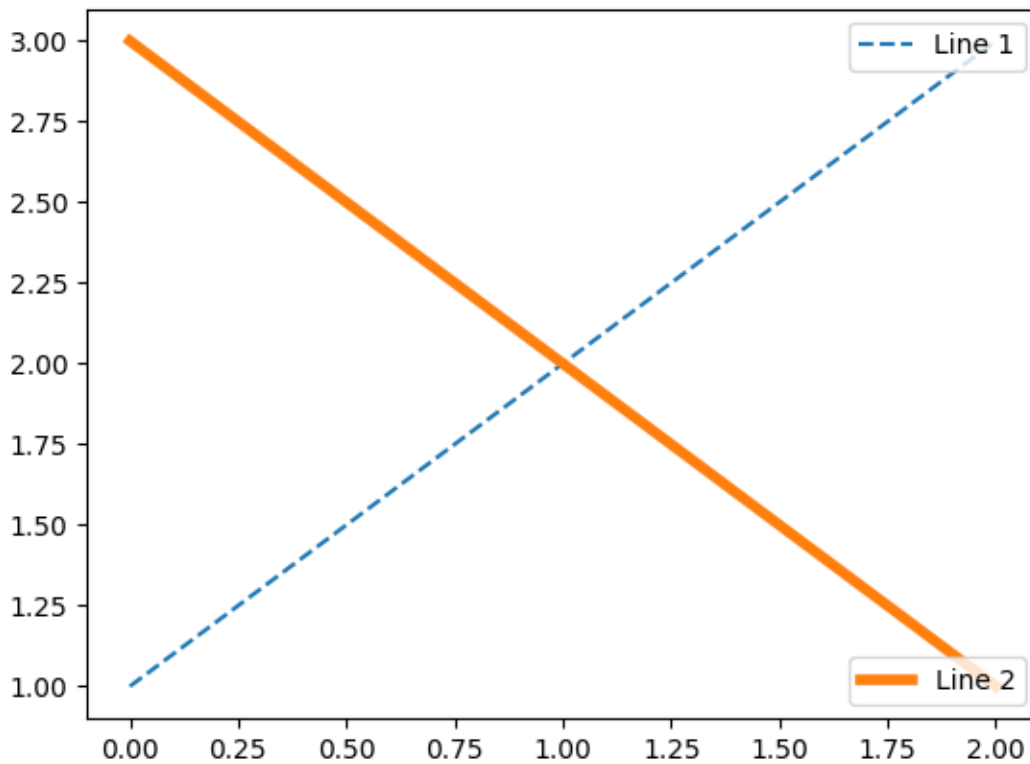
```
line1, = plt.plot([1, 2, 3], label="Line 1", linestyle='--')
line2, = plt.plot([3, 2, 1], label="Line 2", linewidth=4)

# Create a legend for the first line.
first_legend = plt.legend(handles=[line1], loc=1)

# Add the legend manually to the current Axes.
ax = plt.gca().add_artist(first_legend)

# Create another legend for the second line.
plt.legend(handles=[line2], loc=4)

plt.show()
```



Legend Handlers

In order to create legend entries, handles are given as an argument to an appropriate HandlerBase subclass. The choice of handler subclass is determined by the following rules:

1. Update `get_legend_handler_map()` with the value in the `handler_map` keyword.

2. Check if the handle is in the newly created handler_map.
3. Check if the type of handle is in the newly created handler_map.
4. Check if any of the types in the handle's mro is in the newly created handler_map.

For completeness, this logic is mostly implemented in `get_legend_handler()`.

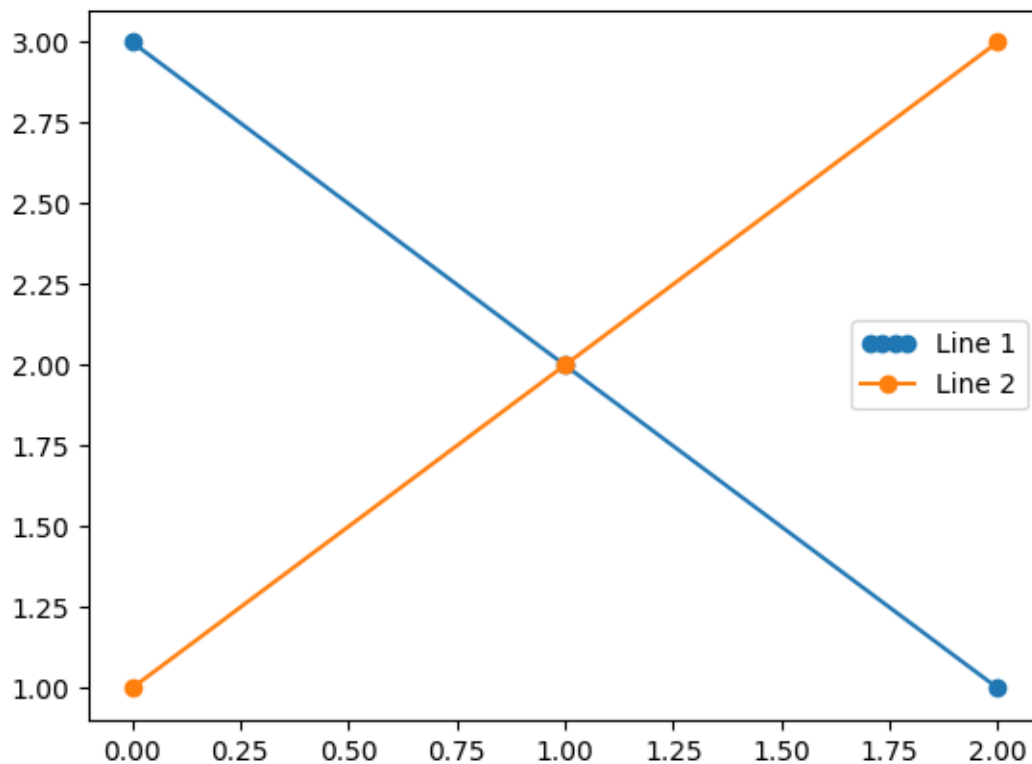
All of this flexibility means that we have the necessary hooks to implement custom handlers for our own type of legend key.

The simplest example of using custom handlers is to instantiate one of the existing `HandlerBase` subclasses. For the sake of simplicity, let's choose `matplotlib.legend_handler.HandlerLine2D` which accepts a `numpoints` argument (note `numpoints` is a keyword on the `legend()` function for convenience). We can then pass the mapping of instance to `Handler` as a keyword to `legend`.

```
from matplotlib.legend_handler import HandlerLine2D

line1, = plt.plot([3, 2, 1], marker='o', label='Line 1')
line2, = plt.plot([1, 2, 3], marker='o', label='Line 2')

plt.legend(handler_map={line1: HandlerLine2D(numpoints=4)})
```



As you can see, "Line 1" now has 4 marker points, where "Line 2" has 2 (the default). Try the

above code, only change the map's key from `line1` to `type(line1)`. Notice how now both `Line2D` instances get 4 markers.

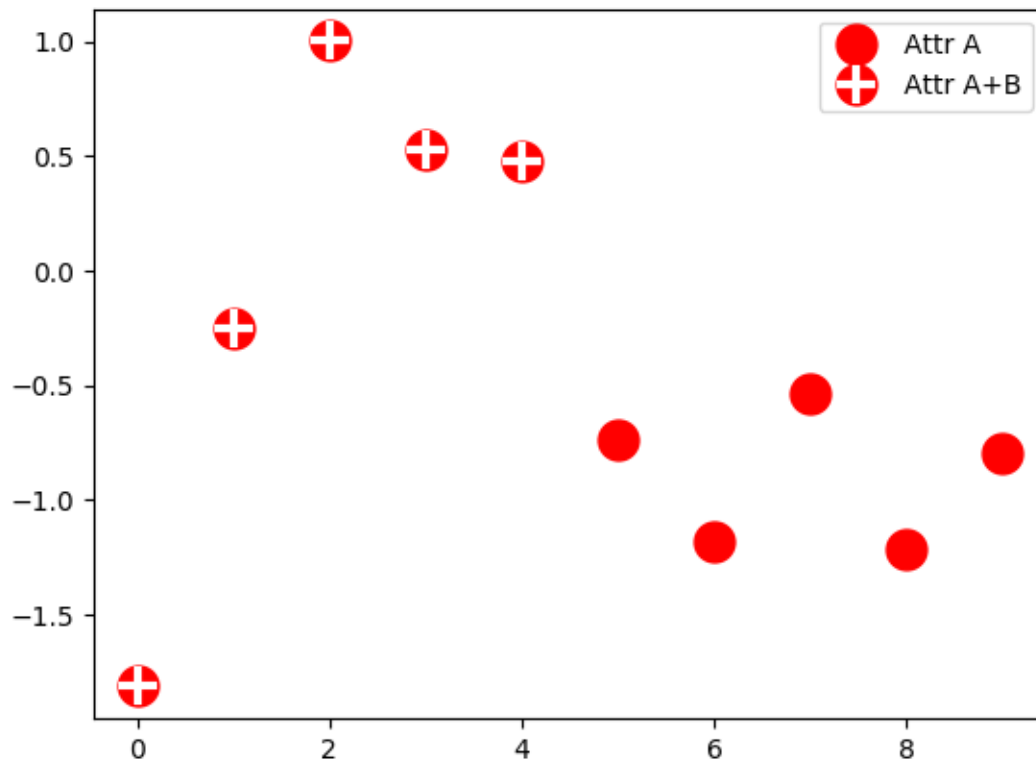
Along with handlers for complex plot types such as errorbars, stem plots and histograms, the default `handler_map` has a special tuple handler (`HandlerTuple`) which simply plots the handles on top of one another for each item in the given tuple. The following example demonstrates combining two legend keys on top of one another:

```
from numpy.random import randn

z = randn(10)

red_dot, = plt.plot(z, "ro", markersize=15)
# Put a white cross over some of the data.
white_cross, = plt.plot(z[:5], "w+", markeredgewidth=3, markersize=15)

plt.legend([red_dot, (red_dot, white_cross)], ["Attr A", "Attr A+B"])
```



The `HandlerTuple` class can also be used to assign several legend keys to the same entry:

```
from matplotlib.legend_handler import HandlerLine2D, HandlerTuple
```

(continues on next page)

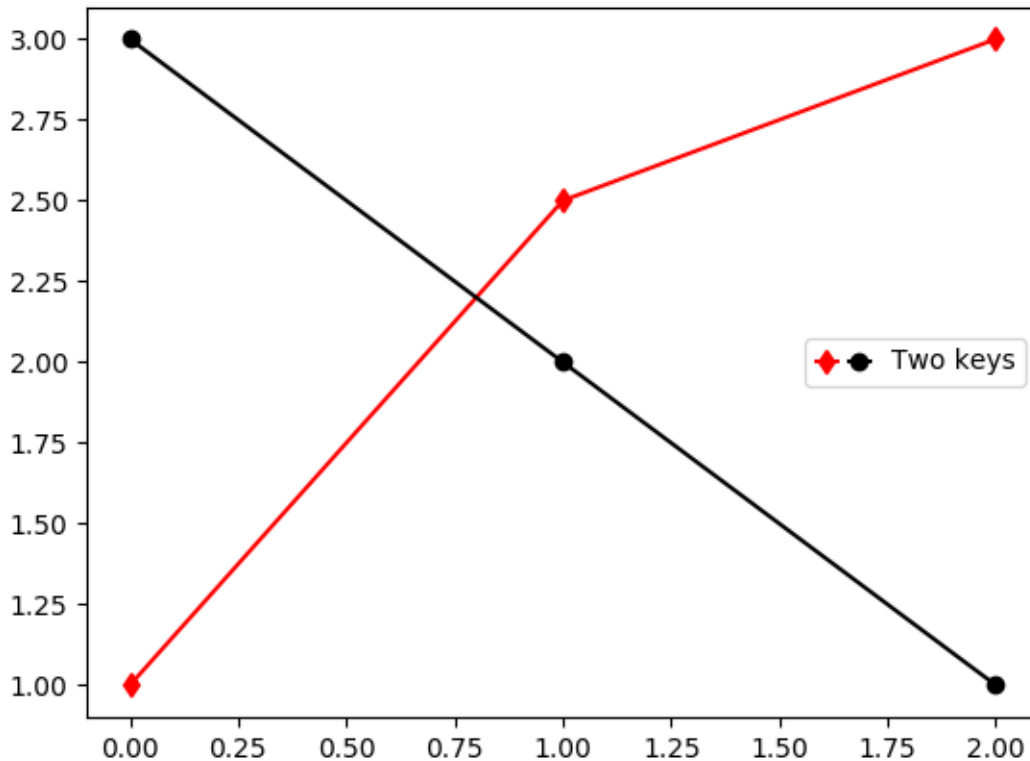
(continued from previous page)

```

p1, = plt.plot([1, 2.5, 3], 'r-d')
p2, = plt.plot([3, 2, 1], 'k-o')

l = plt.legend([(p1, p2)], ['Two keys'], numpoints=1,
               handler_map={tuple: HandlerTuple(ndivide=None)})

```



Implementing a custom legend handler

A custom handler can be implemented to turn any handle into a legend key (handles don't necessarily need to be matplotlib artists). The handler must implement a "legend_artist" method which returns a single artist for the legend to use. Signature details about the "legend_artist" are documented at `legend_artist()`.

```

import matplotlib.patches as mpatches

class AnyObject(object):
    pass

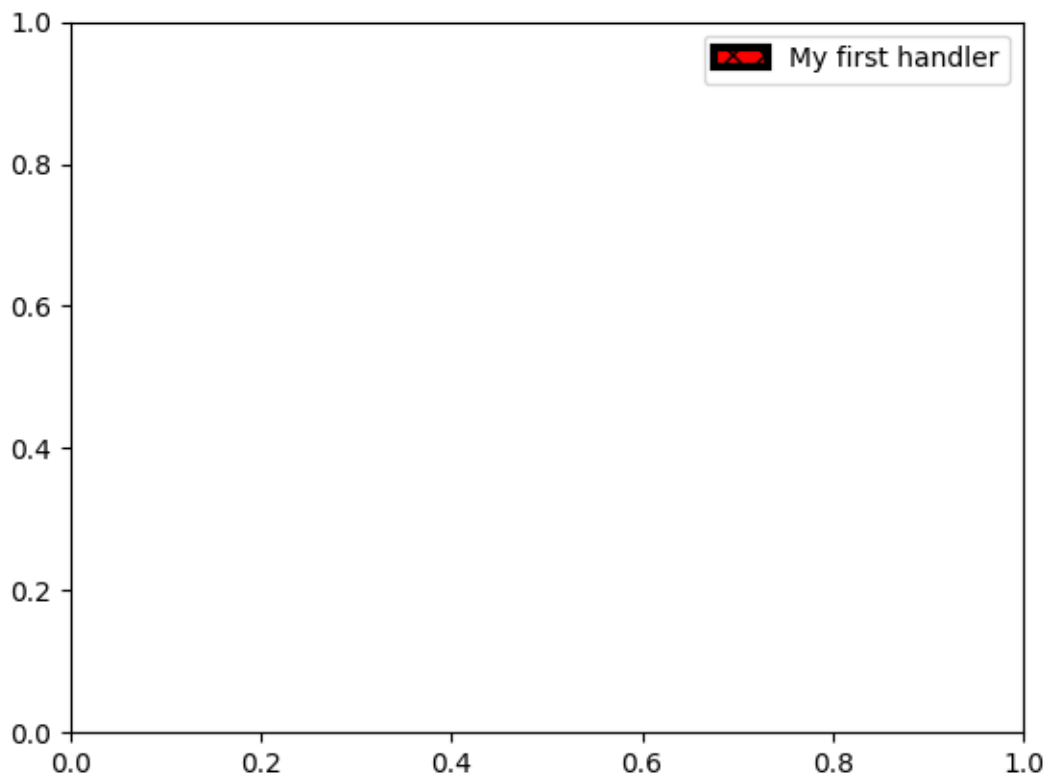
```

(continues on next page)

(continued from previous page)

```
class AnyObjectHandler(object):
    def legend_artist(self, legend, orig_handle, fontsize, handlebox):
        x0, y0 = handlebox.xdescent, handlebox.ydescent
        width, height = handlebox.width, handlebox.height
        patch = mpatches.Rectangle([x0, y0], width, height, facecolor='red',
                                   edgecolor='black', hatch='xx', lw=3,
                                   transform=handlebox.get_transform())
        handlebox.add_artist(patch)
        return patch

plt.legend([AnyObject()], ['My first handler'],
          handler_map={AnyObject: AnyObjectHandler()})
```



Alternatively, had we wanted to globally accept `AnyObject` instances without needing to manually set the `handler_map` keyword all the time, we could have registered the new handler with:

```
from matplotlib.legend import Legend
Legend.update_default_handler_map({AnyObject: AnyObjectHandler()})
```

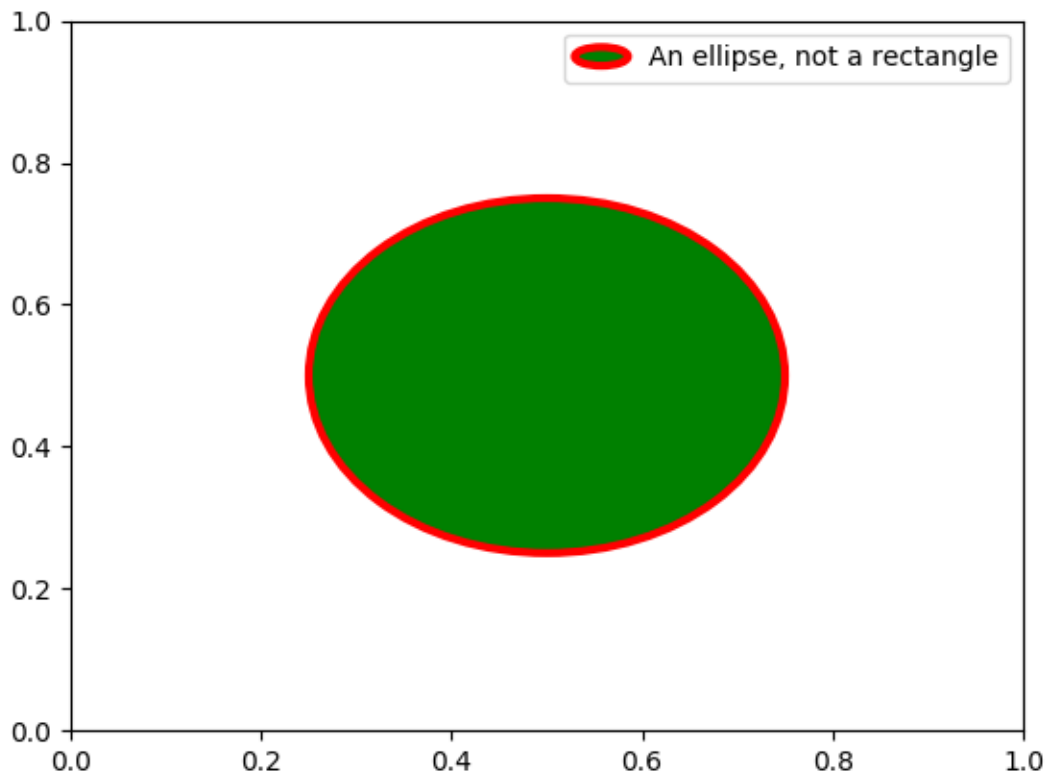
Whilst the power here is clear, remember that there are already many handlers implemented and what you want to achieve may already be easily possible with existing classes. For example, to produce elliptical legend keys, rather than rectangular ones:

```
from matplotlib.legend_handler import HandlerPatch

class HandlerEllipse(HandlerPatch):
    def create_artists(self, legend, orig_handle,
                      xdescent, ydescent, width, height, fontsize, trans):
        center = 0.5 * width - 0.5 * xdescent, 0.5 * height - 0.5 * ydescent
        p = mpatches.Ellipse(xy=center, width=width + xdescent,
                             height=height + ydescent)
        self.update_prop(p, orig_handle, legend)
        p.set_transform(trans)
        return [p]

c = mpatches.Circle((0.5, 0.5), 0.25, facecolor="green",
                    edgecolor="red", linewidth=3)
plt.gca().add_patch(c)

plt.legend([c], ["An ellipse, not a rectangle"],
           handler_map={mpatches.Circle: HandlerEllipse()})
```

Note: [Click here to download the full example code](#)

3.2.3 Artist tutorial

Using Artist objects to render on the canvas.

There are three layers to the matplotlib API.

- the `matplotlib.backend_bases.FigureCanvas` is the area onto which the figure is drawn
- the `matplotlib.backend_bases.Renderer` is the object which knows how to draw on the `FigureCanvas`
- and the `matplotlib.artist.Artist` is the object that knows how to use a renderer to paint onto the canvas.

The `FigureCanvas` and `Renderer` handle all the details of talking to user interface toolkits like `wxPython` or drawing languages like `PostScript®`, and the `Artist` handles all the high level constructs like representing and laying out the figure, text, and lines. The typical user will spend 95% of their time working with the Artists.

There are two types of Artists: primitives and containers. The primitives represent the standard graphical objects we want to paint onto our canvas: Line2D, Rectangle, Text, AxesImage, etc., and the containers are places to put them (Axis, Axes and Figure). The standard use is to create a Figure instance, use the Figure to create one or more Axes or Subplot instances, and use the Axes instance helper methods to create the primitives. In the example below, we create a Figure instance using `matplotlib.pyplot.figure()`, which is a convenience method for instantiating Figure instances and connecting them with your user interface or drawing toolkit FigureCanvas. As we will discuss below, this is not necessary – you can work directly with PostScript, PDF Gtk+, or wxPython FigureCanvas instances, instantiate your Figures directly and connect them yourselves – but since we are focusing here on the Artist API we'll let pyplot handle some of those details for us:

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(2,1,1) # two rows, one column, first plot
```

The Axes is probably the most important class in the matplotlib API, and the one you will be working with most of the time. This is because the Axes is the plotting area into which most of the objects go, and the Axes has many special helper methods (`plot()`, `text()`, `hist()`, `imshow()`) to create the most common graphics primitives (Line2D, Text, Rectangle, Image, respectively). These helper methods will take your data (e.g., numpy arrays and strings) and create primitive Artist instances as needed (e.g., Line2D), add them to the relevant containers, and draw them when requested. Most of you are probably familiar with the Subplot, which is just a special case of an Axes that lives on a regular rows by columns grid of Subplot instances. If you want to create an Axes at an arbitrary location, simply use the `add_axes()` method which takes a list of [left, bottom, width, height] values in 0-1 relative figure coordinates:

```
fig2 = plt.figure()
ax2 = fig2.add_axes([0.15, 0.1, 0.7, 0.3])
```

Continuing with our example:

```
import numpy as np
t = np.arange(0.0, 1.0, 0.01)
s = np.sin(2*np.pi*t)
line, = ax.plot(t, s, color='blue', lw=2)
```

In this example, `ax` is the Axes instance created by the `fig.add_subplot` call above (remember Subplot is just a subclass of Axes) and when you call `ax.plot`, it creates a Line2D instance and adds it to the `Axes.lines` list. In the interactive `ipython` session below, you can see that the `Axes.lines` list is length one and contains the same line that was returned by the `line, = ax.plot...` call:

```
In [101]: ax.lines[0]
Out[101]: <matplotlib.lines.Line2D instance at 0x19a95710>

In [102]: line
Out[102]: <matplotlib.lines.Line2D instance at 0x19a95710>
```

If you make subsequent calls to `ax.plot` (and the hold state is "on" which is the default) then

additional lines will be added to the list. You can remove lines later simply by calling the list methods; either of these will work:

```
del ax.lines[0]
ax.lines.remove(line) # one or the other, not both!
```

The Axes also has helper methods to configure and decorate the x-axis and y-axis tick, tick labels and axis labels:

```
xtext = ax.set_xlabel('my xdata') # returns a Text instance
ytext = ax.set_ylabel('my ydata')
```

When you call `ax.set_xlabel`, it passes the information on the Text instance of the XAxis. Each Axes instance contains an XAxis and a YAxis instance, which handle the layout and drawing of the ticks, tick labels and axis labels.

Try creating the figure below.

```
import numpy as np
import matplotlib.pyplot as plt

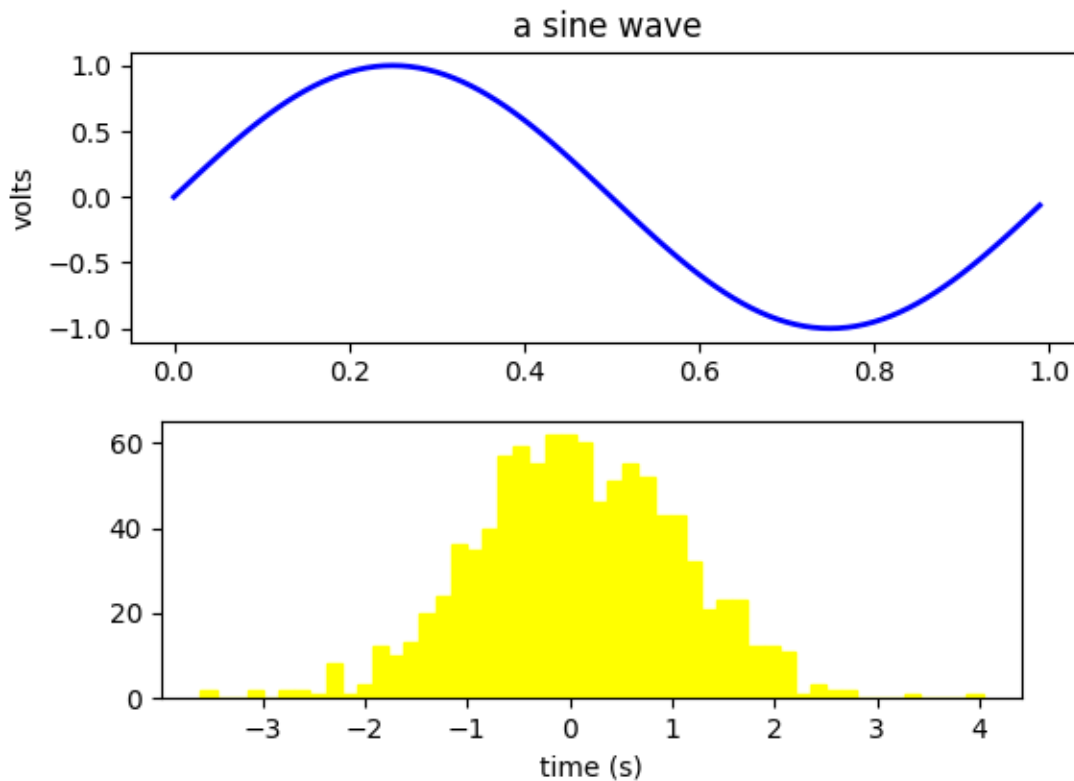
fig = plt.figure()
fig.subplots_adjust(top=0.8)
ax1 = fig.add_subplot(211)
ax1.set_ylabel('volts')
ax1.set_title('a sine wave')

t = np.arange(0.0, 1.0, 0.01)
s = np.sin(2*np.pi*t)
line, = ax1.plot(t, s, color='blue', lw=2)

# Fixing random state for reproducibility
np.random.seed(19680801)

ax2 = fig.add_axes([0.15, 0.1, 0.7, 0.3])
n, bins, patches = ax2.hist(np.random.randn(1000), 50,
                             facecolor='yellow', edgecolor='yellow')
ax2.set_xlabel('time (s)')

plt.show()
```



Customizing your objects

Every element in the figure is represented by a matplotlib Artist, and each has an extensive list of properties to configure its appearance. The figure itself contains a Rectangle exactly the size of the figure, which you can use to set the background color and transparency of the figures. Likewise, each Axes bounding box (the standard white box with black edges in the typical matplotlib plot, has a Rectangle instance that determines the color, transparency, and other properties of the Axes. These instances are stored as member variables `Figure.patch` and `Axes.patch` ("Patch" is a name inherited from MATLAB, and is a 2D "patch" of color on the figure, e.g., rectangles, circles and polygons). Every matplotlib Artist has the following properties

Property	Description
alpha	The transparency - a scalar from 0-1
animated	A boolean that is used to facilitate animated drawing
axes	The axes that the Artist lives in, possibly None
clip_box	The bounding box that clips the Artist
clip_on	Whether clipping is enabled
clip_path	The path the artist is clipped to
contains	A picking function to test whether the artist contains the pick point
figure	The figure instance the artist lives in, possibly None
label	A text label (e.g., for auto-labeling)
picker	A python object that controls object picking
transform	The transformation
visible	A boolean whether the artist should be drawn
zorder	A number which determines the drawing order
rasterized	Boolean; Turns vectors into rastergraphics: (for compression & eps transparency)

Each of the properties is accessed with an old-fashioned setter or getter (yes we know this irritates Pythonistas and we plan to support direct access via properties or traits but it hasn't been done yet). For example, to multiply the current alpha by a half:

```
a = o.get_alpha()
o.set_alpha(0.5*a)
```

If you want to set a number of properties at once, you can also use the set method with keyword arguments. For example:

```
o.set(alpha=0.5, zorder=2)
```

If you are working interactively at the python shell, a handy way to inspect the Artist properties is to use the `matplotlib.artist.getp()` function (simply `getp()` in `pylab`), which lists the properties and their values. This works for classes derived from `Artist` as well, e.g., `Figure` and `Rectangle`. Here are the `Figure` rectangle properties mentioned above:

```
In [149]: matplotlib.artist.getp(fig.patch)
alpha = 1.0
animated = False
antialiased or aa = True
axes = None
clip_box = None
clip_on = False
clip_path = None
contains = None
edgecolor or ec = w
facecolor or fc = 0.75
figure = Figure(8.125x6.125)
fill = 1
```

(continues on next page)

(continued from previous page)

```

hatch = None
height = 1
label =
linewidth or lw = 1.0
picker = None
transform = <Affine object at 0x134cca84>
verts = ((0, 0), (0, 1), (1, 1), (1, 0))
visible = True
width = 1
window_extent = <Bbox object at 0x134acb2c>
x = 0
y = 0
zorder = 1

```

The docstrings for all of the classes also contain the Artist properties, so you can consult the interactive "help" or the artist Module for a listing of properties for a given object.

Object containers

Now that we know how to inspect and set the properties of a given object we want to configure, we need to know how to get at that object. As mentioned in the introduction, there are two kinds of objects: primitives and containers. The primitives are usually the things you want to configure (the font of a Text instance, the width of a Line2D) although the containers also have some properties as well – for example the Axes Artist is a container that contains many of the primitives in your plot, but it also has properties like the xscale to control whether the xaxis is 'linear' or 'log'. In this section we'll review where the various container objects store the Artists that you want to get at.

Figure container

The top level container Artist is the `matplotlib.figure.Figure`, and it contains everything in the figure. The background of the figure is a Rectangle which is stored in `Figure.patch`. As you add subplots (`add_subplot()`) and axes (`add_axes()`) to the figure these will be appended to the `Figure.axes`. These are also returned by the methods that create them:

```

In [156]: fig = plt.figure()

In [157]: ax1 = fig.add_subplot(211)

In [158]: ax2 = fig.add_axes([0.1, 0.1, 0.7, 0.3])

In [159]: ax1
Out[159]: <matplotlib.axes.Subplot instance at 0xd54b26c>

```

(continues on next page)

(continued from previous page)

```
In [160]: print(fig.axes)
[<matplotlib.axes.Subplot instance at 0xd54b26c>, <matplotlib.axes.Axes instance at 0xd3f0b2c>]
```

Because the figure maintains the concept of the "current axes" (see `Figure.gca` and `Figure.sca`) to support the pylab/pyplot state machine, you should not insert or remove axes directly from the axes list, but rather use the `add_subplot()` and `add_axes()` methods to insert, and the `delaxes()` method to delete. You are free however, to iterate over the list of axes or index into it to get access to Axes instances you want to customize. Here is an example which turns all the axes grids on:

```
for ax in fig.axes:
    ax.grid(True)
```

The figure also has its own text, lines, patches and images, which you can use to add primitives directly. The default coordinate system for the Figure will simply be in pixels (which is not usually what you want) but you can control this by setting the transform property of the Artist you are adding to the figure.

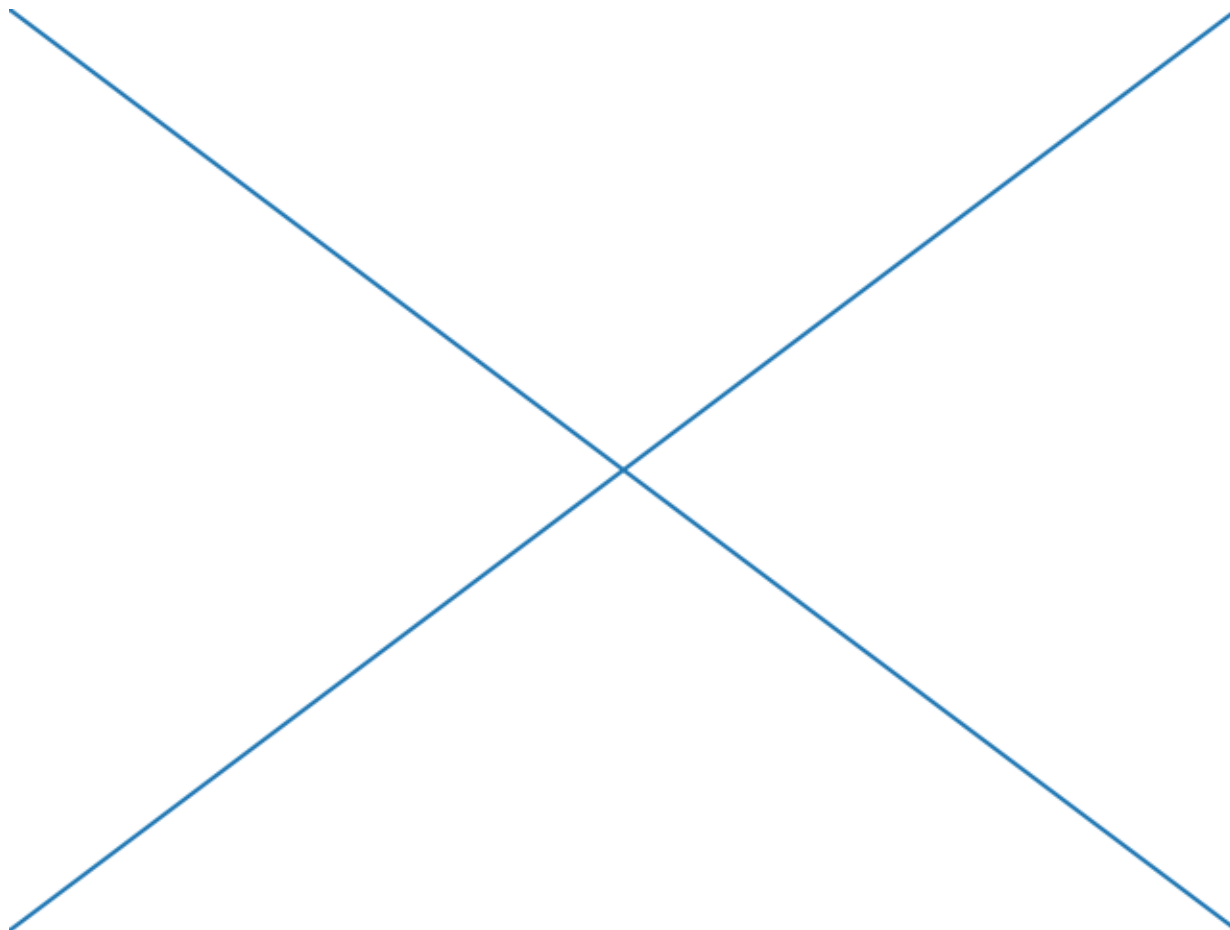
More useful is "figure coordinates" where (0, 0) is the bottom-left of the figure and (1, 1) is the top-right of the figure which you can obtain by setting the Artist transform to `fig.transFigure`:

```
import matplotlib.lines as lines

fig = plt.figure()

l1 = lines.Line2D([0, 1], [0, 1], transform=fig.transFigure, figure=fig)
l2 = lines.Line2D([0, 1], [1, 0], transform=fig.transFigure, figure=fig)
fig.lines.extend([l1, l2])

plt.show()
```



Here is a summary of the Artists the figure contains

Figure attribute	Description
axes	A list of Axes instances (includes Subplot)
patch	The Rectangle background
images	A list of FigureImage patches - useful for raw pixel display
legends	A list of Figure Legend instances (different from Axes.legends)
lines	A list of Figure Line2D instances (rarely used, see Axes.lines)
patches	A list of Figure patches (rarely used, see Axes.patches)
texts	A list Figure Text instances

Axes container

The `matplotlib.axes.Axes` is the center of the matplotlib universe – it contains the vast majority of all the Artists used in a figure with many helper methods to create and add these Artists to itself, as well as helper methods to access and customize the Artists it contains. Like the Figure, it contains a Patch patch which is a Rectangle for Cartesian coordinates and a Circle for polar coordinates; this patch determines the shape, background and border of the plotting region:


```
ax = fig.add_subplot(111)
rect = ax.patch # a Rectangle instance
rect.set_facecolor('green')
```

When you call a plotting method, e.g., the canonical `plot()` and pass in arrays or lists of values, the method will create a `matplotlib.lines.Line2D()` instance, update the line with all the `Line2D` properties passed as keyword arguments, add the line to the `Axes.lines` container, and returns it to you:

```
In [213]: x, y = np.random.rand(2, 100)

In [214]: line, = ax.plot(x, y, '-', color='blue', linewidth=2)
```

`plot` returns a list of lines because you can pass in multiple `x, y` pairs to `plot`, and we are unpacking the first element of the length one list into the `line` variable. The line has been added to the `Axes.lines` list:

```
In [229]: print(ax.lines)
[<matplotlib.lines.Line2D instance at 0xd378b0c>]
```

Similarly, methods that create patches, like `bar()` creates a list of rectangles, will add the patches to the `Axes.patches` list:

```
In [233]: n, bins, rectangles = ax.hist(np.random.randn(1000), 50, facecolor='yellow')

In [234]: rectangles
Out[234]: <a list of 50 Patch objects>

In [235]: print(len(ax.patches))
```

You should not add objects directly to the `Axes.lines` or `Axes.patches` lists unless you know exactly what you are doing, because the `Axes` needs to do a few things when it creates and adds an object. It sets the figure and axes property of the `Artist`, as well as the default `Axes` transformation (unless a transformation is set). It also inspects the data contained in the `Artist` to update the data structures controlling auto-scaling, so that the view limits can be adjusted to contain the plotted data. You can, nonetheless, create objects yourself and add them directly to the `Axes` using helper methods like `add_line()` and `add_patch()`. Here is an annotated interactive session illustrating what is going on:

```
In [262]: fig, ax = plt.subplots()

# create a rectangle instance
In [263]: rect = matplotlib.patches.Rectangle( (1,1), width=5, height=12)

# by default the axes instance is None
In [264]: print(rect.get_axes())
None
```

(continues on next page)

(continued from previous page)

```
# and the transformation instance is set to the "identity transform"
In [265]: print(rect.get_transform())
<Affine object at 0x13695544>

# now we add the Rectangle to the Axes
In [266]: ax.add_patch(rect)

# and notice that the ax.add_patch method has set the axes
# instance
In [267]: print(rect.get_axes())
Axes(0.125,0.1;0.775x0.8)

# and the transformation has been set too
In [268]: print(rect.get_transform())
<Affine object at 0x15009ca4>

# the default axes transformation is ax.transData
In [269]: print(ax.transData)
<Affine object at 0x15009ca4>

# notice that the xlims of the Axes have not been changed
In [270]: print(ax.get_xlim())
(0.0, 1.0)

# but the data limits have been updated to encompass the rectangle
In [271]: print(ax.dataLim.bounds)
(1.0, 1.0, 5.0, 12.0)

# we can manually invoke the auto-scaling machinery
In [272]: ax.autoscale_view()

# and now the xlim are updated to encompass the rectangle
In [273]: print(ax.get_xlim())
(1.0, 6.0)

# we have to manually force a figure draw
In [274]: ax.figure.canvas.draw()
```

There are many, many Axes helper methods for creating primitive Artists and adding them to their respective containers. The table below summarizes a small sampling of them, the kinds of Artist they create, and where they store them

Helper method	Artist	Container
ax.annotate - text annotations	Annotate	ax.texts
ax.bar - bar charts	Rectangle	ax.patches
ax.errorbar - error bar plots	Line2D and Rectangle	ax.lines and ax.patches
ax.fill - shared area	Polygon	ax.patches
ax.hist - histograms	Rectangle	ax.patches
ax.imshow - image data	AxesImage	ax.images
ax.legend - axes legends	Legend	ax.legend
ax.plot - xy plots	Line2D	ax.lines
ax.scatter - scatter charts	PolygonCollection	ax.collections
ax.text - text	Text	ax.texts

In addition to all of these Artists, the Axes contains two important Artist containers: the XAxis and YAxis, which handle the drawing of the ticks and labels. These are stored as instance variables `xaxis` and `yaxis`. The XAxis and YAxis containers will be detailed below, but note that the Axes contains many helper methods which forward calls on to the Axis instances so you often do not need to work with them directly unless you want to. For example, you can set the font color of the XAxis ticklabels using the Axes helper method:

```
for label in ax.get_xticklabels():  
    label.set_color('orange')
```

Below is a summary of the Artists that the Axes contains

Axes attribute	Description
<code>artists</code>	A list of Artist instances
<code>patch</code>	Rectangle instance for Axes background
<code>collections</code>	A list of Collection instances
<code>images</code>	A list of AxesImage
<code>legends</code>	A list of Legend instances
<code>lines</code>	A list of Line2D instances
<code>patches</code>	A list of Patch instances
<code>texts</code>	A list of Text instances
<code>xaxis</code>	<code>matplotlib.axis.XAxis</code> instance
<code>yaxis</code>	<code>matplotlib.axis.YAxis</code> instance

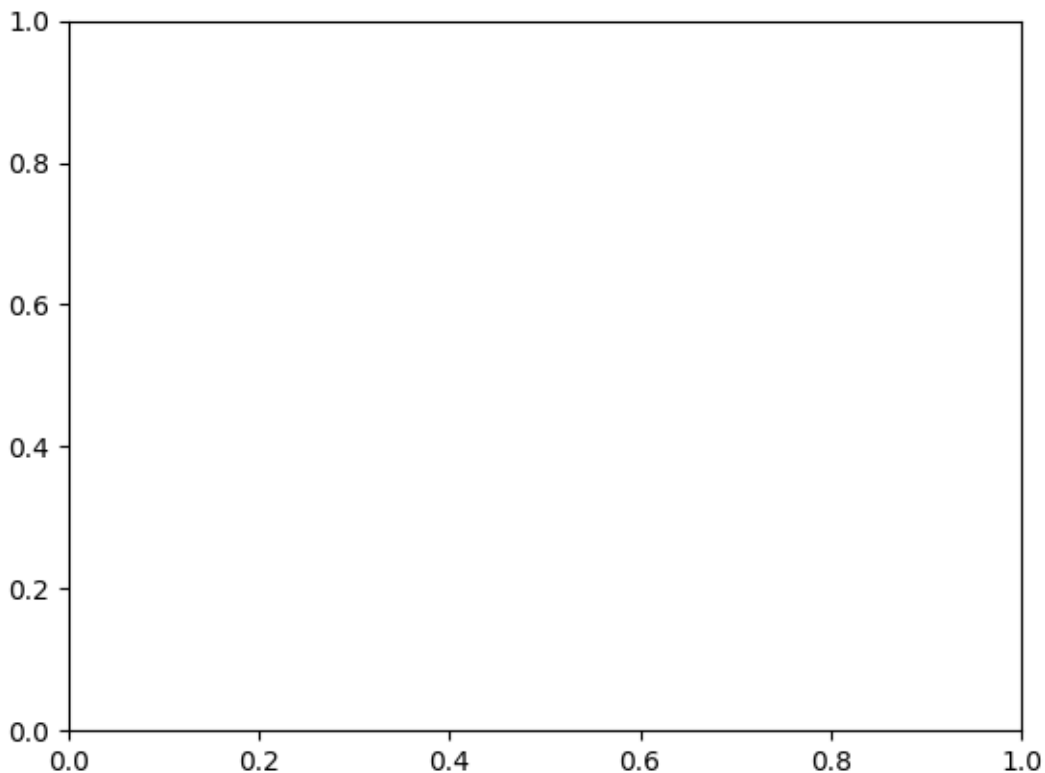
Axis containers

The `matplotlib.axis.Axis` instances handle the drawing of the tick lines, the grid lines, the tick labels and the axis label. You can configure the left and right ticks separately for the y-axis, and the upper and lower ticks separately for the x-axis. The Axis also stores the data and view intervals used in auto-scaling, panning and zooming, as well as the Locator and Formatter instances which control where the ticks are placed and how they are represented as strings.

Each Axis object contains a label attribute (this is what pylab modifies in calls to `xlabel()` and `ylabel()`) as well as a list of major and minor ticks. The ticks are XTick and YTick instances, which

contain the actual line and text primitives that render the ticks and ticklabels. Because the ticks are dynamically created as needed (e.g., when panning and zooming), you should access the lists of major and minor ticks through their accessor methods `get_major_ticks()` and `get_minor_ticks()`. Although the ticks contain all the primitives and will be covered below, Axis instances have accessor methods that return the tick lines, tick labels, tick locations etc.:

```
fig, ax = plt.subplots()
axis = ax.xaxis
axis.get_ticklocs()
```



```
axis.get_ticklabels()
```

note there are twice as many ticklines as labels because by default there are tick lines at the top and bottom but only tick labels below the xaxis; this can be customized

```
axis.get_ticklines()
```

by default you get the major ticks back

```
axis.get_ticklines()
```

but you can also ask for the minor ticks

```
axis.get_ticklines(minor=True)
```

```
# Here is a summary of some of the useful accessor methods of the ``Axis``
```

```
# (these have corresponding setters where useful, such as
```

```
# set_major_formatter)
```

```
#
```

```
# =====
```

```
→ =====
```

```
# Accessor method      Description
```

```
# =====
```

```
→ =====
```

```
# get_scale            The scale of the axis, e.g., 'log' or 'linear'
```

```
# get_view_interval    The interval instance of the axis view limits
```

```
# get_data_interval    The interval instance of the axis data limits
```

```
# get_gridlines        A list of grid lines for the Axis
```

```
# get_label            The axis label - a Text instance
```

```
# get_ticklabels       A list of Text instances - keyword minor=True|False
```

```
# get_ticklines        A list of Line2D instances - keyword minor=True|False
```

```
# get_ticklocs         A list of Tick locations - keyword minor=True|False
```

```
# get_major_locator    The matplotlib.ticker.Locator instance for major ticks
```

```
# get_major_formatter  The matplotlib.ticker.Formatter instance for major ticks
```

```
# get_minor_locator    The matplotlib.ticker.Locator instance for minor ticks
```

```
# get_minor_formatter  The matplotlib.ticker.Formatter instance for minor ticks
```

```
# get_major_ticks      A list of Tick instances for major ticks
```

```
# get_minor_ticks      A list of Tick instances for minor ticks
```

```
# grid                Turn the grid on or off for the major or minor ticks
```

```
# =====
```

```
→ =====
```

```
#
```

```
# Here is an example, not recommended for its beauty, which customizes
```

```
# the axes and tick properties
```

```
# plt.figure creates a matplotlib.figure.Figure instance
```

```
fig = plt.figure()
```

```
rect = fig.patch # a rectangle instance
```

```
rect.set_facecolor('lightgoldenrodyellow')
```

```
ax1 = fig.add_axes([0.1, 0.3, 0.4, 0.4])
```

```
rect = ax1.patch
```

```
rect.set_facecolor('lightslategray')
```

```
for label in ax1.xaxis.get_ticklabels():
```

```
    # label is a Text instance
```

```
    label.set_color('red')
```

```
    label.set_rotation(45)
```

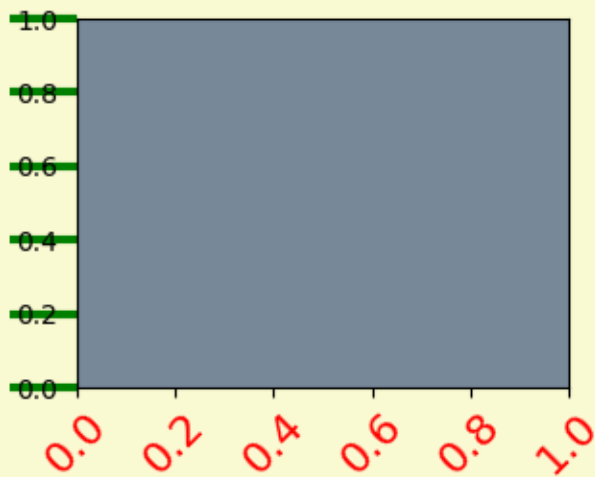
(continues on next page)

(continued from previous page)

```
label.set_fontsize(16)

for line in ax1.yaxis.get_ticklines():
    # line is a Line2D instance
    line.set_color('green')
    line.set_markersize(25)
    line.set_markedgewidth(3)

plt.show()
```



Tick containers

The `matplotlib.axis.Tick` is the final container object in our descent from the Figure to the Axes to the Axis to the Tick. The Tick contains the tick and grid line instances, as well as the label instances for the upper and lower ticks. Each of these is accessible directly as an attribute of the Tick. In addition, there are boolean variables that determine whether the upper labels and ticks are on for the x-axis and whether the right labels and ticks are on for the y-axis.

Tick attribute	Description
tick1line	Line2D instance
tick2line	Line2D instance
gridline	Line2D instance
label1	Text instance
label2	Text instance
gridOn	boolean which determines whether to draw the gridline
tick1On	boolean which determines whether to draw the 1st tickline
tick2On	boolean which determines whether to draw the 2nd tickline
label1On	boolean which determines whether to draw the 1st tick label
label2On	boolean which determines whether to draw the 2nd tick label

Here is an example which sets the formatter for the right side ticks with dollar signs and colors them green on the right side of the yaxis

```
import matplotlib.ticker as ticker

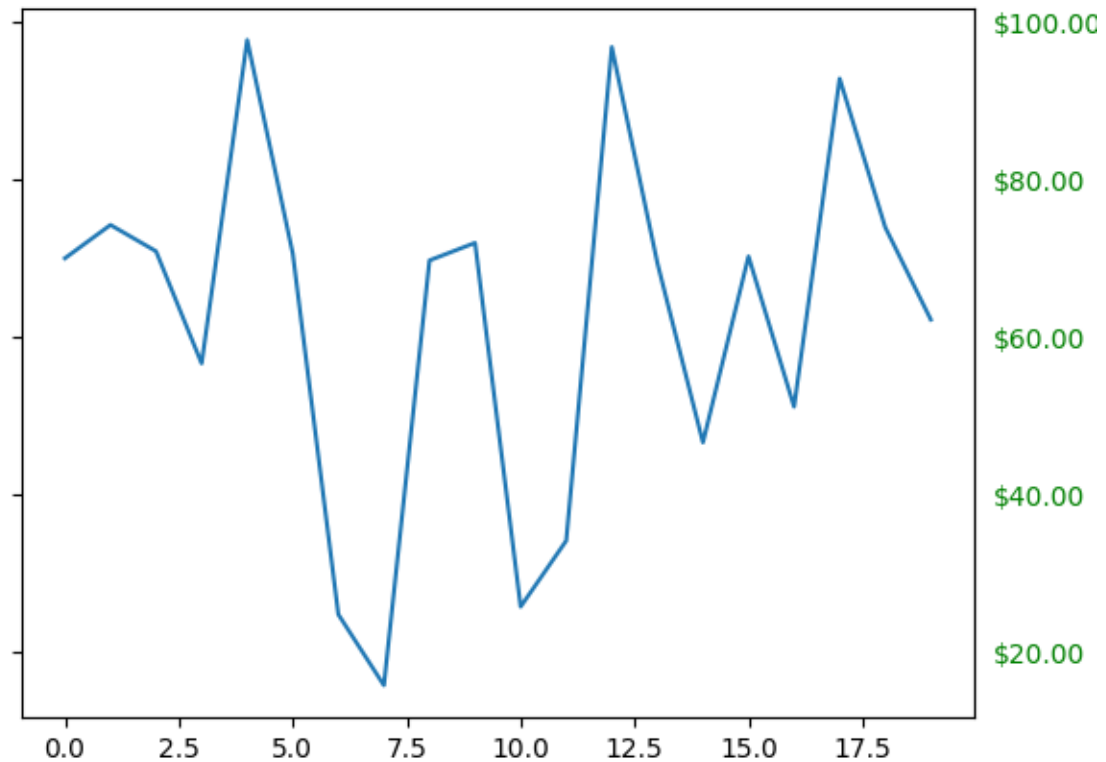
# Fixing random state for reproducibility
np.random.seed(19680801)

fig, ax = plt.subplots()
ax.plot(100*np.random.rand(20))

formatter = ticker.FormatStrFormatter('$%1.2f')
ax.yaxis.set_major_formatter(formatter)

for tick in ax.yaxis.get_major_ticks():
    tick.label1On = False
    tick.label2On = True
    tick.label2.set_color('green')

plt.show()
```



Note: [Click here to download the full example code](#)

3.2.4 Customizing Figure Layouts Using GridSpec and Other Functions

How to create grid-shaped combinations of axes.

`subplots()` Perhaps the primary function used to create figures and axes. It's also similar to `subplot()`, but creates and places all axes on the figure at once.

`GridSpec` Specifies the geometry of the grid that a subplot will be placed. The number of rows and number of columns of the grid need to be set. Optionally, the subplot layout parameters (e.g., left, right, etc.) can be tuned.

`SubplotSpec` Specifies the location of the subplot in the given `GridSpec`.

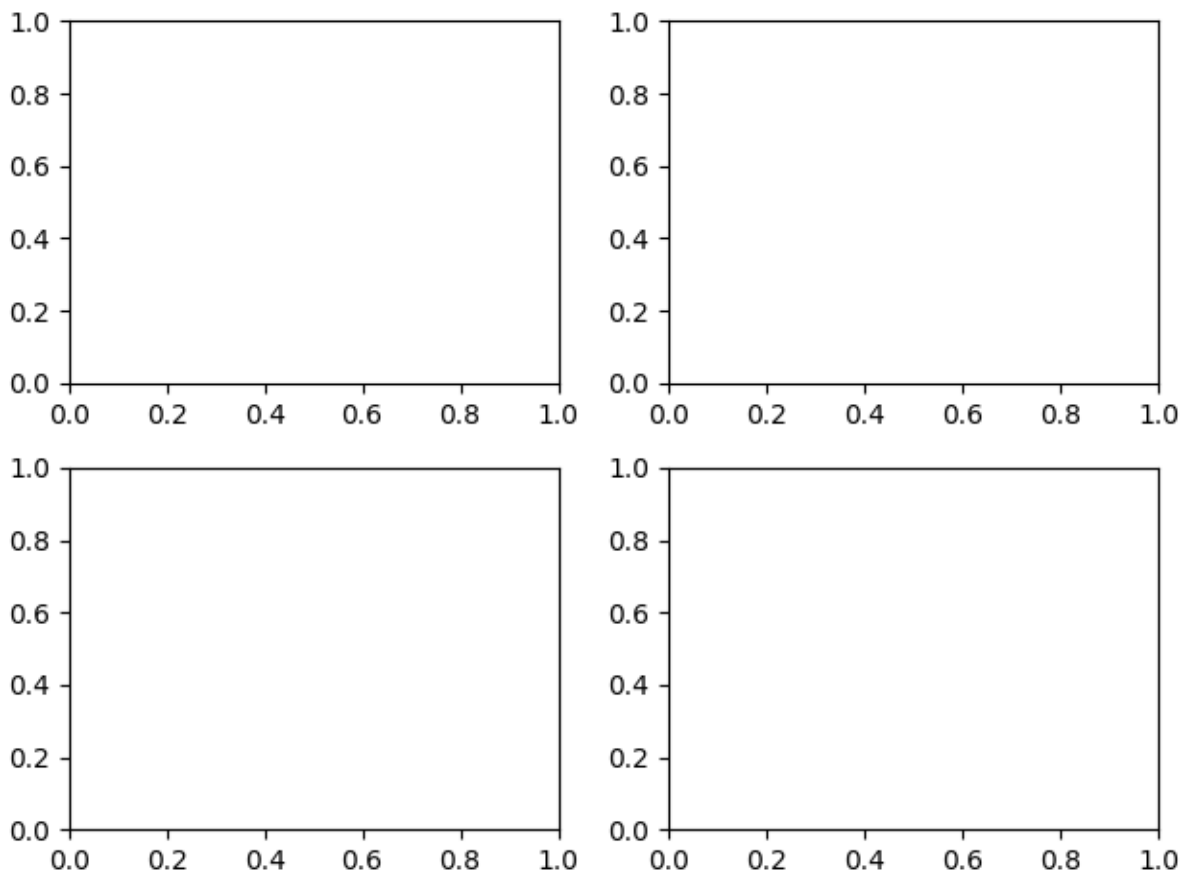
`subplot2grid()` A helper function that is similar to `subplot()`, but uses 0-based indexing and let subplot to occupy multiple cells. This function is not covered in this tutorial.


```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
```

Basic Quickstart Guide

These first two examples show how to create a basic 4-by-4 grid using both `subplots()` and `gridspec`. Using `subplots()` is quite simple. It returns a `Figure` instance and an array of `Axes` objects.

```
fig1, f1_axes = plt.subplots(ncols=2, rows=2)
fig1.tight_layout()
```



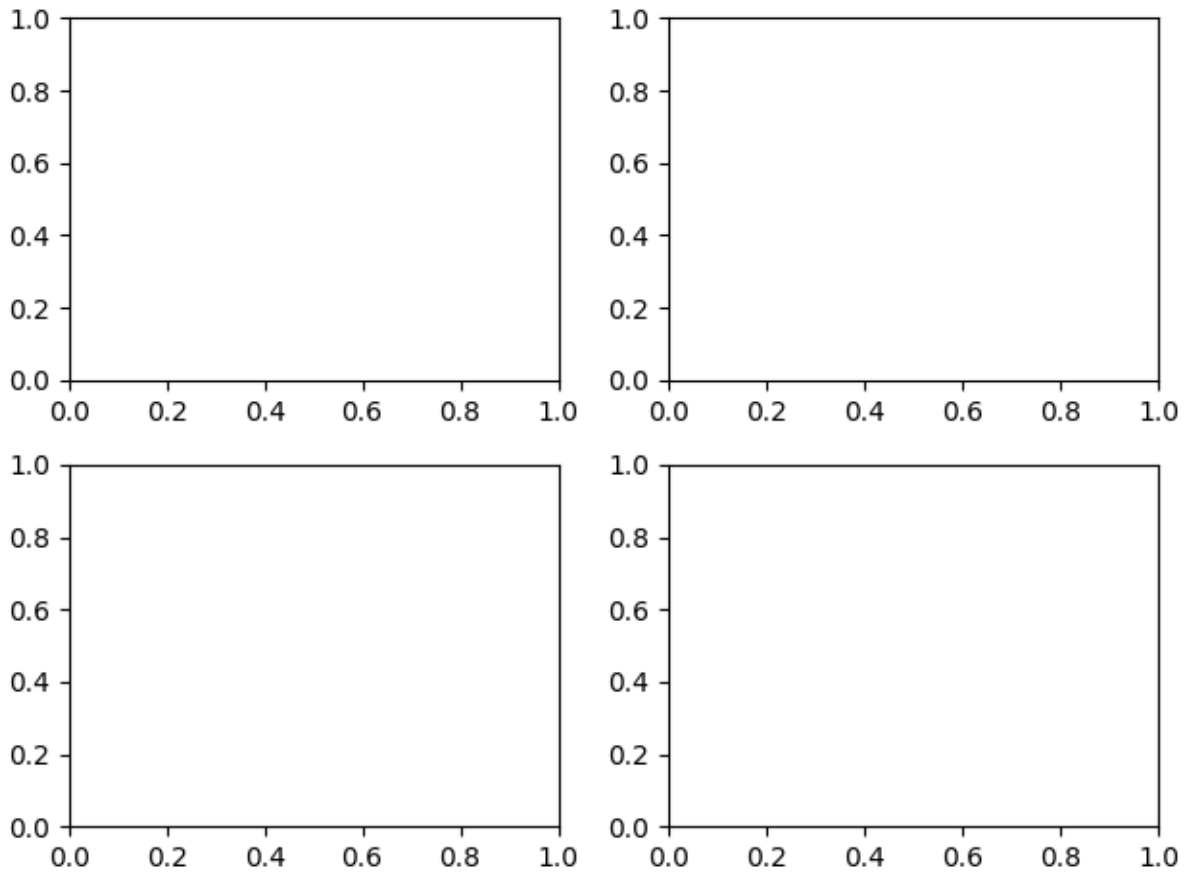
For a simple use case such as this, `gridspec` is perhaps overly verbose. You have to create the figure and `GridSpec` instance separately, then pass elements of `gridspec` instance to the `add_subplot()` method to create the axes objects. The elements of the `gridspec` are accessed in generally the same manner as numpy arrays.

```
fig2 = plt.figure()
spec2 = gridspec.GridSpec(ncols=2, rows=2)
f2_ax1 = fig2.add_subplot(spec2[0, 0])
```

(continues on next page)

(continued from previous page)

```
f2_ax2 = fig2.add_subplot(spec2[0, 1])
f2_ax3 = fig2.add_subplot(spec2[1, 0])
f2_ax4 = fig2.add_subplot(spec2[1, 1])
fig2.tight_layout()
```

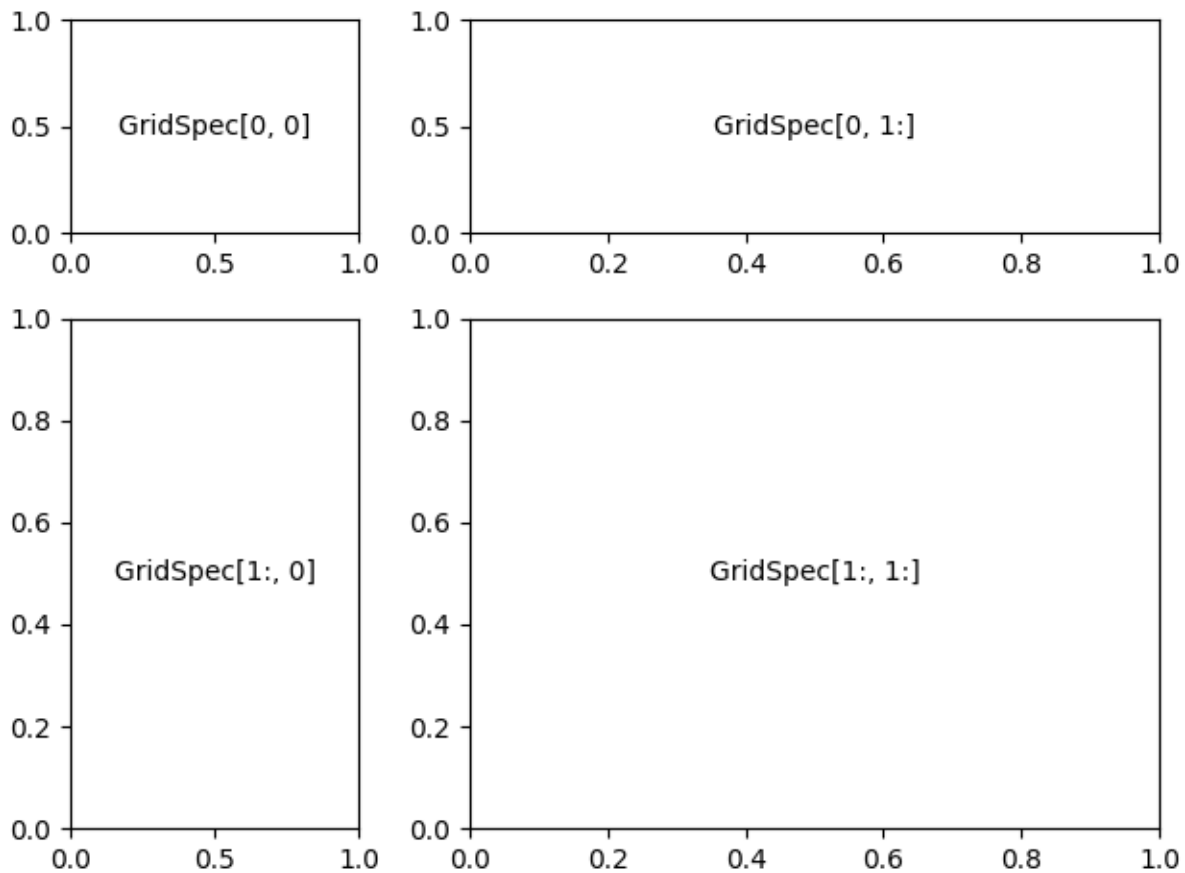


When you want to have subplots of different sizes, however, `gridspec` becomes indispensable and provides a couple of options. The method shown here initializes a uniform grid specification, and then uses typical numpy indexing and slices to allocate multiple "cells" for a given subplot.

```
fig3 = plt.figure()
spec3 = gridspec.GridSpec(ncols=3, nrows=3)
anno_opts = dict(xy=(0.5, 0.5), xycoords='axes fraction',
                  va='center', ha='center')

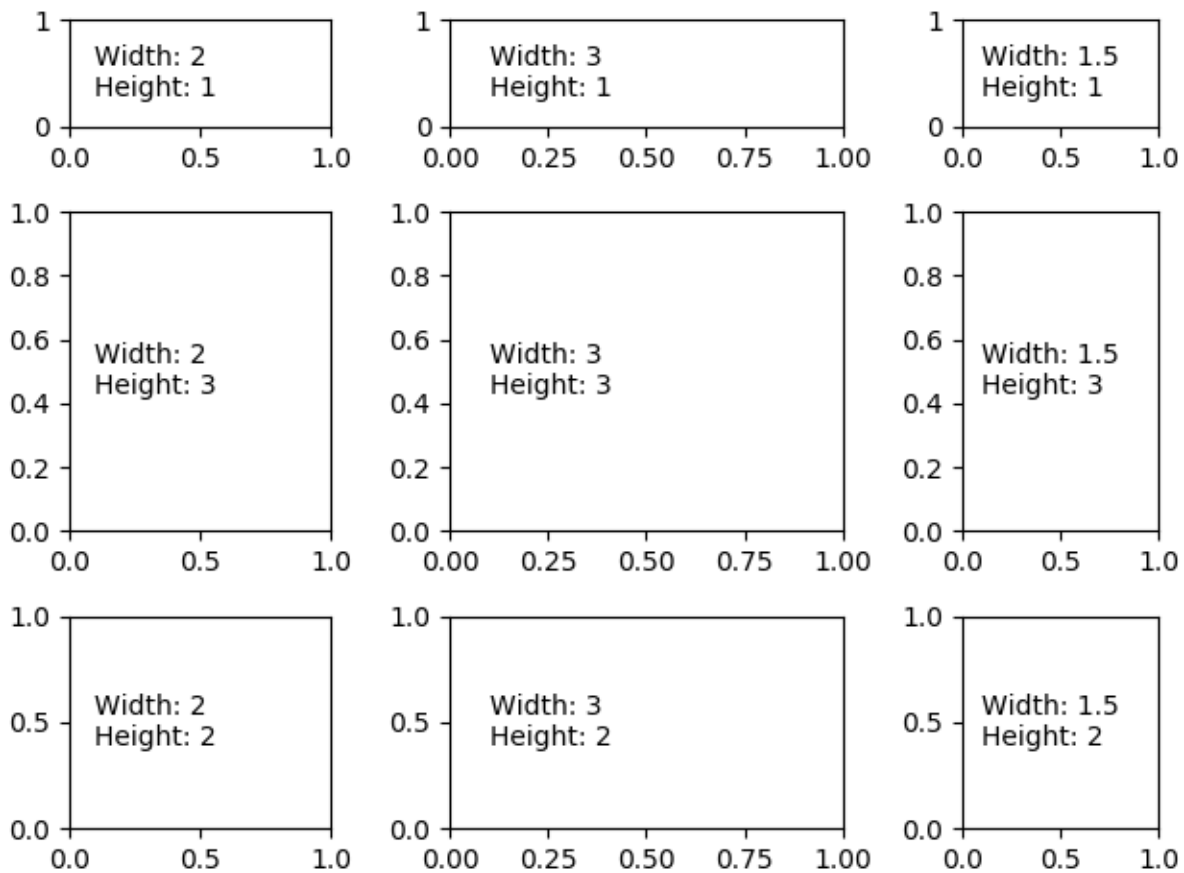
fig3.add_subplot(spec3[0, 0]).annotate('GridSpec[0, 0]', **anno_opts)
fig3.add_subplot(spec3[0, 1:]).annotate('GridSpec[0, 1:]', **anno_opts)
fig3.add_subplot(spec3[1:, 0]).annotate('GridSpec[1:, 0]', **anno_opts)
fig3.add_subplot(spec3[1:, 1:]).annotate('GridSpec[1:, 1:]', **anno_opts)

fig3.tight_layout()
```



Other option is to use the `width_ratios` and `height_ratios` parameters. These keyword arguments are lists of numbers. Note that absolute values are meaningless, only their relative ratios matter. That means that `width_ratios=[2, 4, 8]` is equivalent to `width_ratios=[1, 2, 4]` within equally wide figures. For the sake of demonstration, we'll blindly create the axes within for loops since we won't need them later.

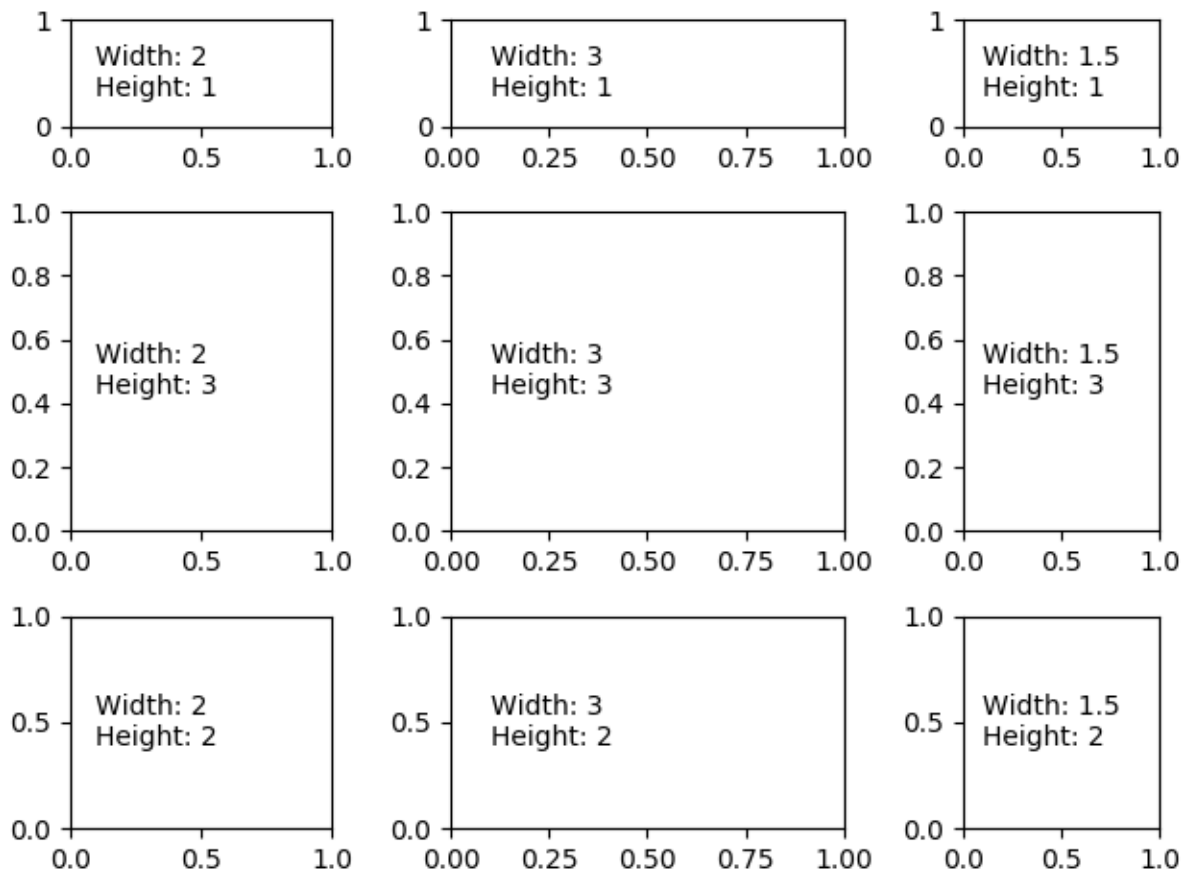
```
fig4 = plt.figure()
widths = [2, 3, 1.5]
heights = [1, 3, 2]
spec4 = gridspec.GridSpec(ncols=3, nrows=3, width_ratios=widths,
                           height_ratios=heights)
for row in range(3):
    for col in range(3):
        ax = fig4.add_subplot(spec4[row, col])
        label = 'Width: {} \n Height: {}'.format(widths[col], heights[row])
        ax.annotate(label, (0.1, 0.5), xycoords='axes fraction', va='center')
fig4.tight_layout()
```



Learning to use `width_ratios` and `height_ratios` is particularly useful since the top-level function `subplots()` accepts them within the `gridspec_kw` parameter. For that matter, any parameter accepted by `GridSpec` can be passed to `subplots()` via the `gridspec_kw` parameter. This example recreates the previous figure without directly using a `gridspec` instance.

```
gs_kw = dict(width_ratios=widths, height_ratios=heights)
fig5, f5_axes = plt.subplots(ncols=3, nrows=3, gridspec_kw=gs_kw)
for r, row in enumerate(f5_axes):
    for c, ax in enumerate(row):
        label = 'Width: {} \n Height: {}'.format(widths[c], heights[r])
        ax.annotate(label, (0.1, 0.5), xycoords='axes fraction', va='center')

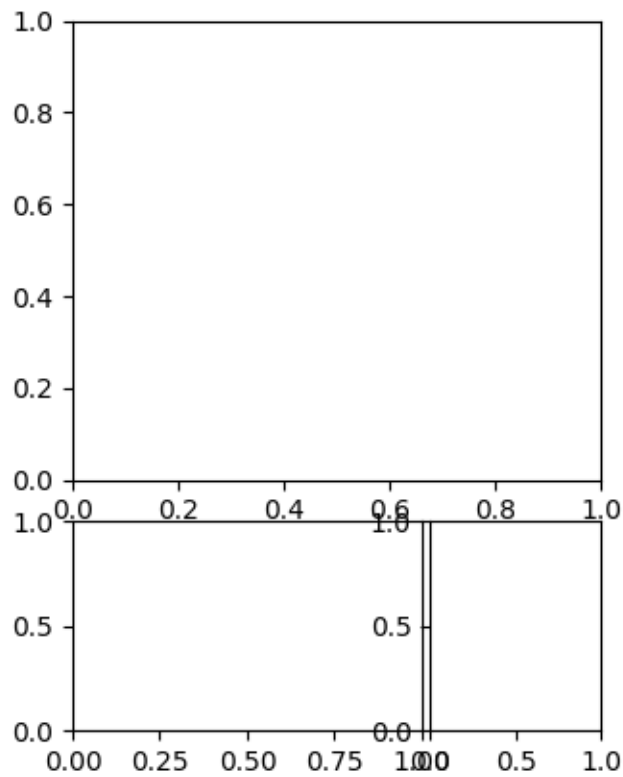
fig5.tight_layout()
```



Fine Adjustments to a Gridspec Layout

When a `GridSpec` is explicitly used, you can adjust the layout parameters of subplots that are created from the `GridSpec`.

```
fig = plt.figure()
gs1 = gridspec.GridSpec(nrows=3, ncols=3, left=0.05, right=0.48, wspace=0.05)
ax1 = fig.add_subplot(gs1[:-1, :])
ax2 = fig.add_subplot(gs1[-1, :-1])
ax3 = fig.add_subplot(gs1[-1, -1])
```

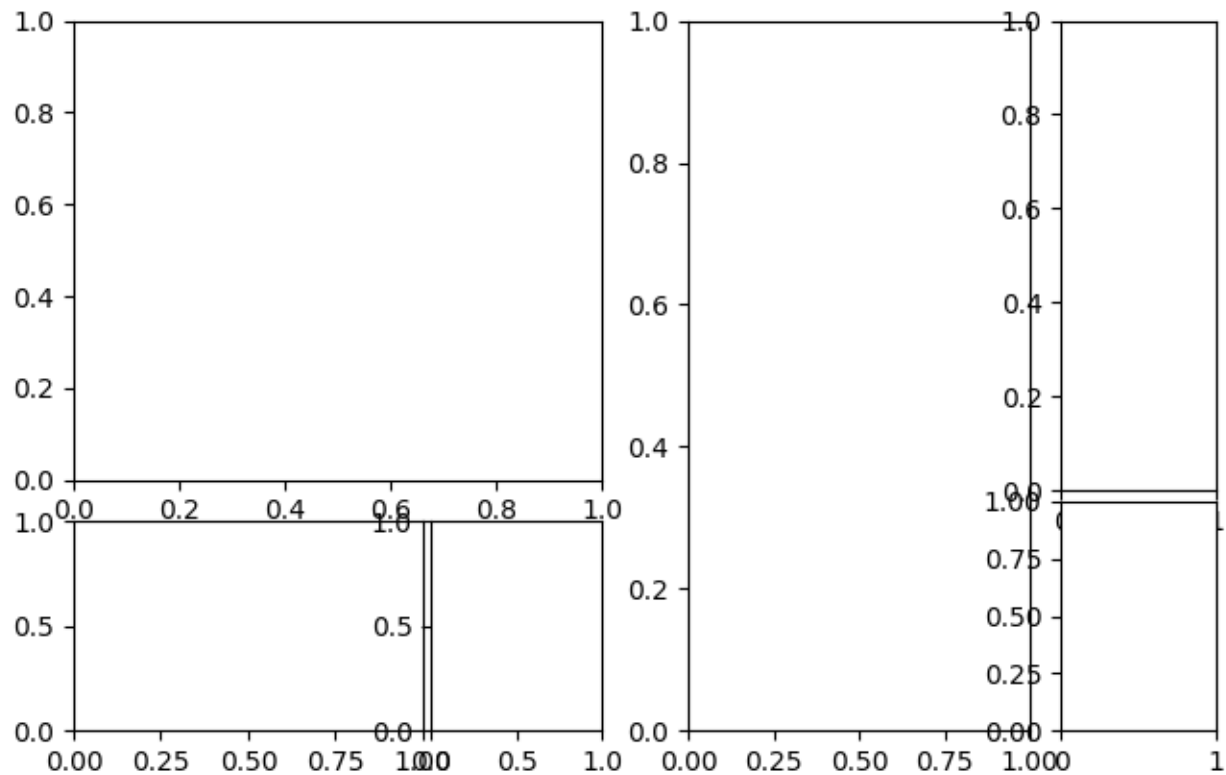


This is similar to `subplots_adjust()`, but it only affects the subplots that are created from the given `GridSpec`.

For example, compare the left and right sides of this figure:

```
fig = plt.figure()
gs1 = gridspec.GridSpec(nrows=3, ncols=3, left=0.05, right=0.48,
                        wspace=0.05)
ax1 = fig.add_subplot(gs1[:-1, :])
ax2 = fig.add_subplot(gs1[-1, :-1])
ax3 = fig.add_subplot(gs1[-1, -1])

gs2 = gridspec.GridSpec(nrows=3, ncols=3, left=0.55, right=0.98,
                        hspace=0.05)
ax4 = fig.add_subplot(gs2[:, :-1])
ax5 = fig.add_subplot(gs2[:-1, -1])
ax6 = fig.add_subplot(gs2[-1, -1])
```



GridSpec using SubplotSpec

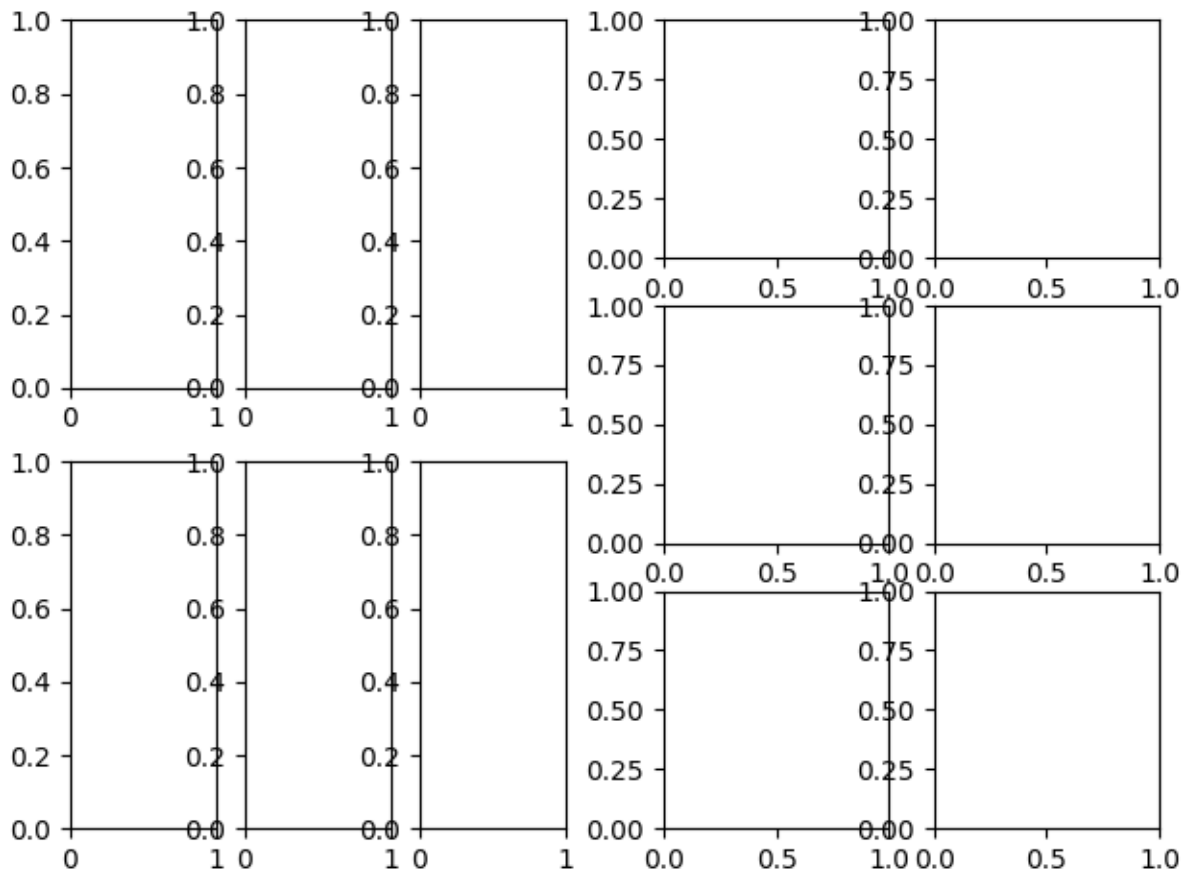
You can create GridSpec from the SubplotSpec, in which case its layout parameters are set to that of the location of the given SubplotSpec.

```
fig = plt.figure()
gs0 = gridspec.GridSpec(1, 2)

gs00 = gridspec.GridSpecFromSubplotSpec(2, 3, subplot_spec=gs0[0])
gs01 = gridspec.GridSpecFromSubplotSpec(3, 2, subplot_spec=gs0[1])

for a in range(2):
    for b in range(3):
        fig.add_subplot(gs00[a, b])
        fig.add_subplot(gs01[b, a])

fig.tight_layout()
```



A Complex Nested GridSpec using SubplotSpec

Here's a more sophisticated example of nested GridSpec where we put a box around each cell of the outer 4x4 grid, by hiding appropriate spines in each of the inner 3x3 grids.

```
import numpy as np
from itertools import product

def squiggle_xy(a, b, c, d, i=np.arange(0.0, 2*np.pi, 0.05)):
    return np.sin(i*a)*np.cos(i*b), np.sin(i*c)*np.cos(i*d)

fig = plt.figure(figsize=(8, 8))

# gridspec inside gridspec
outer_grid = gridspec.GridSpec(4, 4, wspace=0.0, hspace=0.0)

for i in range(16):
    inner_grid = gridspec.GridSpecFromSubplotSpec(
        3, 3, subplot_spec=outer_grid[i], wspace=0.0, hspace=0.0)
```

(continues on next page)

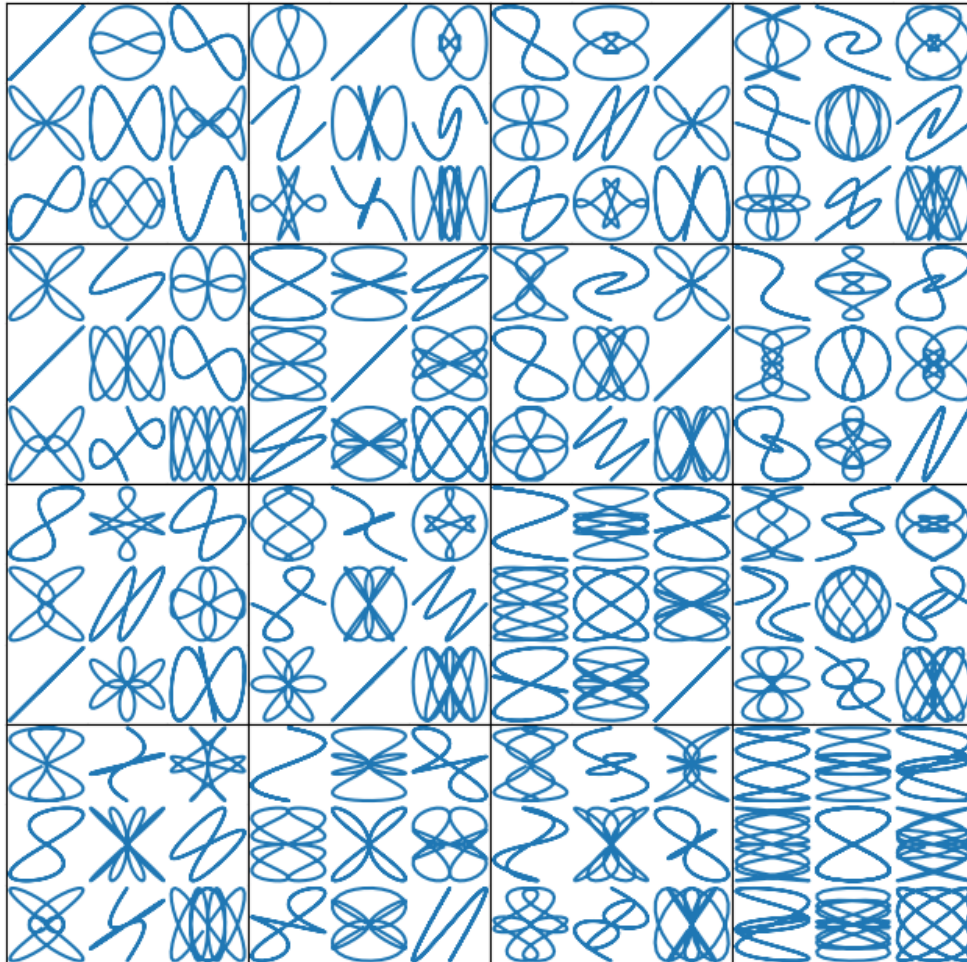
(continued from previous page)

```
a, b = int(i/4)+1, i % 4+1
for j, (c, d) in enumerate(product(range(1, 4), repeat=2)):
    ax = plt.Subplot(fig, inner_grid[j])
    ax.plot(*squiggle_xy(a, b, c, d))
    ax.set_xticks([])
    ax.set_yticks([])
    fig.add_subplot(ax)

all_axes = fig.get_axes()

# show only the outside spines
for ax in all_axes:
    for sp in ax.spines.values():
        sp.set_visible(False)
    if ax.is_first_row():
        ax.spines['top'].set_visible(True)
    if ax.is_last_row():
        ax.spines['bottom'].set_visible(True)
    if ax.is_first_col():
        ax.spines['left'].set_visible(True)
    if ax.is_last_col():
        ax.spines['right'].set_visible(True)

plt.show()
```



Total running time of the script: (0 minutes 1.859 seconds)

Note: [Click here to download the full example code](#)

3.2.5 Tight Layout guide

How to use tight-layout to fit plots within your figure cleanly.

`tight_layout` automatically adjusts subplot params so that the subplot(s) fits in to the figure area. This is an experimental feature and may not work for some cases. It only checks the extents of ticklabels, axis labels, and titles.

Simple Example

In matplotlib, the location of axes (including subplots) are specified in normalized figure coordinates. It can happen that your axis labels or titles (or sometimes even ticklabels) go outside the figure area, and are thus clipped.

```
# sphinx_gallery_thumbnail_number = 7

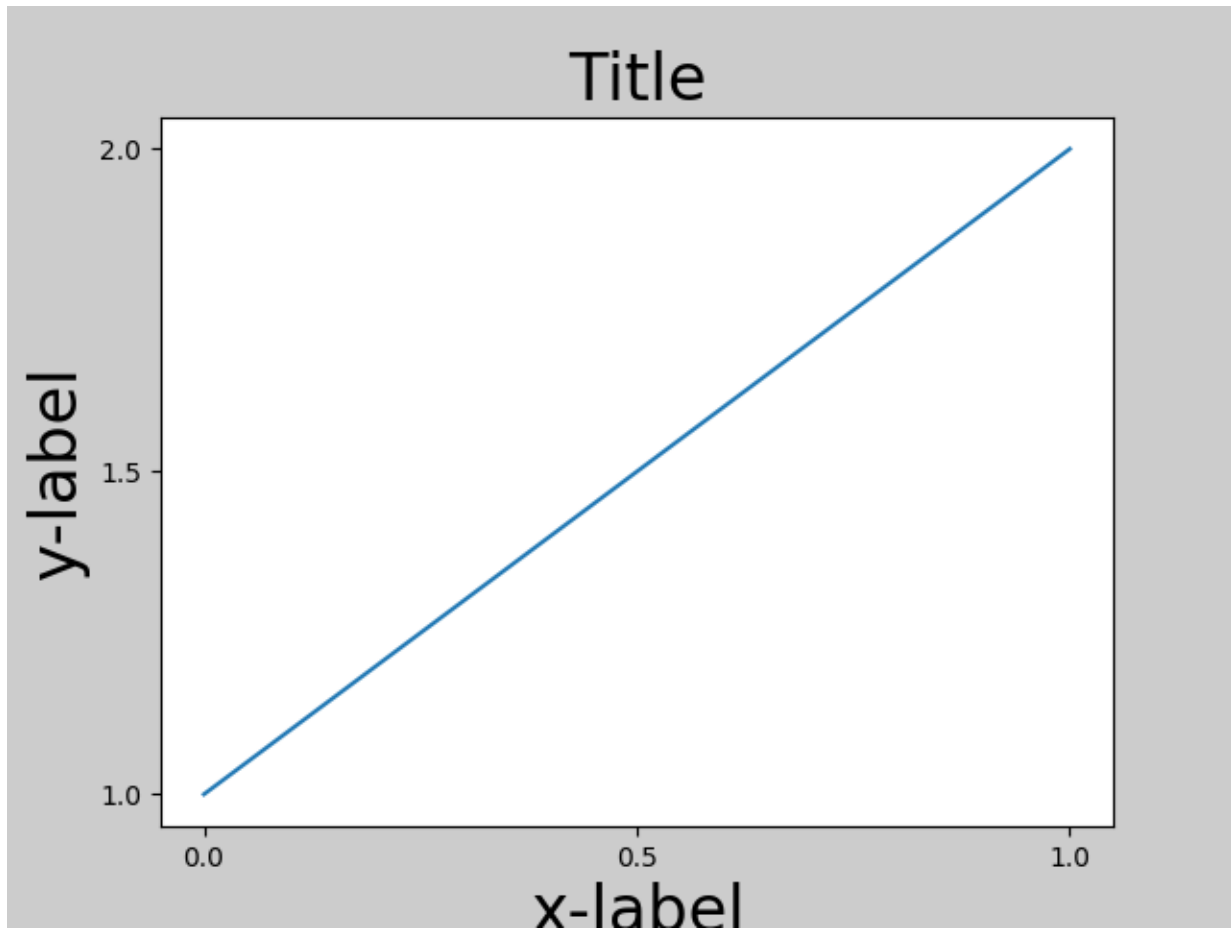
import matplotlib.pyplot as plt
import numpy as np

plt.rcParams['savefig.facecolor'] = "0.8"

def example__plot(ax, fontsize=12):
    ax.plot([1, 2])

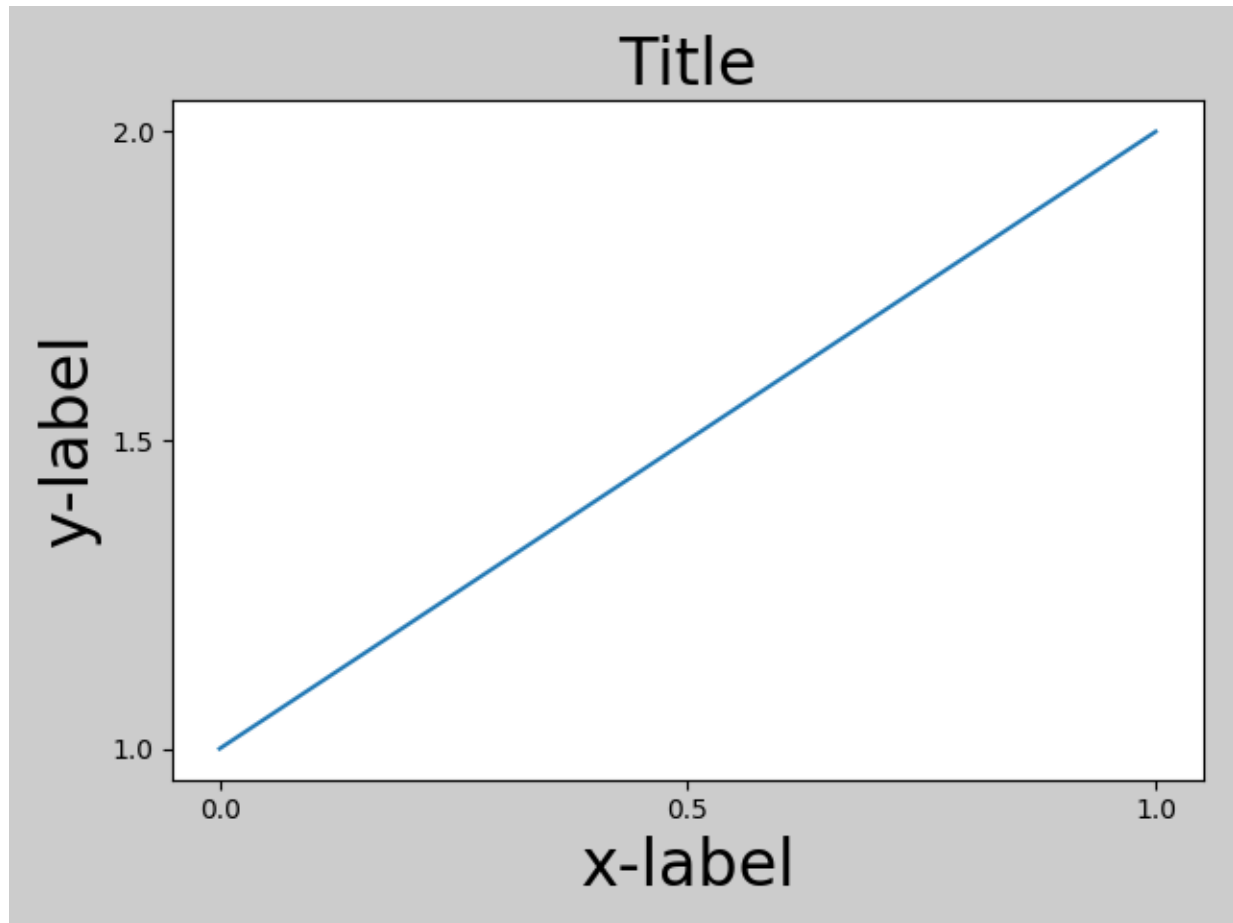
    ax.locator_params(nbins=3)
    ax.set_xlabel('x-label', fontsize=fontsize)
    ax.set_ylabel('y-label', fontsize=fontsize)
    ax.set_title('Title', fontsize=fontsize)

plt.close('all')
fig, ax = plt.subplots()
example__plot(ax, fontsize=24)
```



To prevent this, the location of axes needs to be adjusted. For subplots, this can be done by adjusting the subplot params (Move the edge of an axes to make room for tick labels). Matplotlib v1.1 introduces a new command `tight_layout()` that does this automatically for you.

```
fig, ax = plt.subplots()
example_plot(ax, fontsize=24)
plt.tight_layout()
```

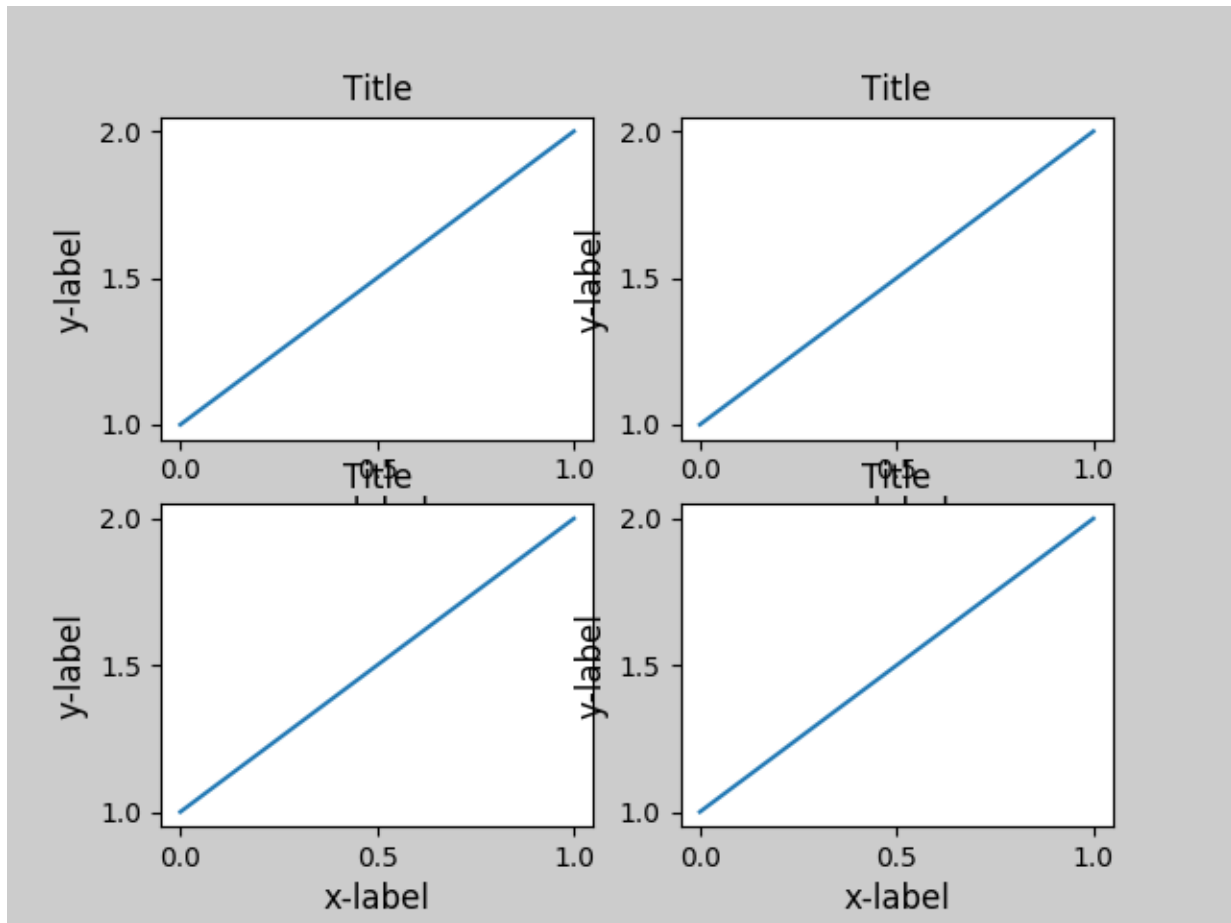


Note that `matplotlib.pyplot.tight_layout()` will only adjust the subplot params when it is called. In order to perform this adjustment each time the figure is redrawn, you can call `fig.set_tight_layout(True)`, or, equivalently, set the figure `autolayout` rcParam to `True`.

When you have multiple subplots, often you see labels of different axes overlapping each other.

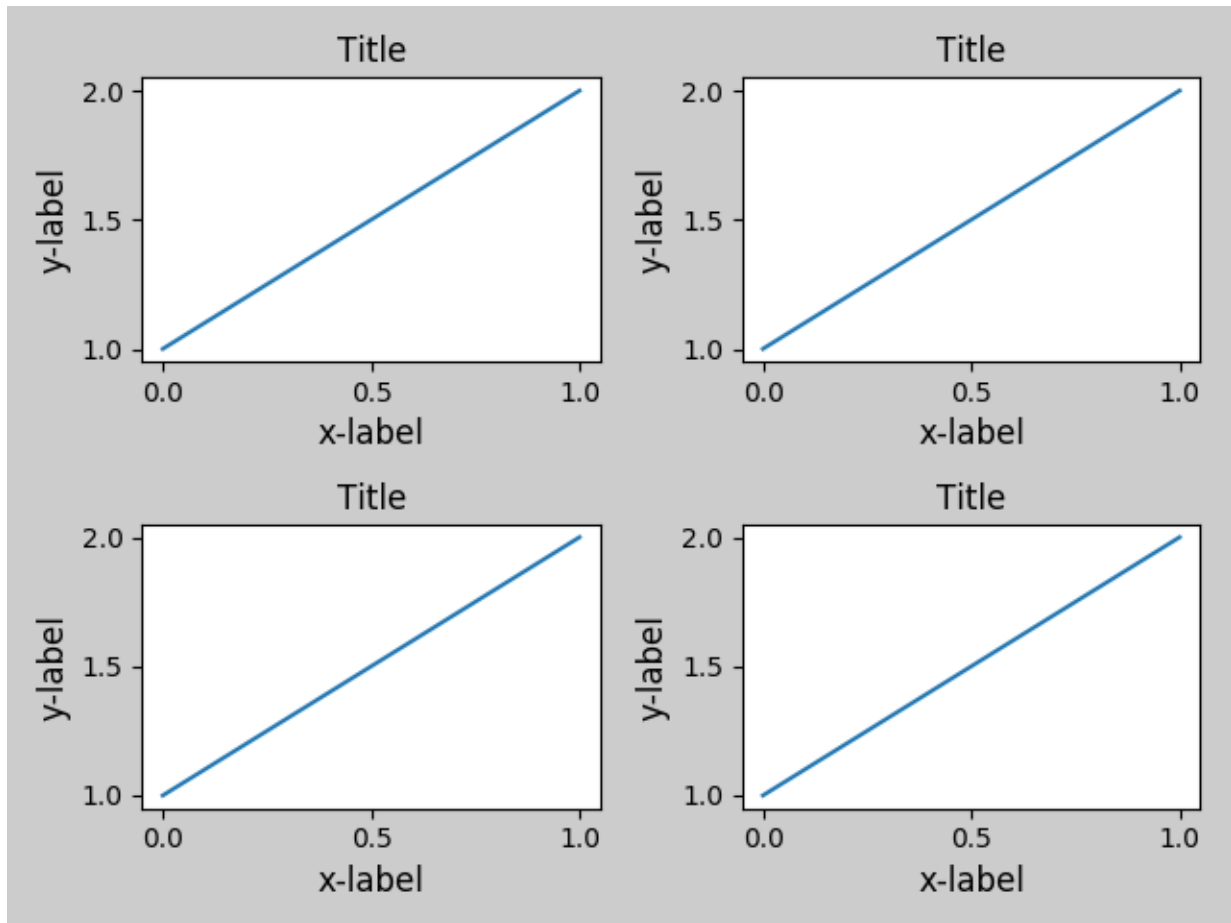
```
plt.close('all')

fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2)
example_plot(ax1)
example_plot(ax2)
example_plot(ax3)
example_plot(ax4)
```



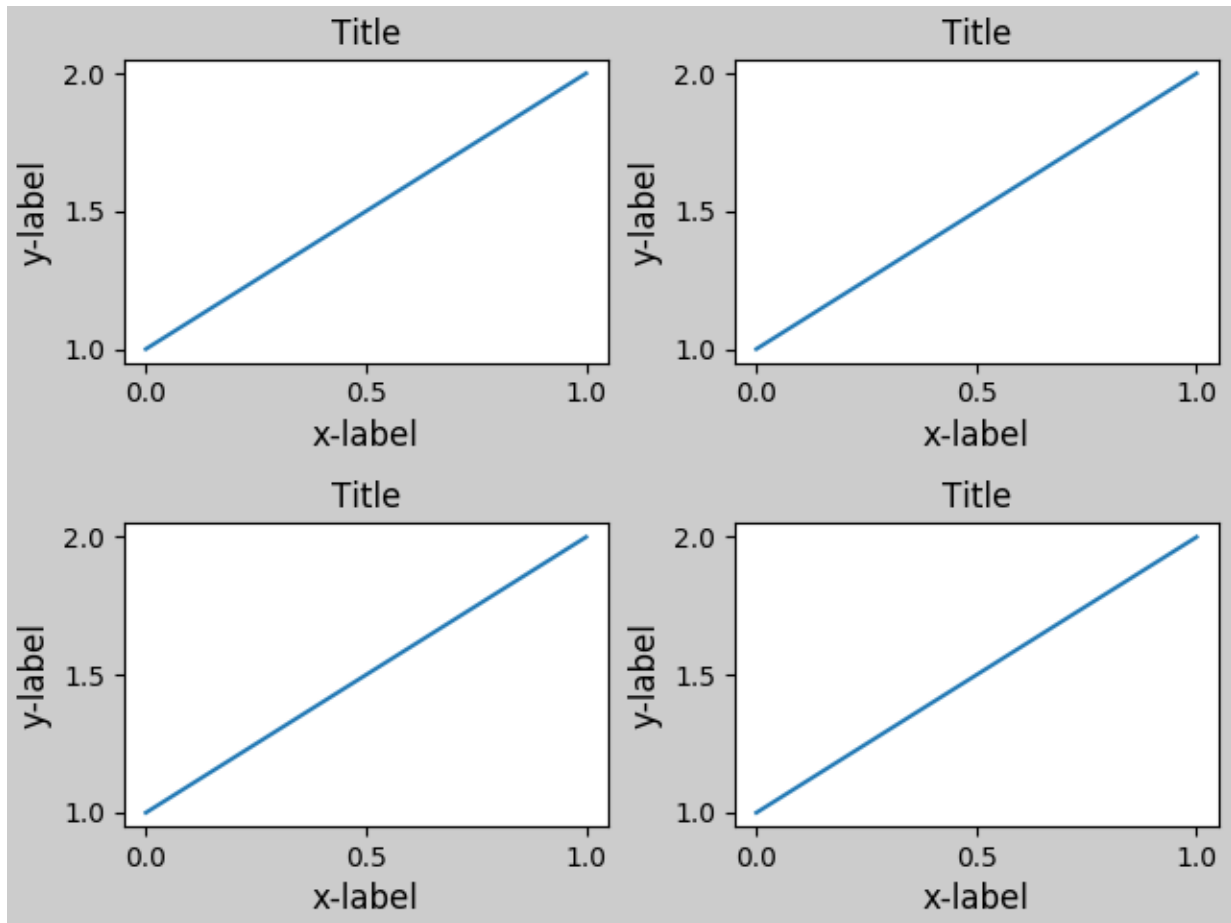
`tight_layout()` will also adjust spacing between subplots to minimize the overlaps.

```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2)
example_plot(ax1)
example_plot(ax2)
example_plot(ax3)
example_plot(ax4)
plt.tight_layout()
```



`tight_layout()` can take keyword arguments of `pad`, `w_pad` and `h_pad`. These control the extra padding around the figure border and between subplots. The pads are specified in fraction of fontsize.

```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2)
example_plot(ax1)
example_plot(ax2)
example_plot(ax3)
example_plot(ax4)
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
```



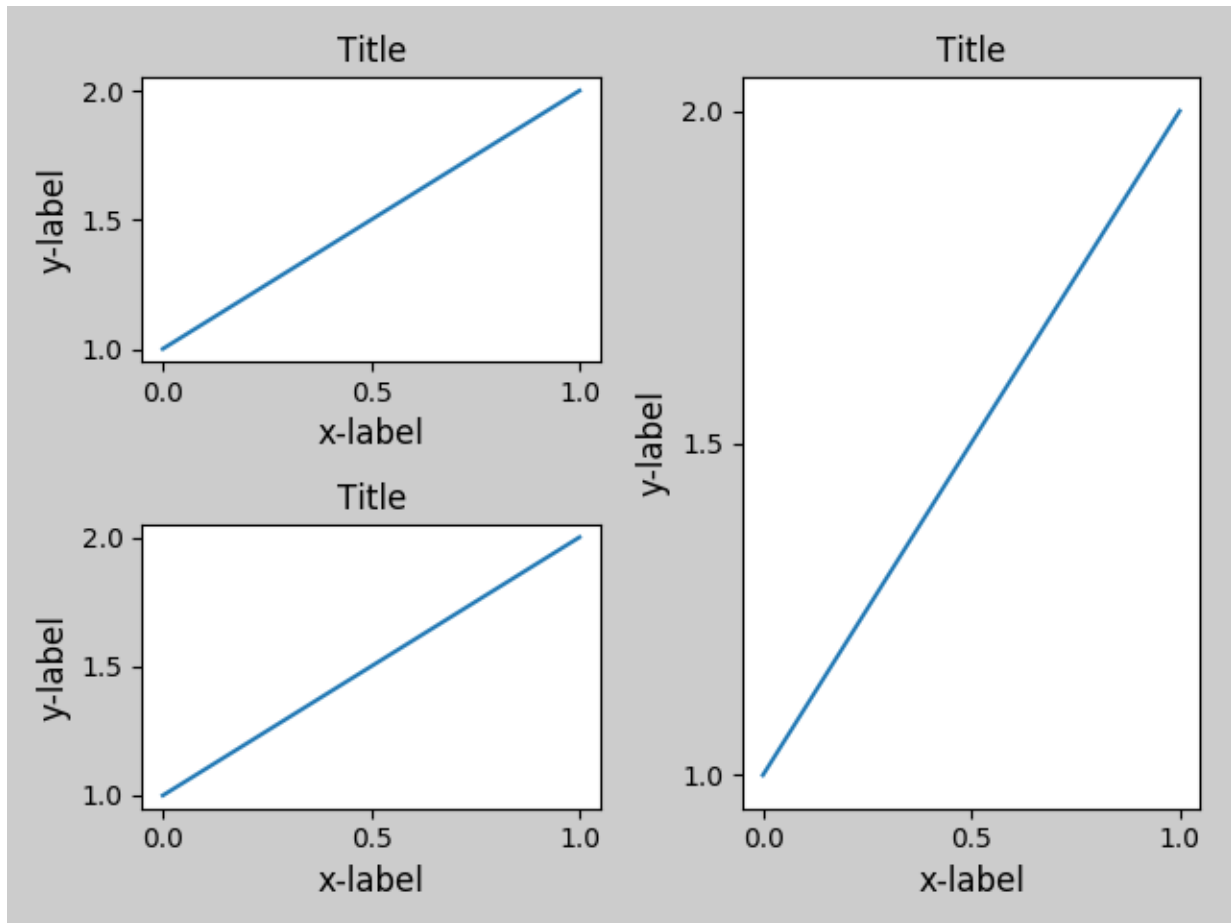
`tight_layout()` will work even if the sizes of subplots are different as far as their grid specification is compatible. In the example below, `ax1` and `ax2` are subplots of a 2x2 grid, while `ax3` is of a 1x2 grid.

```
plt.close('all')
fig = plt.figure()

ax1 = plt.subplot(221)
ax2 = plt.subplot(223)
ax3 = plt.subplot(122)

example_plot(ax1)
example_plot(ax2)
example_plot(ax3)

plt.tight_layout()
```

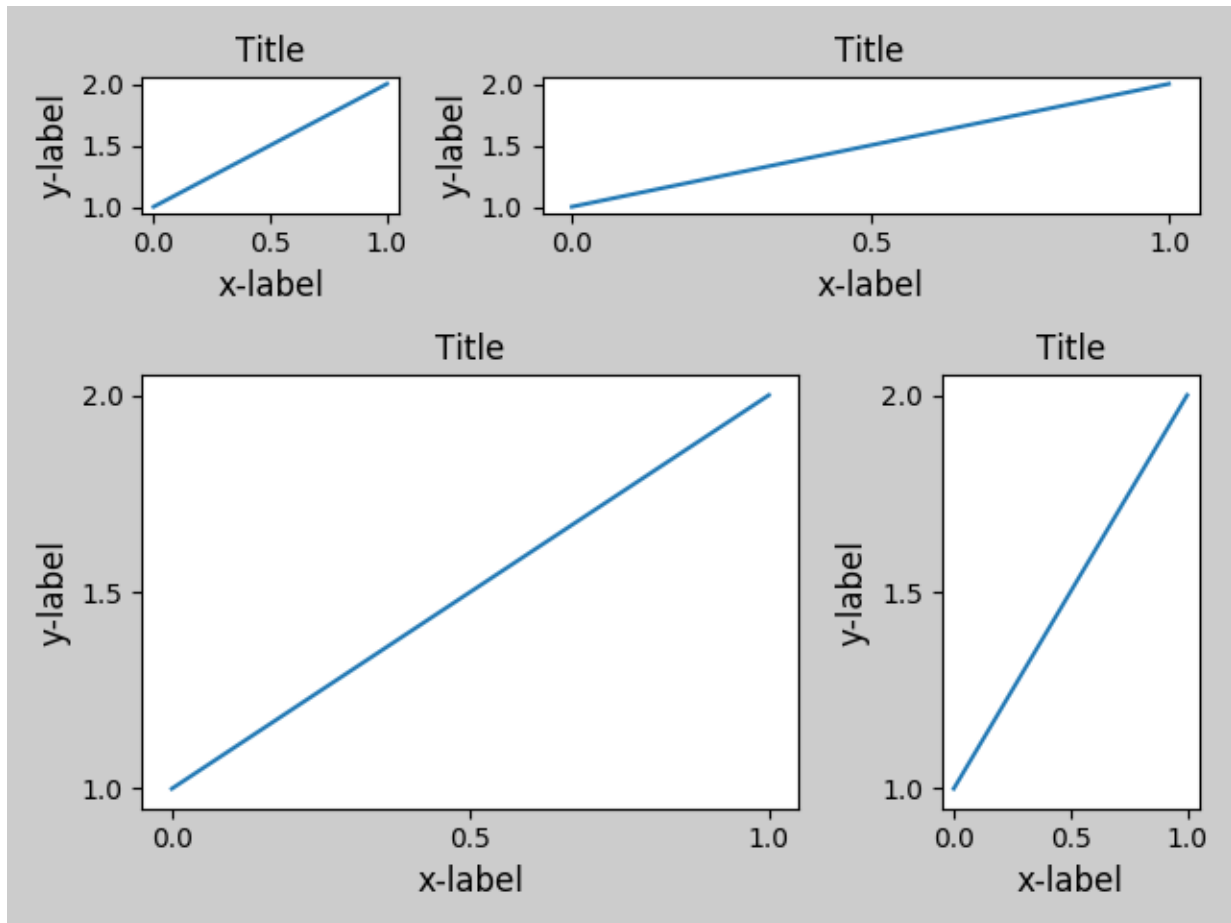
It works with subplots created with `subplot2grid()`. In general, subplots created from the `gridspec` ([Customizing Figure Layouts Using GridSpec and Other Functions](#)) will work.

```
plt.close('all')
fig = plt.figure()

ax1 = plt.subplot2grid((3, 3), (0, 0))
ax2 = plt.subplot2grid((3, 3), (0, 1), colspan=2)
ax3 = plt.subplot2grid((3, 3), (1, 0), colspan=2, rowspan=2)
ax4 = plt.subplot2grid((3, 3), (1, 2), rowspan=2)

example_plot(ax1)
example_plot(ax2)
example_plot(ax3)
example_plot(ax4)

plt.tight_layout()
```



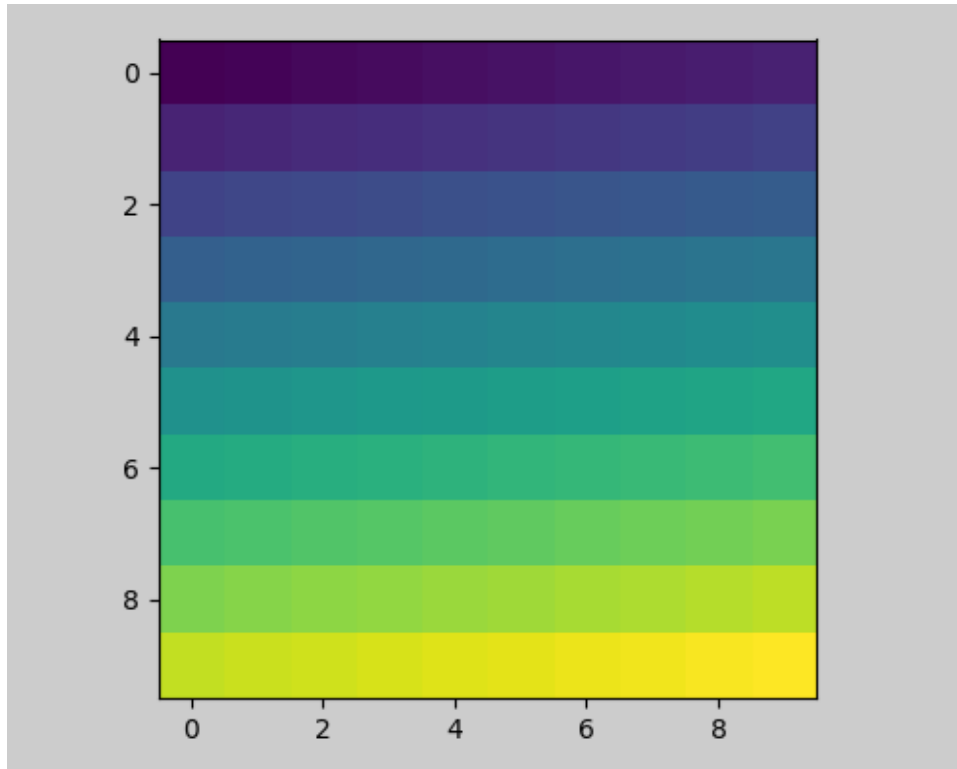
Although not thoroughly tested, it seems to work for subplots with `aspect != "auto"` (e.g., axes with images).

```
arr = np.arange(100).reshape((10, 10))

plt.close('all')
fig = plt.figure(figsize=(5, 4))

ax = plt.subplot(111)
im = ax.imshow(arr, interpolation="none")

plt.tight_layout()
```



Caveats

- `tight_layout()` only considers ticklabels, axis labels, and titles. Thus, other artists may be clipped and also may overlap.
- It assumes that the extra space needed for ticklabels, axis labels, and titles is independent of original location of axes. This is often true, but there are rare cases where it is not.
- `pad=0` clips some of the texts by a few pixels. This may be a bug or a limitation of the current algorithm and it is not clear why it happens. Meanwhile, use of `pad` at least larger than 0.3 is recommended.

Use with GridSpec

GridSpec has its own `tight_layout()` method (the pyplot api `tight_layout()` also works).

```
import matplotlib.gridspec as gridspec

plt.close('all')
fig = plt.figure()

gs1 = gridspec.GridSpec(2, 1)
ax1 = fig.add_subplot(gs1[0])
ax2 = fig.add_subplot(gs1[1])
```

(continues on next page)

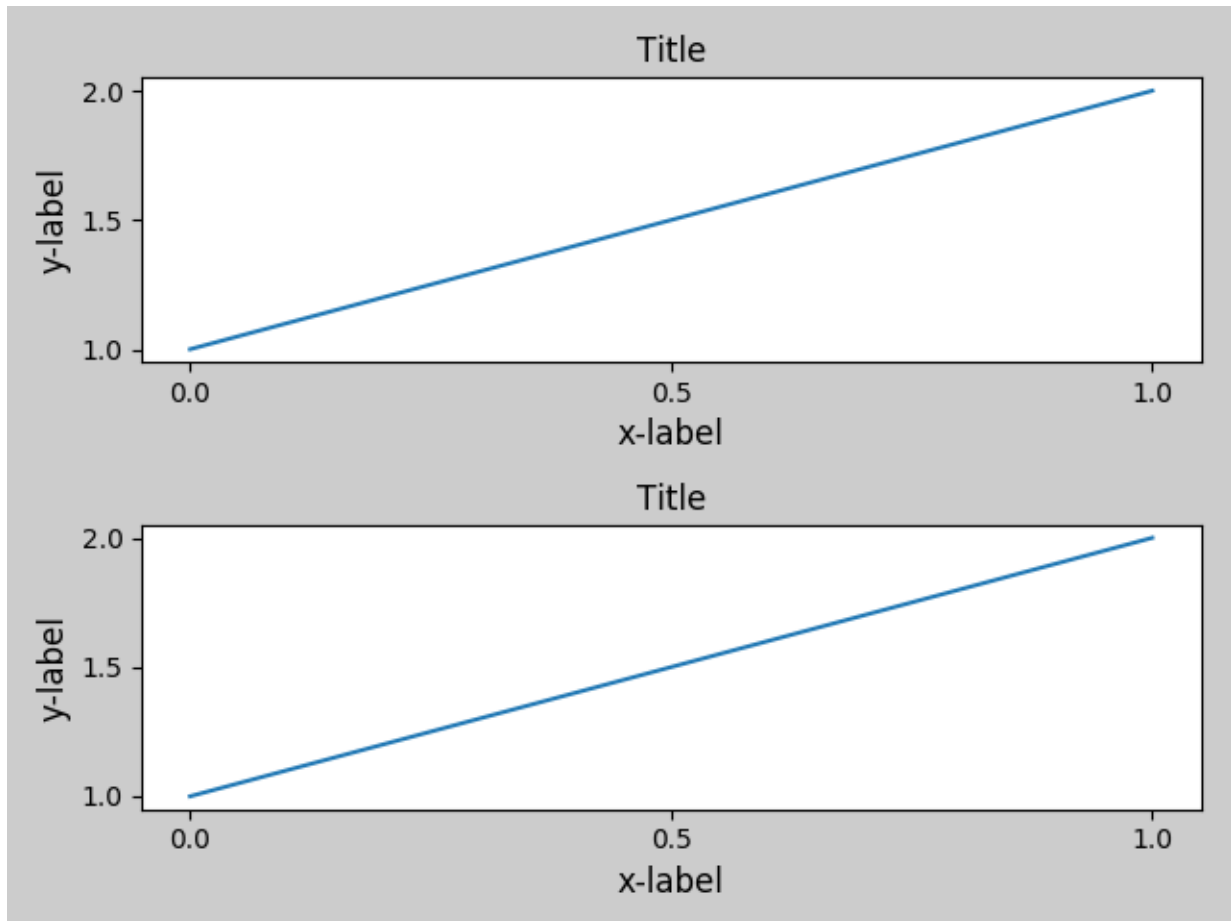
(continued from previous page)

```

example_plot(ax1)
example_plot(ax2)

gs1.tight_layout(fig)

```



You may provide an optional `rect` parameter, which specifies the bounding box that the subplots will be fit inside. The coordinates must be in normalized figure coordinates and the default is (0, 0, 1, 1).

```

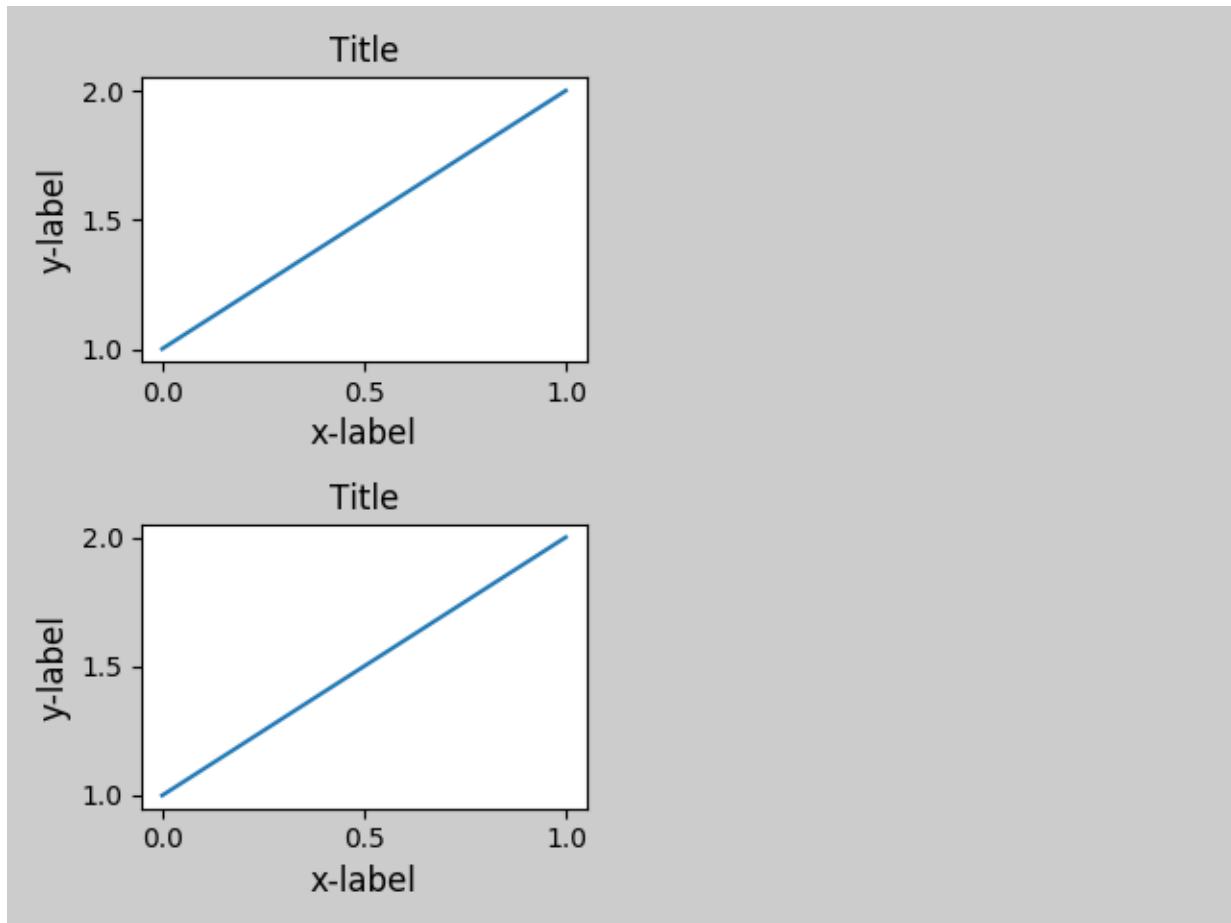
fig = plt.figure()

gs1 = gridspec.GridSpec(2, 1)
ax1 = fig.add_subplot(gs1[0])
ax2 = fig.add_subplot(gs1[1])

example_plot(ax1)
example_plot(ax2)

gs1.tight_layout(fig, rect=[0, 0, 0.5, 1])

```



For example, this can be used for a figure with multiple gridspecs.

```
fig = plt.figure()

gs1 = gridspec.GridSpec(2, 1)
ax1 = fig.add_subplot(gs1[0])
ax2 = fig.add_subplot(gs1[1])

example_plot(ax1)
example_plot(ax2)

gs1.tight_layout(fig, rect=[0, 0, 0.5, 1])

gs2 = gridspec.GridSpec(3, 1)

for ss in gs2:
    ax = fig.add_subplot(ss)
    example_plot(ax)
    ax.set_title("")
    ax.set_xlabel("")
```

(continues on next page)

(continued from previous page)

```

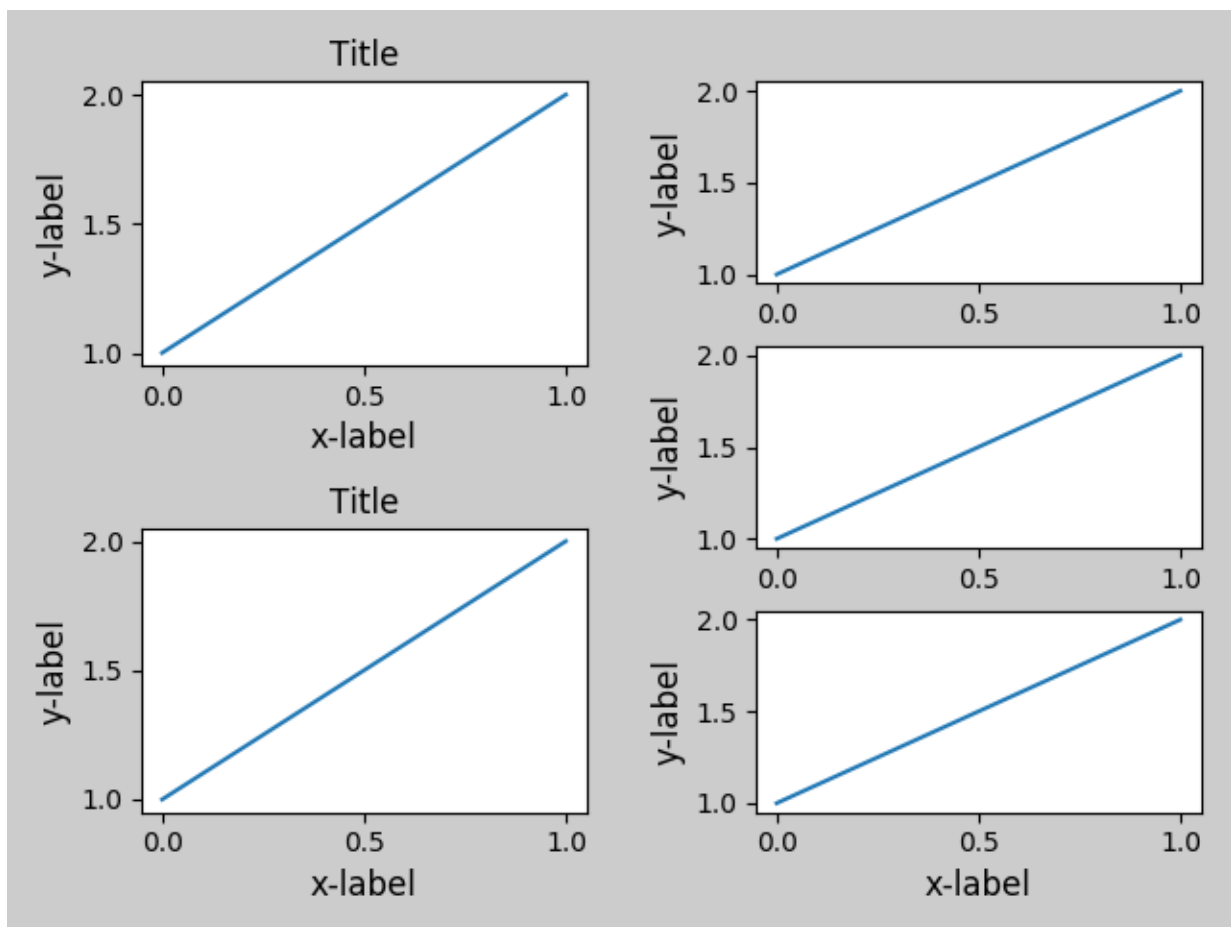
ax.set_xlabel("x-label", fontsize=12)

gs2.tight_layout(fig, rect=[0.5, 0, 1, 1], h_pad=0.5)

# We may try to match the top and bottom of two grids ::
top = min(gs1.top, gs2.top)
bottom = max(gs1.bottom, gs2.bottom)

gs1.update(top=top, bottom=bottom)
gs2.update(top=top, bottom=bottom)
plt.show()

```



While this should be mostly good enough, adjusting top and bottom may require adjustment of hspace also. To update hspace & vspace, we call `tight_layout()` again with updated `rect` argument. Note that the `rect` argument specifies the area including the ticklabels, etc. Thus, we will increase the bottom (which is 0 for the normal case) by the difference between the bottom from above and the bottom of each gridspec. Same thing for the top.

```
fig = plt.gcf()
```

(continues on next page)

(continued from previous page)

```
gs1 = gridspec.GridSpec(2, 1)
ax1 = fig.add_subplot(gs1[0])
ax2 = fig.add_subplot(gs1[1])

example_plot(ax1)
example_plot(ax2)

gs1.tight_layout(fig, rect=[0, 0, 0.5, 1])

gs2 = gridspec.GridSpec(3, 1)

for ss in gs2:
    ax = fig.add_subplot(ss)
    example_plot(ax)
    ax.set_title("")
    ax.set_xlabel("")

ax.set_xlabel("x-label", fontsize=12)

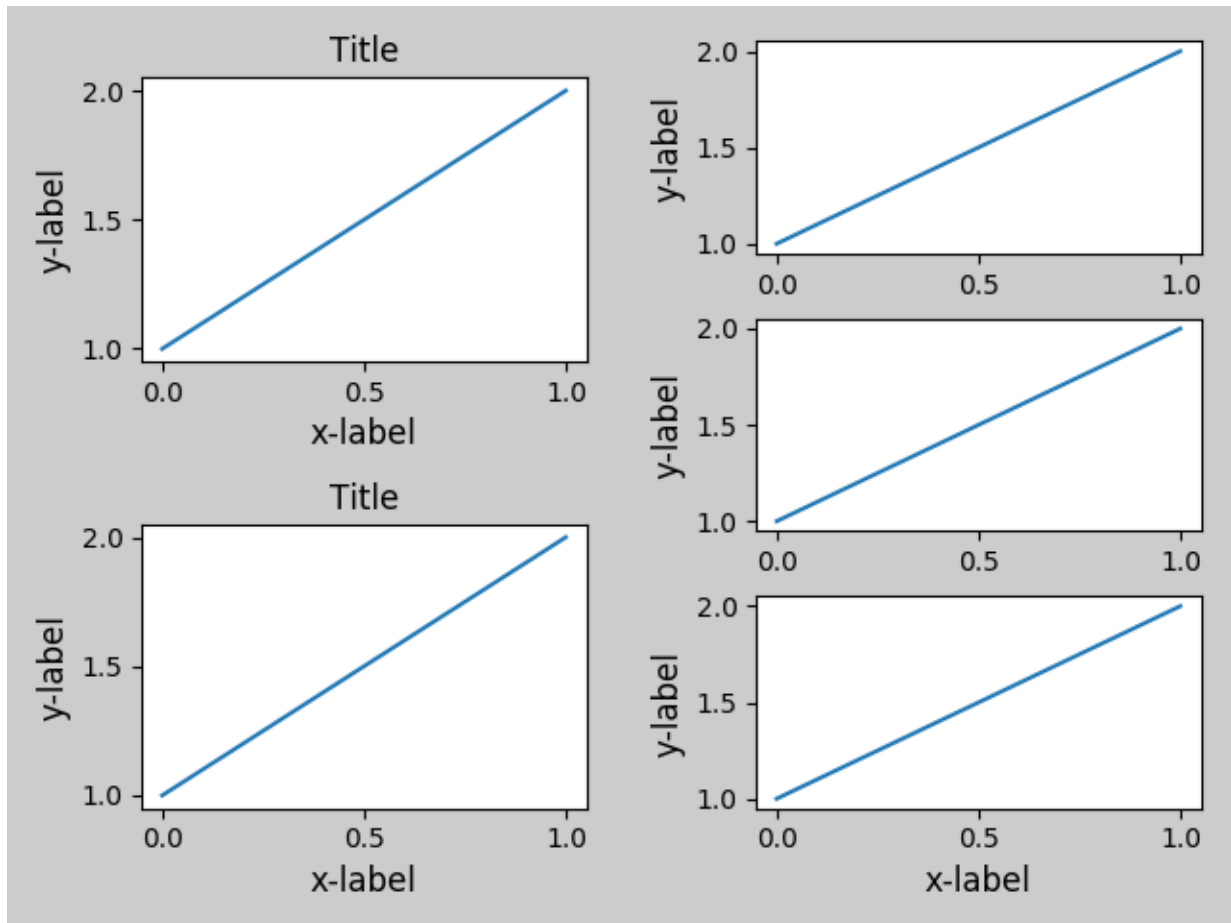
gs2.tight_layout(fig, rect=[0.5, 0, 1, 1], h_pad=0.5)

top = min(gs1.top, gs2.top)
bottom = max(gs1.bottom, gs2.bottom)

gs1.update(top=top, bottom=bottom)
gs2.update(top=top, bottom=bottom)

top = min(gs1.top, gs2.top)
bottom = max(gs1.bottom, gs2.bottom)

gs1.tight_layout(fig, rect=[None, 0 + (bottom-gs1.bottom),
                           0.5, 1 - (gs1.top-top)])
gs2.tight_layout(fig, rect=[0.5, 0 + (bottom-gs2.bottom),
                           None, 1 - (gs2.top-top)],
                  h_pad=0.5)
```



Use with AxesGrid1

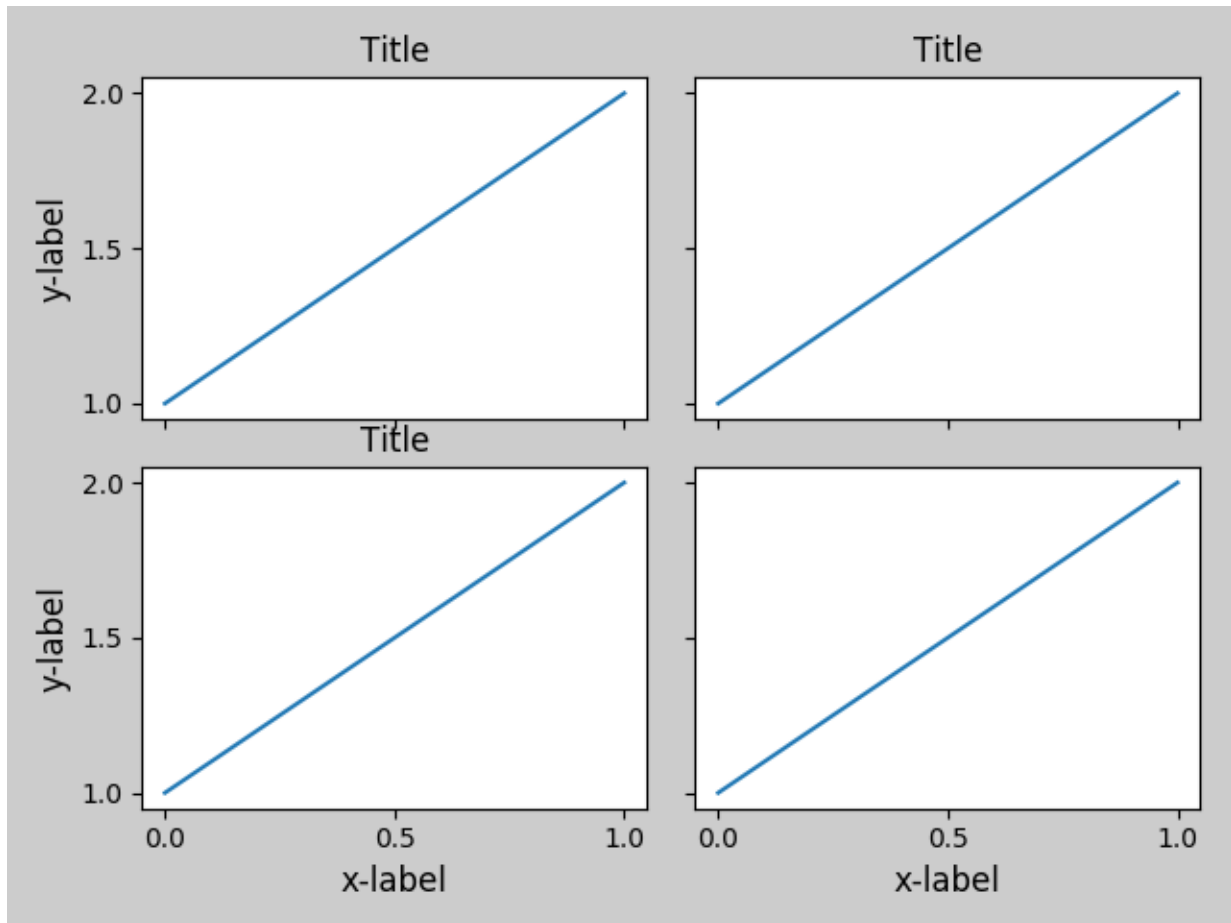
While limited, the `axes_grid1` toolkit is also supported.

```
from mpl_toolkits.axes_grid1 import Grid

plt.close('all')
fig = plt.figure()
grid = Grid(fig, rect=111, nrows_ncols=(2, 2),
            axes_pad=0.25, label_mode='L',
            )

for ax in grid:
    example_plot(ax)
ax.title.set_visible(False)

plt.tight_layout()
```

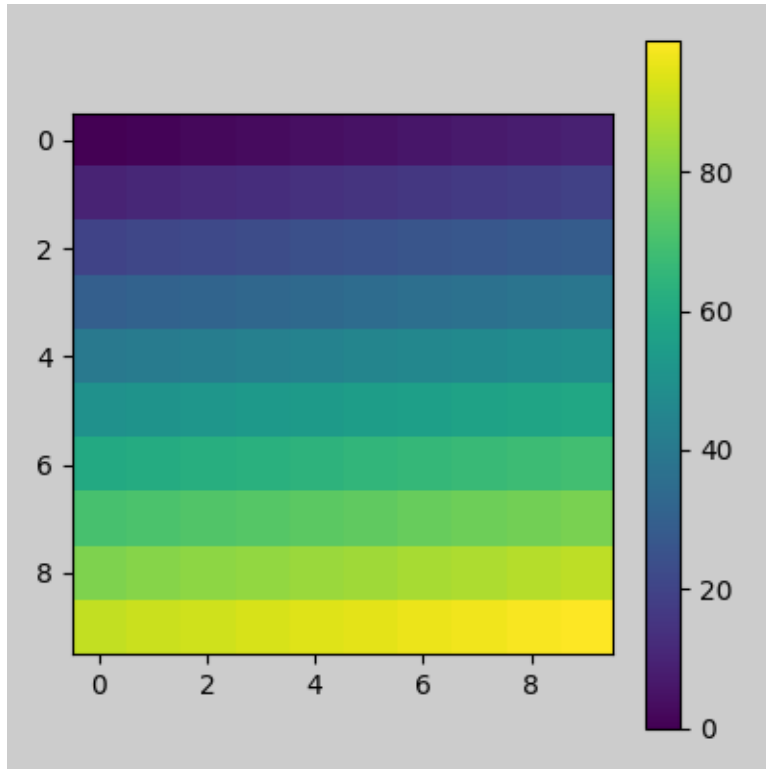
Colorbar

If you create a colorbar with the `colorbar()` command, the created colorbar is an instance of `Axes`, not `Subplot`, so `tight_layout` does not work. With Matplotlib v1.1, you may create a colorbar as a subplot using the `gridspec`.

```
plt.close('all')
arr = np.arange(100).reshape((10, 10))
fig = plt.figure(figsize=(4, 4))
im = plt.imshow(arr, interpolation="none")

plt.colorbar(im, use_gridspec=True)

plt.tight_layout()
```



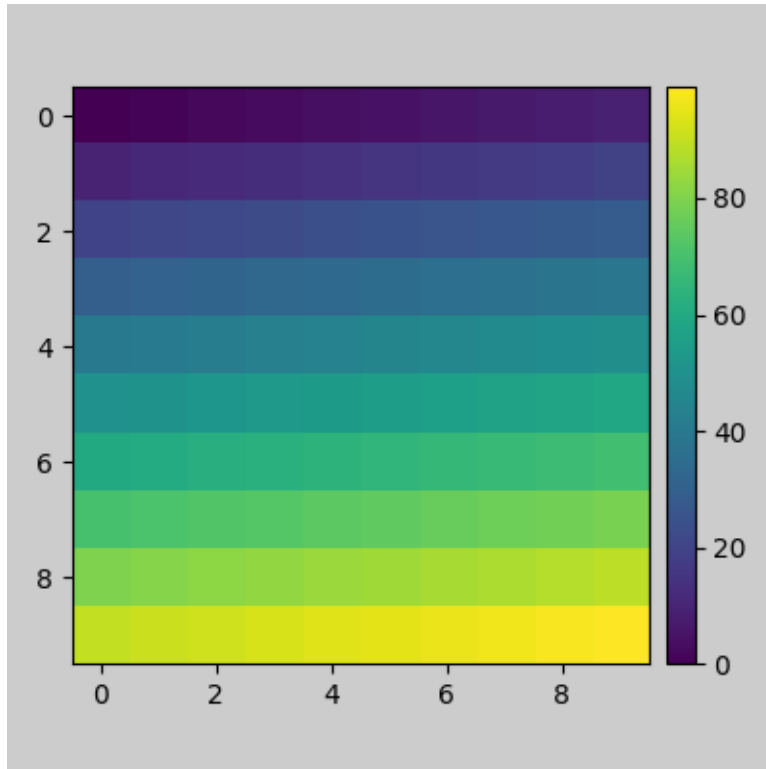
Another option is to use AxesGrid1 toolkit to explicitly create an axes for colorbar.

```
from mpl_toolkits.axes_grid1 import make_axes_locatable

plt.close('all')
arr = np.arange(100).reshape((10, 10))
fig = plt.figure(figsize=(4, 4))
im = plt.imshow(arr, interpolation="none")

divider = make_axes_locatable(plt.gca())
cax = divider.append_axes("right", "5%", pad="3%")
plt.colorbar(im, cax=cax)

plt.tight_layout()
```



Note: [Click here to download the full example code](#)

3.2.6 origin and extent in imshow

`imshow()` allows you to render an image (either a 2D array which will be color-mapped (based on `norm` and `cmap`) or and 3D RGB(A) array which will be used as-is) to a rectangular region in dataspace. The orientation of the image in the final rendering is controlled by the `origin` and `extent` kwargs (and attributes on the resulting `AxesImage` instance) and the data limits of the axes.

The `extent` kwarg controls the bounding box in data coordinates that the image will fill specified as (left, right, bottom, top) in data coordinates, the `origin` kwarg controls how the image fills that bounding box, and the orientation in the final rendered image is also affected by the axes limits.

Hint: Most of the code below is used for adding labels and informative text to the plots. The described effects of `origin` and `extent` can be seen in the plots without the need to follow all code details.

For a quick understanding, you may want to skip the code details below and directly continue with the discussion of the results.

```
import numpy as np
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```

from matplotlib.gridspec import GridSpec

def index_to_coordinate(index, extent, origin):
    """Return the pixel center of an index."""
    left, right, bottom, top = extent

    hshift = 0.5 * np.sign(right - left)
    left, right = left + hshift, right - hshift
    vshift = 0.5 * np.sign(top - bottom)
    bottom, top = bottom + vshift, top - vshift

    if origin == 'upper':
        bottom, top = top, bottom

    return {
        "[0, 0]": (left, bottom),
        "[M', 0]": (left, top),
        "[0, N']": (right, bottom),
        "[M', N']": (right, top),
    }[index]

def get_index_label_pos(index, extent, origin, inverted_xindex):
    """
    Return the desired position and horizontal alignment of an index label.
    """
    if extent is None:
        extent = lookup_extent(origin)
    left, right, bottom, top = extent
    x, y = index_to_coordinate(index, extent, origin)

    is_x0 = index[-2:] == "0]"
    halign = 'left' if is_x0 ^ inverted_xindex else 'right'
    hshift = 0.5 * np.sign(left - right)
    x += hshift * (1 if is_x0 else -1)
    return x, y, halign

def get_color(index, data, cmap):
    """Return the data color of an index."""
    val = {
        "[0, 0]": data[0, 0],
        "[0, N']": data[0, -1],
        "[M', 0]": data[-1, 0],
    }

```

(continues on next page)

(continued from previous page)

```

    "[M', N']": data[-1, -1],
    }[index]
    return cmap(val / data.max())

def lookup_extent(origin):
    """Return extent for label positioning when not given explicitly."""
    if origin == 'lower':
        return (-0.5, 6.5, -0.5, 5.5)
    else:
        return (-0.5, 6.5, 5.5, -0.5)

def set_extent_None_text(ax):
    ax.text(3, 2.5, 'equals\nextent=None', size='large',
           ha='center', va='center', color='w')

def plot_imshow_with_labels(ax, data, extent, origin, xlim, ylim):
    """Actually run ``imshow()`` and add extent and index labels."""
    im = ax.imshow(data, origin=origin, extent=extent)

    # extent labels (left, right, bottom, top)
    left, right, bottom, top = im.get_extent()
    if xlim is None or top > bottom:
        upper_string, lower_string = 'top', 'bottom'
    else:
        upper_string, lower_string = 'bottom', 'top'
    if ylim is None or left < right:
        port_string, starboard_string = 'left', 'right'
        inverted_xindex = False
    else:
        port_string, starboard_string = 'right', 'left'
        inverted_xindex = True
    bbox_kwargs = {'fc': 'w', 'alpha': .75, 'boxstyle': "round4"}
    ann_kwargs = {'xycoords': 'axes fraction',
                  'textcoords': 'offset points',
                  'bbox': bbox_kwargs}
    ax.annotate(upper_string, xy=(.5, 1), xytext=(0, -1),
                ha='center', va='top', **ann_kwargs)
    ax.annotate(lower_string, xy=(.5, 0), xytext=(0, 1),
                ha='center', va='bottom', **ann_kwargs)
    ax.annotate(port_string, xy=(0, .5), xytext=(1, 0),
                ha='left', va='center', rotation=90,
                **ann_kwargs)

```

(continues on next page)

(continued from previous page)

```

ax.annotate(starboard_string, xy=(1, .5), xytext=(-1, 0),
            ha='right', va='center', rotation=-90,
            **ann_kwargs)
ax.set_title('origin: {origin}'.format(origin=origin))

# index labels
for index in ["[0, 0]", "[0, N]", "[M, 0]", "[M, N]"]:
    tx, ty, halign = get_index_label_pos(index, extent, origin,
                                         inverted_xindex)
    facecolor = get_color(index, data, im.get_cmap())
    ax.text(tx, ty, index, color='white', ha=halign, va='center',
            bbox={'boxstyle': 'square', 'facecolor': facecolor})
if xlim:
    ax.set_xlim(*xlim)
if ylim:
    ax.set_ylim(*ylim)

def generate_imshow_demo_grid(extents, xlim=None, ylim=None):
    N = len(extents)
    fig = plt.figure(tight_layout=True)
    fig.set_size_inches(6, N * (11.25) / 5)
    gs = GridSpec(N, 5, figure=fig)

    columns = {'label': [fig.add_subplot(gs[j, 0]) for j in range(N)],
               'upper': [fig.add_subplot(gs[j, 1:3]) for j in range(N)],
               'lower': [fig.add_subplot(gs[j, 3:5]) for j in range(N)]}
    x, y = np.ogrid[0:6, 0:7]
    data = x + y

    for origin in ['upper', 'lower']:
        for ax, extent in zip(columns[origin], extents):
            plot_imshow_with_labels(ax, data, extent, origin, xlim, ylim)

    for ax, extent in zip(columns['label'], extents):
        text_kwargs = {'ha': 'right',
                       'va': 'center',
                       'xycoords': 'axes fraction',
                       'xy': (1, .5)}
        if extent is None:
            ax.annotate('None', **text_kwargs)
            ax.set_title('extent=')
        else:
            left, right, bottom, top = extent
            text = ('left: {left:0.1f}\nright: {right:0.1f}\n' +

```

(continues on next page)

(continued from previous page)

```

'bottom: {bottom:0.1f}\ntop: {top:0.1f}\n').format(
    left=left, right=right, bottom=bottom, top=top)

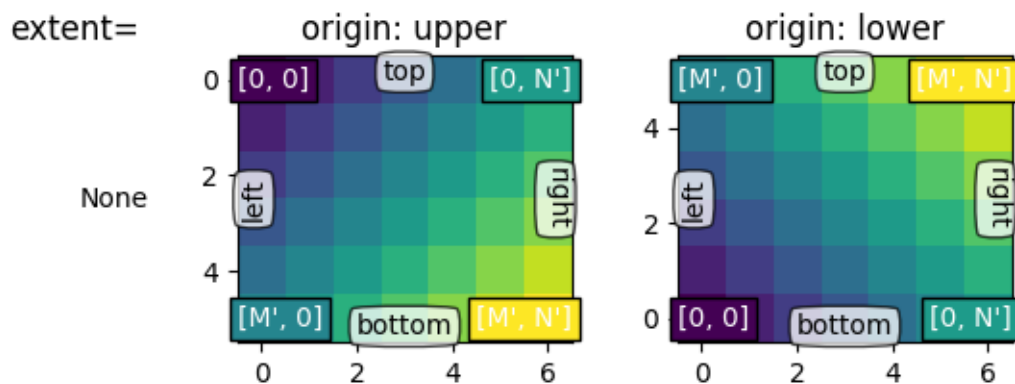
    ax.annotate(text, **text_kwargs)
    ax.axis('off')
    return columns

```

Default extent

First, let's have a look at the default `extent=None`

```
generate_imshow_demo_grid(extents=[None])
```



Generally, for an array of shape (M, N) , the first index runs along the vertical, the second index runs along the horizontal. The pixel centers are at integer positions ranging from 0 to $N' = N - 1$ horizontally and from 0 to $M' = M - 1$ vertically. `origin` determines how the data is filled in the bounding box.

For `origin='lower'`:

- $[0, 0]$ is at (left, bottom)
- $[M', 0]$ is at (left, top)
- $[0, N']$ is at (right, bottom)
- $[M', N']$ is at (right, top)

`origin='upper'` reverses the vertical axes direction and filling:

- $[0, 0]$ is at (left, top)
- $[M', 0]$ is at (left, bottom)
- $[0, N']$ is at (right, top)
- $[M', N']$ is at (right, bottom)

In summary, the position of the $[0, 0]$ index as well as the extent are influenced by origin:

origin	$[0, 0]$ position	extent
upper	top left	$(-0.5, \text{numcols}-0.5, \text{numrows}-0.5, -0.5)$
lower	bottom left	$(-0.5, \text{numcols}-0.5, -0.5, \text{numrows}-0.5)$

The default value of origin is set by `rcParams["image.origin"]` which defaults to 'upper' to match the matrix indexing conventions in math and computer graphics image indexing conventions.

Explicit extent

By setting extent we define the coordinates of the image area. The underlying image data is interpolated/resampled to fill that area.

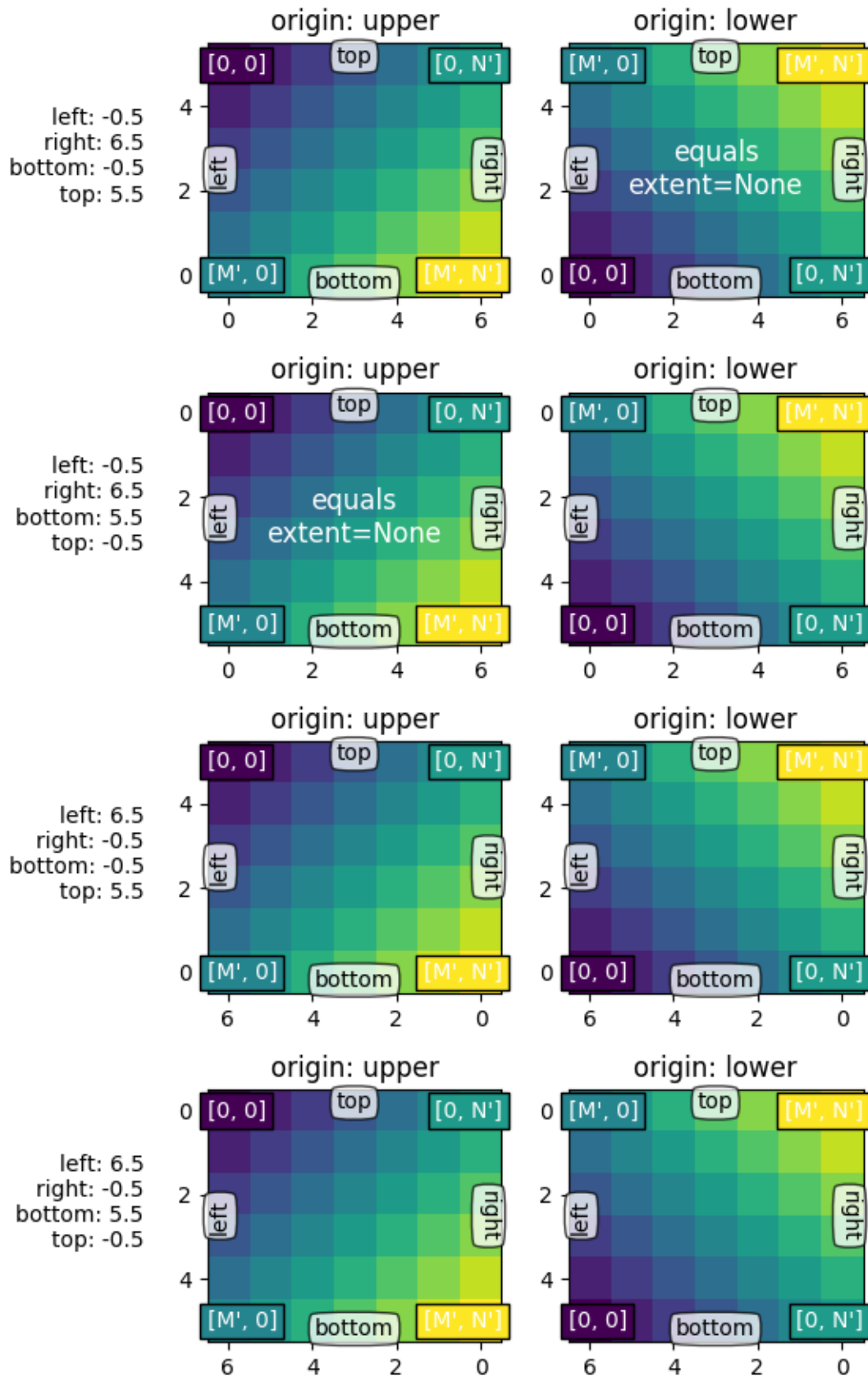
If the axes is set to autoscale, then the view limits of the axes are set to match the extent which ensures that the coordinate set by (left, bottom) is at the bottom left of the axes! However, this may invert the axis so they do not increase in the 'natural' direction.

```

extents = [(-0.5, 6.5, -0.5, 5.5),
           (-0.5, 6.5, 5.5, -0.5),
           (6.5, -0.5, -0.5, 5.5),
           (6.5, -0.5, 5.5, -0.5)]

columns = generate_imshow_demo_grid(extents)
set_extent_None_text(columns['upper'][1])
set_extent_None_text(columns['lower'][0])

```

Explicit extent and axes limits

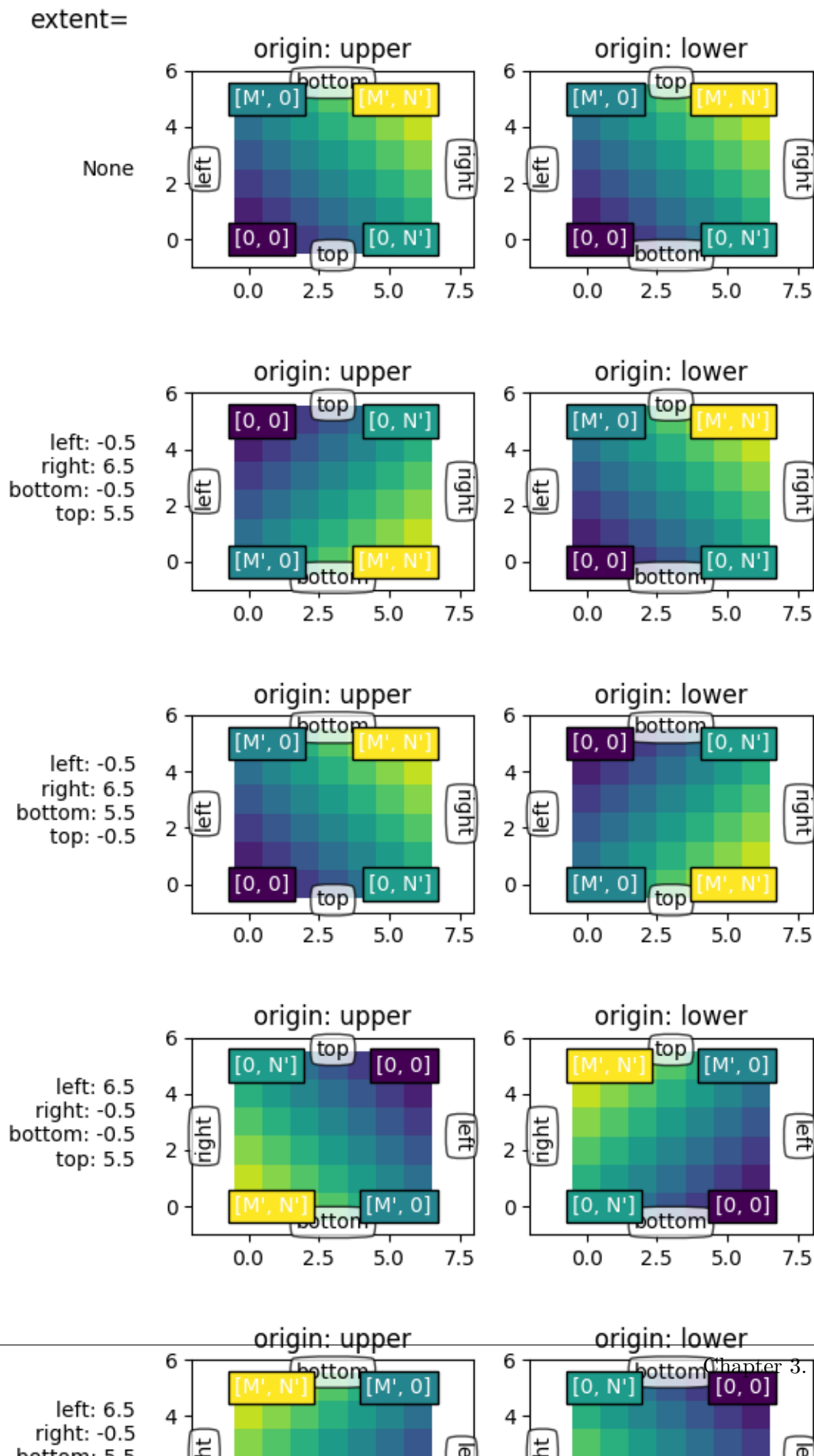
If we fix the axes limits by explicitly setting `set_xlim` / `set_ylim`, we force a certain size and orientation of the axes. This can decouple the 'left-right' and 'top-bottom' sense of the image from the orientation on the screen.

In the example below we have chosen the limits slightly larger than the extent (note the white areas within the Axes).

While we keep the extents as in the examples before, the coordinate (0, 0) is now explicitly put at the bottom left and values increase to up and to the right (from the viewer point of view). We can see that:

- The coordinate (left, bottom) anchors the image which then fills the box going towards the (right, top) point in data space.
- The first column is always closest to the 'left'.
- origin controls if the first row is closest to 'top' or 'bottom'.
- The image may be inverted along either direction.
- The 'left-right' and 'top-bottom' sense of the image may be uncoupled from the orientation on the screen.

```
generate_imshow_demo_grid(extents=[None] + extents,  
                           xlim=(-2, 8), ylim=(-1, 6))
```



Note: [Click here to download the full example code](#)

3.2.7 Constrained Layout Guide

How to use constrained-layout to fit plots within your figure cleanly.

`constrained_layout` automatically adjusts subplots and decorations like legends and colorbars so that they fit in the figure window while still preserving, as best they can, the logical layout requested by the user.

`constrained_layout` is similar to `tight_layout`, but uses a constraint solver to determine the size of axes that allows them to fit.

Warning: As of Matplotlib 2.2, Constrained Layout is experimental. The behaviour and API are subject to change, or the whole functionality may be removed without a deprecation period. If you require your plots to be absolutely reproducible, get the Axes positions after running Constrained Layout and use `ax.set_position()` in your code with `constrained_layout=False`.

Simple Example

In Matplotlib, the location of axes (including subplots) are specified in normalized figure coordinates. It can happen that your axis labels or titles (or sometimes even ticklabels) go outside the figure area, and are thus clipped.

```
# sphinx_gallery_thumbnail_number = 18

#import matplotlib
#matplotlib.use('Qt5Agg')

import warnings

import matplotlib.pyplot as plt
import numpy as np
import matplotlib.colors as mcolors
import matplotlib.gridspec as gridspec

import matplotlib._layoutbox as layoutbox

plt.rcParams['savefig.facecolor'] = "0.8"
plt.rcParams['figure.figsize'] = 4.5, 4.

def example_plot(ax, fontsize=12, nodec=False):
```

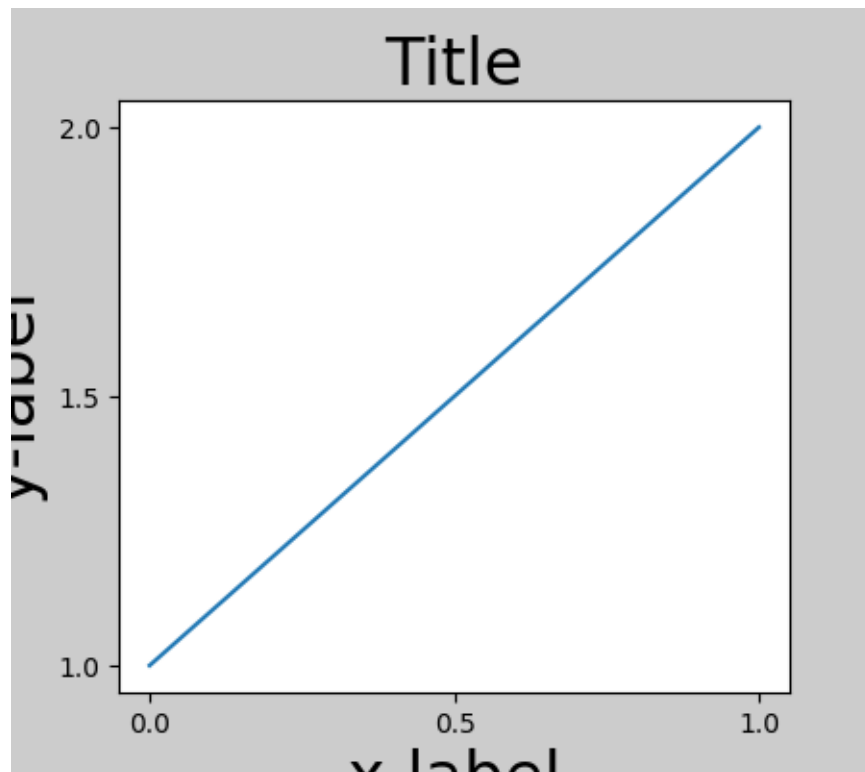
(continues on next page)

(continued from previous page)

```
ax.plot([1, 2])

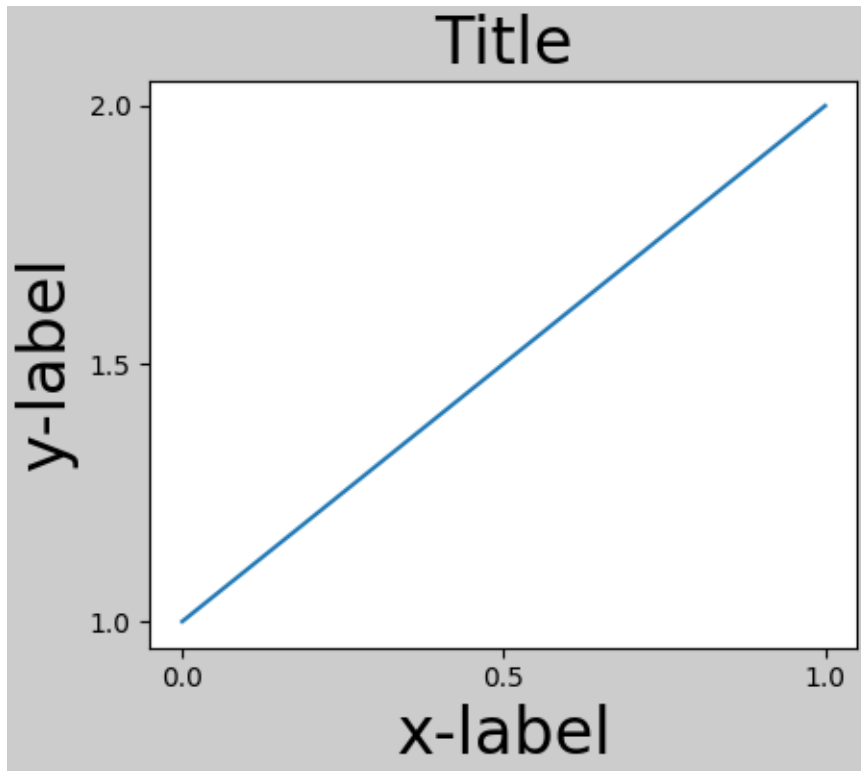
ax.locator_params(nbins=3)
if not nodec:
    ax.set_xlabel('x-label', fontsize=fontsize)
    ax.set_ylabel('y-label', fontsize=fontsize)
    ax.set_title('Title', fontsize=fontsize)
else:
    ax.set_xticklabels('')
    ax.set_yticklabels('')

fig, ax = plt.subplots()
example_plot(ax, fontsize=24)
```



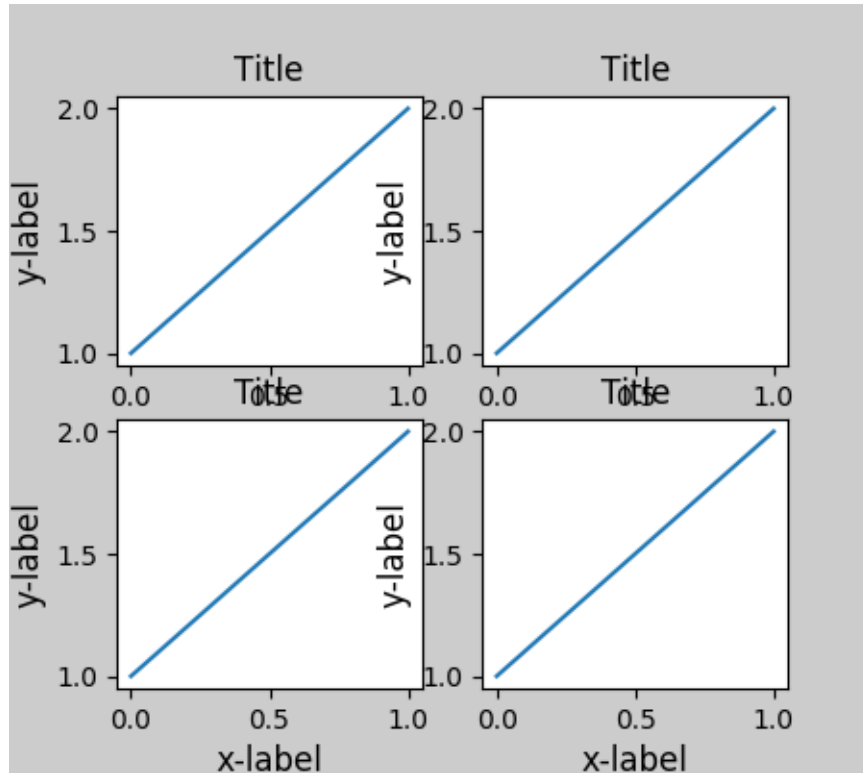
To prevent this, the location of axes needs to be adjusted. For subplots, this can be done by adjusting the subplot params (Move the edge of an axes to make room for tick labels). However, specifying your figure with the `constrained_layout=True` kwarg will do the adjusting automatically.

```
fig, ax = plt.subplots(constrained_layout=True)
example_plot(ax, fontsize=24)
```



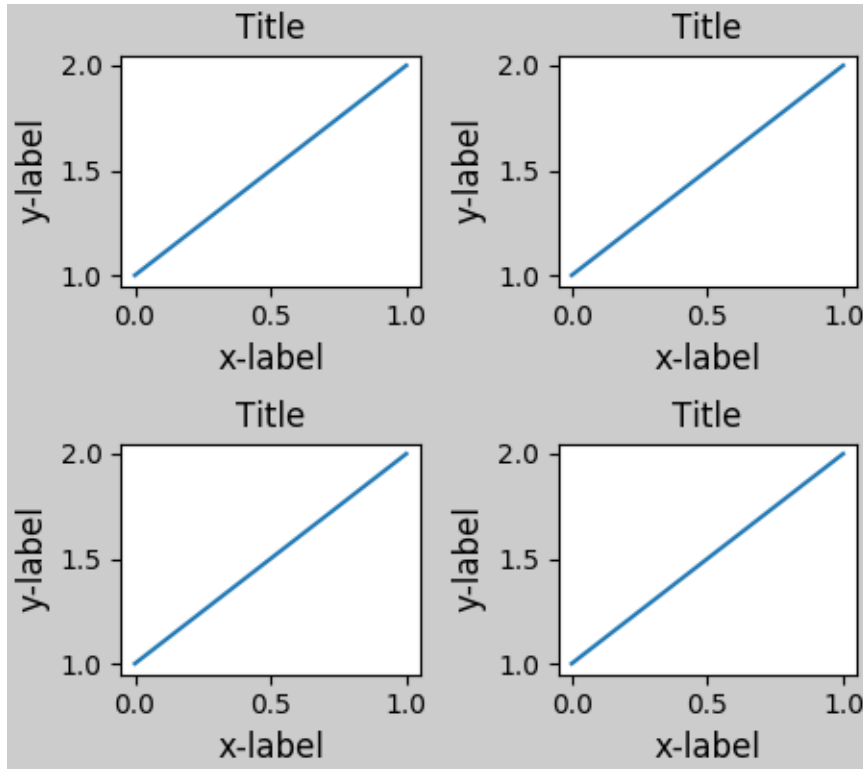
When you have multiple subplots, often you see labels of different axes overlapping each other.

```
fig, axs = plt.subplots(2, 2, constrained_layout=False)
for ax in axs.flatten():
    example_plot(ax)
```



Specifying `constrained_layout=True` in the call to `plt.subplots` causes the layout to be properly constrained.

```
fig, axs = plt.subplots(2, 2, constrained_layout=True)
for ax in axs.flatten():
    example_plot(ax)
```

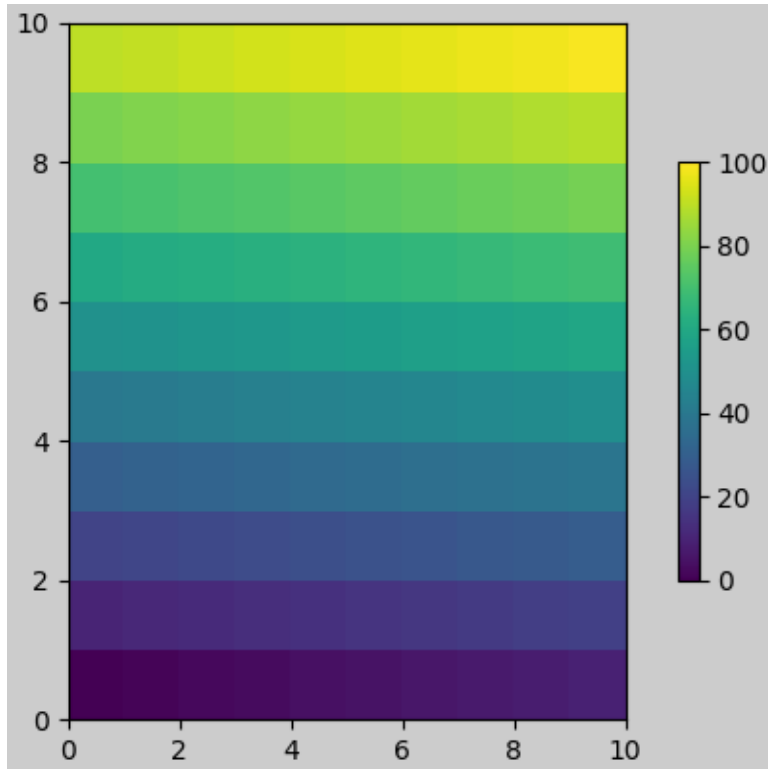


Colorbars

If you create a colorbar with the `colorbar()` command you need to make room for it. `constrained_layout` does this automatically. Note that if you specify `use_gridspec=True` it will be ignored because this option is made for improving the layout via `tight_layout`.

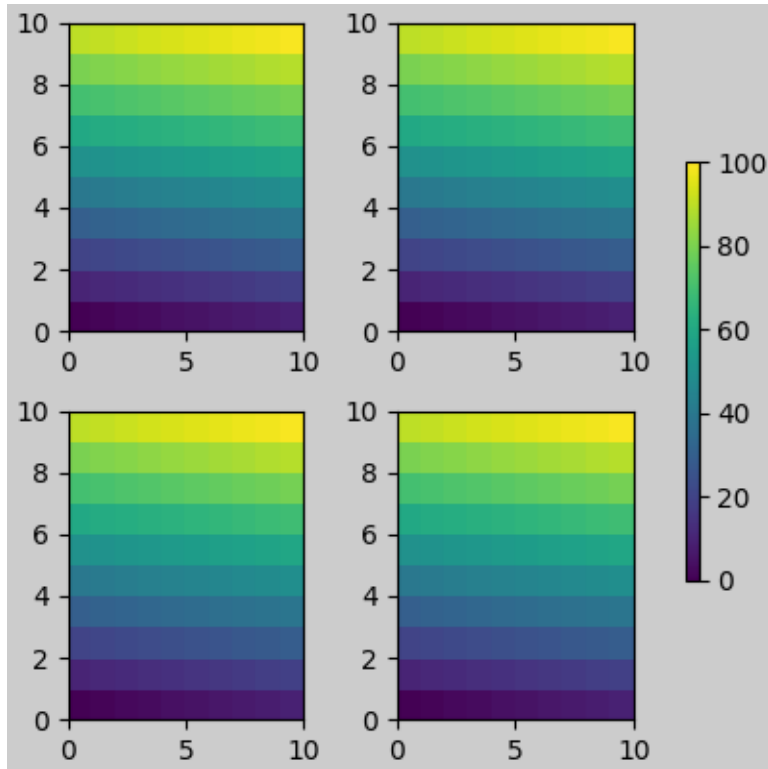
Note: For the `pcolormesh` kwargs (`pc_kwargs`) we use a dictionary. Below we will assign one colorbar to a number of axes each containing a `ScalarMappable`; specifying the `norm` and `colormap` ensures the colorbar is accurate for all the axes.

```
arr = np.arange(100).reshape((10, 10))
norm = mcolors.Normalize(vmin=0., vmax=100.)
# see note above: this makes all pcolormesh calls consistent:
pc_kwargs = {'rasterized':True, 'cmap':'viridis', 'norm':norm}
fig, ax = plt.subplots(figsize=(4, 4), constrained_layout=True)
im = ax.pcolormesh(arr, **pc_kwargs)
fig.colorbar(im, ax=ax, shrink=0.6)
```

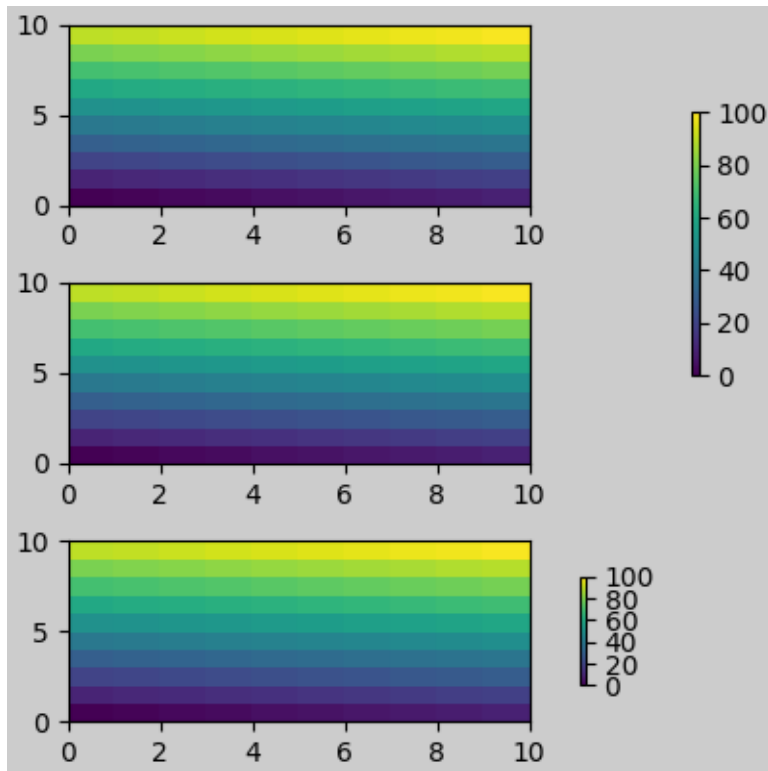
If you specify a list of axes (or other iterable container) to the `ax` argument of `colorbar`, `constrained_layout` will take space from all # axes that share the same `gridspec`.

```
fig, axs = plt.subplots(2, 2, figsize=(4, 4), constrained_layout=True)
for ax in axs.flatten():
    im = ax.pcolormesh(arr, **pc_kwargs)
fig.colorbar(im, ax=axs, shrink=0.6)
```



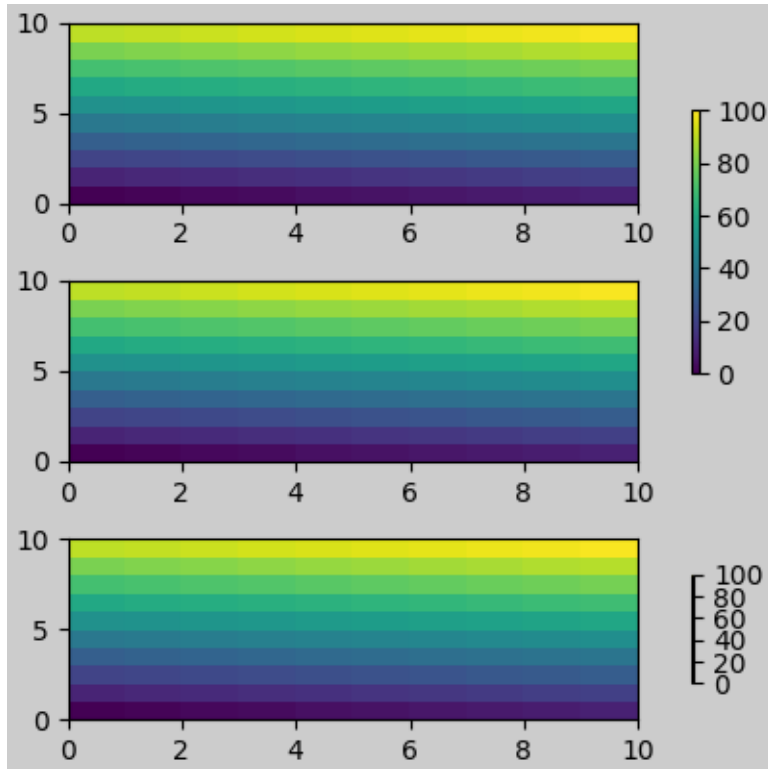
Note that there is a bit of a subtlety when specifying a single axes as the parent. In the following, it might be desirable and expected for the colorbars to line up, but they don't because the colorbar paired with the bottom axes is tied to the subplotspec of the axes, and hence shrinks when the gridspec-level colorbar is added.

```
fig, axs = plt.subplots(3, 1, figsize=(4, 4), constrained_layout=True)
for ax in axs[:2]:
    im = ax.pcolormesh(arr, **pc_kwargs)
    fig.colorbar(im, ax=axs[2], shrink=0.6)
im = axs[2].pcolormesh(arr, **pc_kwargs)
fig.colorbar(im, ax=axs[2], shrink=0.6)
```



The API to make a single-axes behave like a list of axes is to specify it as a list (or other iterable container), as below:

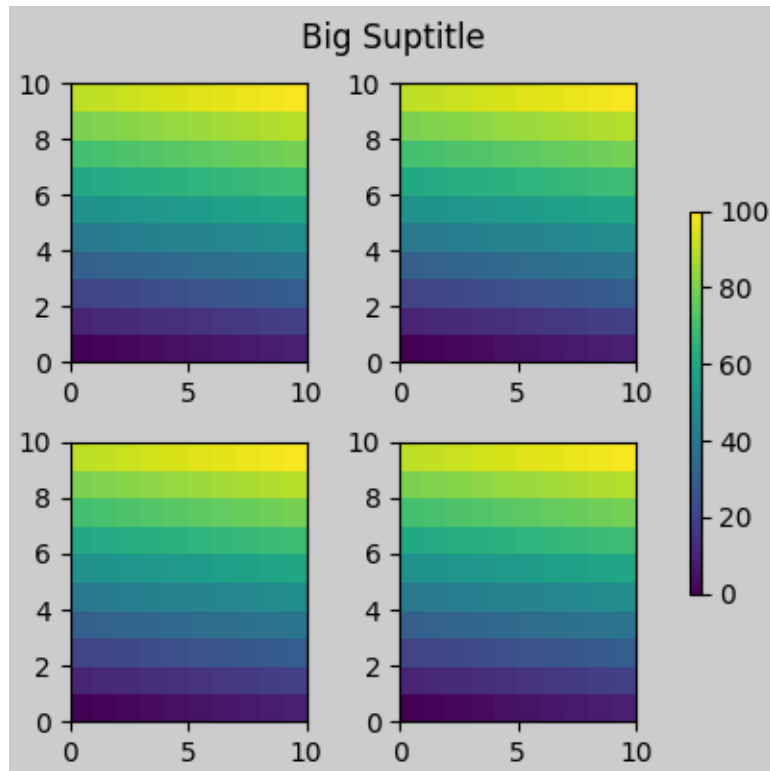
```
fig, axs = plt.subplots(3, 1, figsize=(4, 4), constrained_layout=True)
for ax in axs[:2]:
    im = ax.pcolormesh(arr, **pc_kwargs)
fig.colorbar(im, ax=axs[:2], shrink=0.6)
im = axs[2].pcolormesh(arr, **pc_kwargs)
fig.colorbar(im, ax=[axs[2]], shrink=0.6)
```



Suptitle

`constrained_layout` can also make room for `suptitle`.

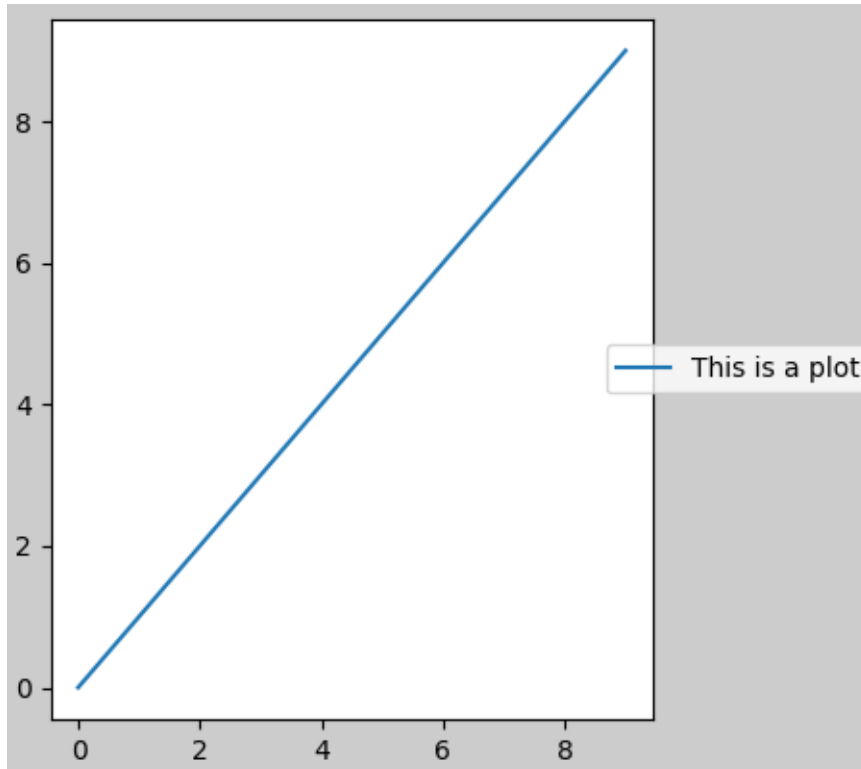
```
fig, axs = plt.subplots(2, 2, figsize=(4, 4), constrained_layout=True)
for ax in axs.flatten():
    im = ax.pcolormesh(arr, **pc_kwargs)
fig.colorbar(im, ax=axs, shrink=0.6)
fig.suptitle('Big Suptitle')
```



Legends

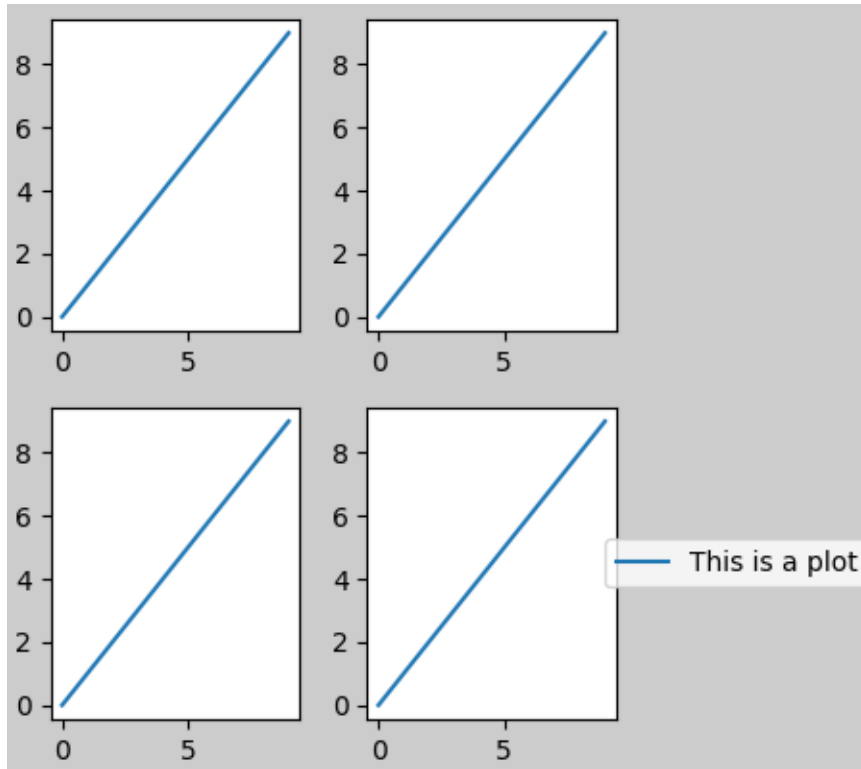
Legends can be placed outside of their parent axis. Constrained-layout is designed to handle this. However, constrained-layout does not handle legends being created via `fig.legend()` (yet).

```
fig, ax = plt.subplots(constrained_layout=True)
ax.plot(np.arange(10), label='This is a plot')
ax.legend(loc='center left', bbox_to_anchor=(0.9, 0.5))
```



However, this will steal space from a subplot layout:

```
fig, axs = plt.subplots(2, 2, constrained_layout=True)
for ax in axs.flatten()[:-1]:
    ax.plot(np.arange(10))
axs[1, 1].plot(np.arange(10), label='This is a plot')
axs[1, 1].legend(loc='center left', bbox_to_anchor=(0.9, 0.5))
```

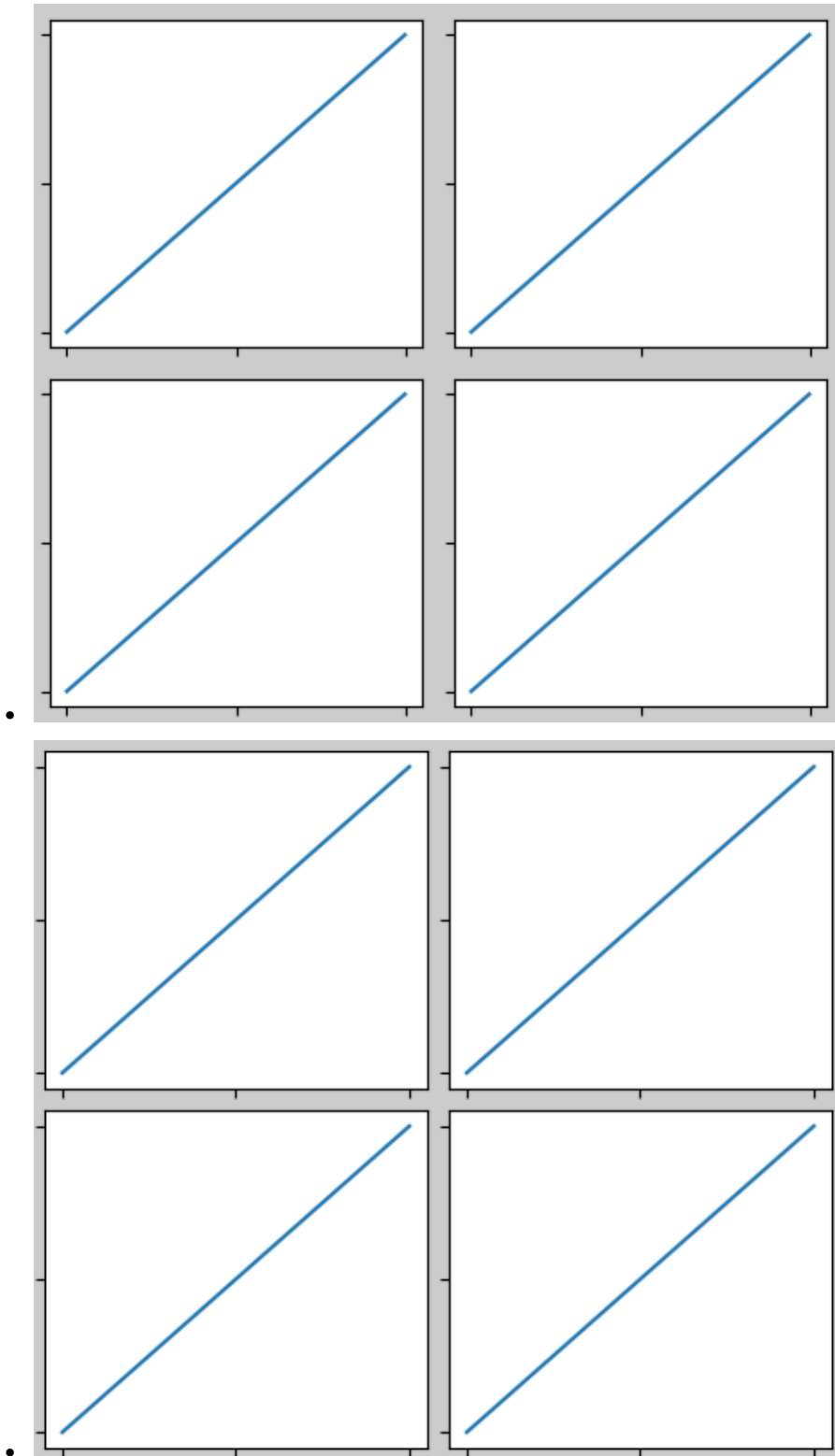


Padding and Spacing

For `constrained_layout`, we have implemented a padding around the edge of each axes. This padding sets the distance from the edge of the plot, and the minimum distance between adjacent plots. It is specified in inches by the keyword arguments `w_pad` and `h_pad` to the function `fig.set_constrained_layout_pads`:

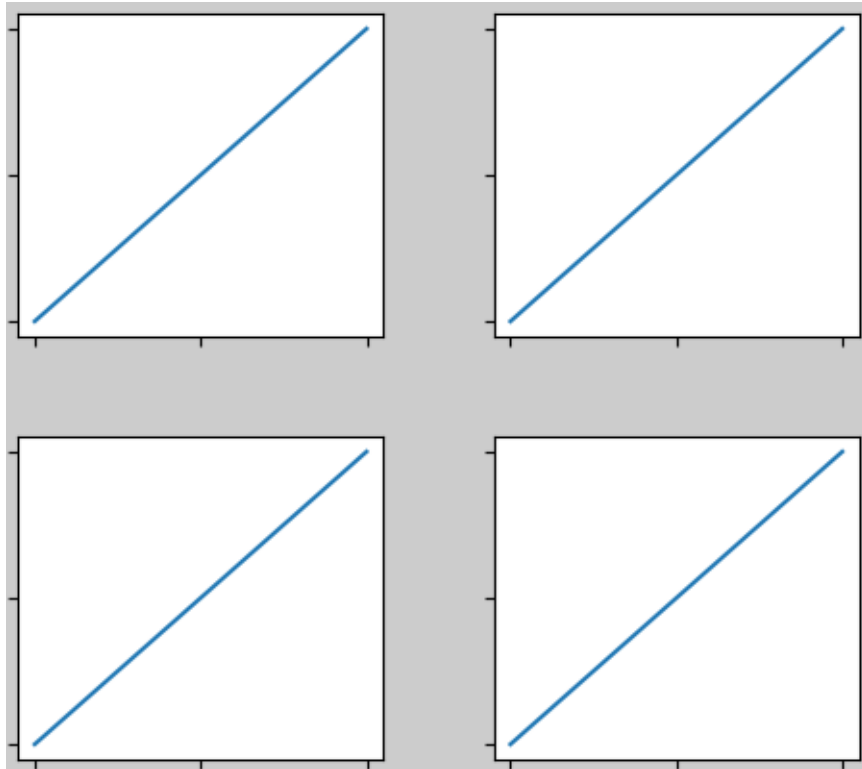
```
fig, axs = plt.subplots(2, 2, constrained_layout=True)
for ax in axs.flatten():
    example_plot(ax, nodec=True)
    ax.set_xticklabels("")
    ax.set_yticklabels("")
fig.set_constrained_layout_pads(w_pad=4./72., h_pad=4./72.,
                                hspace=0., wspace=0.)

fig, axs = plt.subplots(2, 2, constrained_layout=True)
for ax in axs.flatten():
    example_plot(ax, nodec=True)
    ax.set_xticklabels("")
    ax.set_yticklabels("")
fig.set_constrained_layout_pads(w_pad=2./72., h_pad=2./72.,
                                hspace=0., wspace=0.)
```



Spacing between subplots is set by `wspace` and `hspace`. These are specified as a fraction of the size of the subplot group as a whole. If the size of the figure is changed, then these spaces change in proportion. Note in the below how the space at the edges doesn't change from the above, but the space between subplots does.

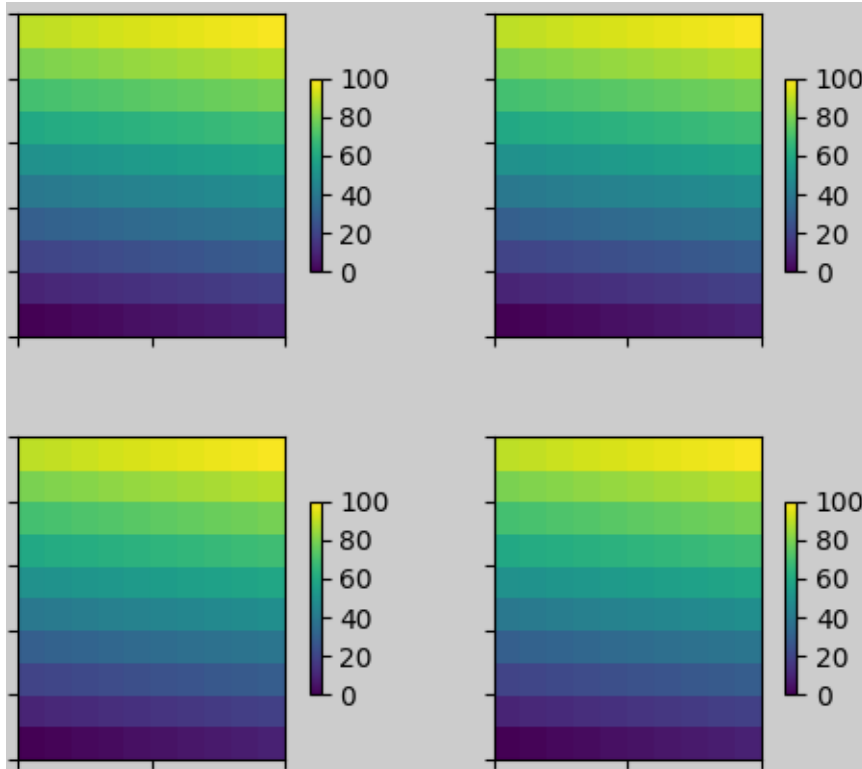

```
fig, axs = plt.subplots(2, 2, constrained_layout=True)
for ax in axs.flatten():
    example_plot(ax, nodec=True)
    ax.set_xticklabels("")
    ax.set_yticklabels("")
fig.set_constrained_layout_pads(w_pad=2./72., h_pad=2./72.,
                                hspace=0.2, wspace=0.2)
```



Spacing with colorbars

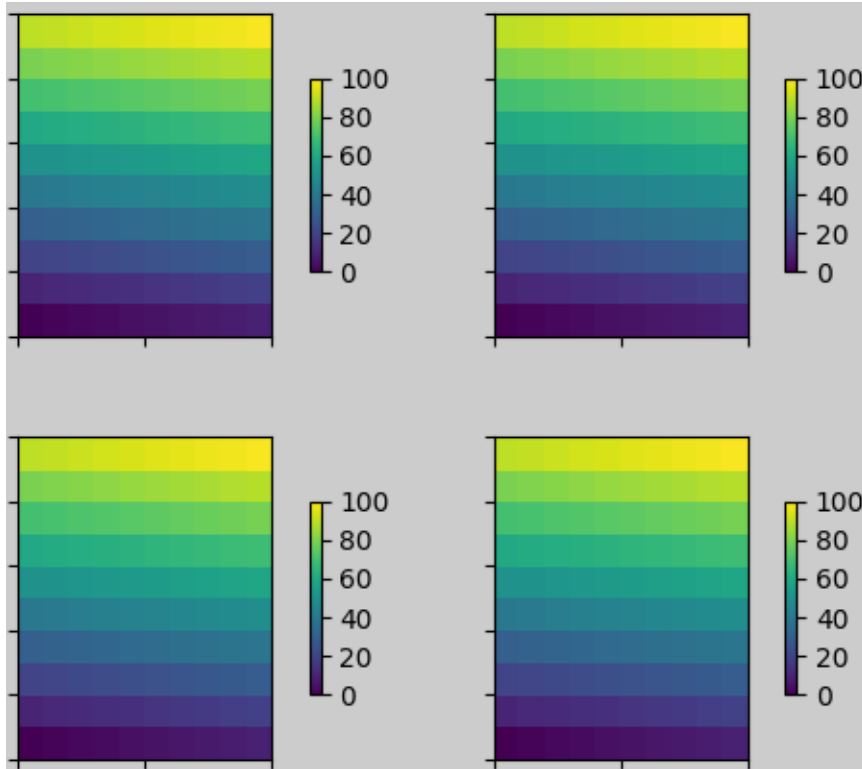
Colorbars still respect the `w_pad` and `h_pad` values. However they will be `wspace` and `hspace` apart from other subplots. Note the use of a `pad` kwarg here in the colorbar call. It defaults to 0.02 of the size of the axis it is attached to.

```
fig, axs = plt.subplots(2, 2, constrained_layout=True)
for ax in axs.flatten():
    pc = ax.pcolormesh(arr, **pc_kwargs)
    fig.colorbar(im, ax=ax, shrink=0.6, pad=0)
    ax.set_xticklabels("")
    ax.set_yticklabels("")
fig.set_constrained_layout_pads(w_pad=2./72., h_pad=2./72.,
                                hspace=0.2, wspace=0.2)
```



In the above example, the colorbar will not ever be closer than 2 pts to the plot, but if we want it a bit further away, we can specify its value for `pad` to be non-zero.

```
fig, axs = plt.subplots(2, 2, constrained_layout=True)
for ax in axs.flatten():
    pc = ax.pcolormesh(arr, **pc_kwargs)
    fig.colorbar(im, ax=ax, shrink=0.6, pad=0.05)
    ax.set_xticklabels('')
    ax.set_yticklabels('')
fig.set_constrained_layout_pads(w_pad=2./72., h_pad=2./72.,
                                hspace=0.2, wspace=0.2)
```

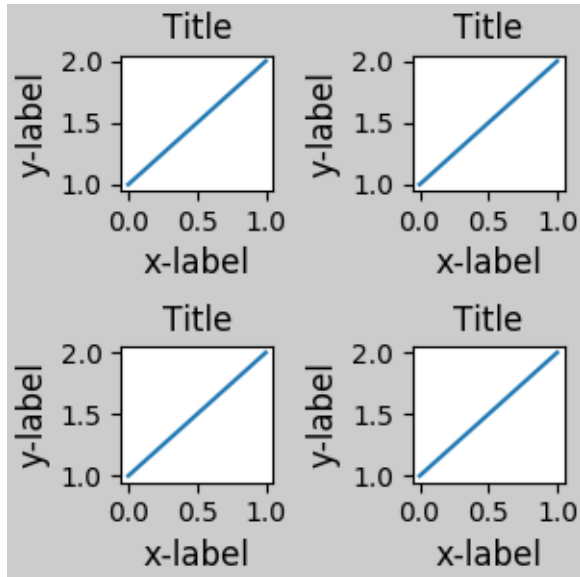


rcParams

There are five rcParams that can be set, either in a script or in the matplotlibrc file. They all have the prefix `figure.constrained_layout`:

- `use`: Whether to use `constrained_layout`. Default is `False`
- `w_pad`, `h_pad`: Padding around axes objects. Float representing inches. Default is `3./72.` inches (3 pts)
- `wspace`, `hspace`: Space between subplot groups. Float representing a fraction of the subplot widths being separated. Default is `0.02`.

```
plt.rcParams['figure.constrained_layout.use'] = True
fig, axs = plt.subplots(2, 2, figsize=(3, 3))
for ax in axs.flatten():
    example_plot(ax)
```



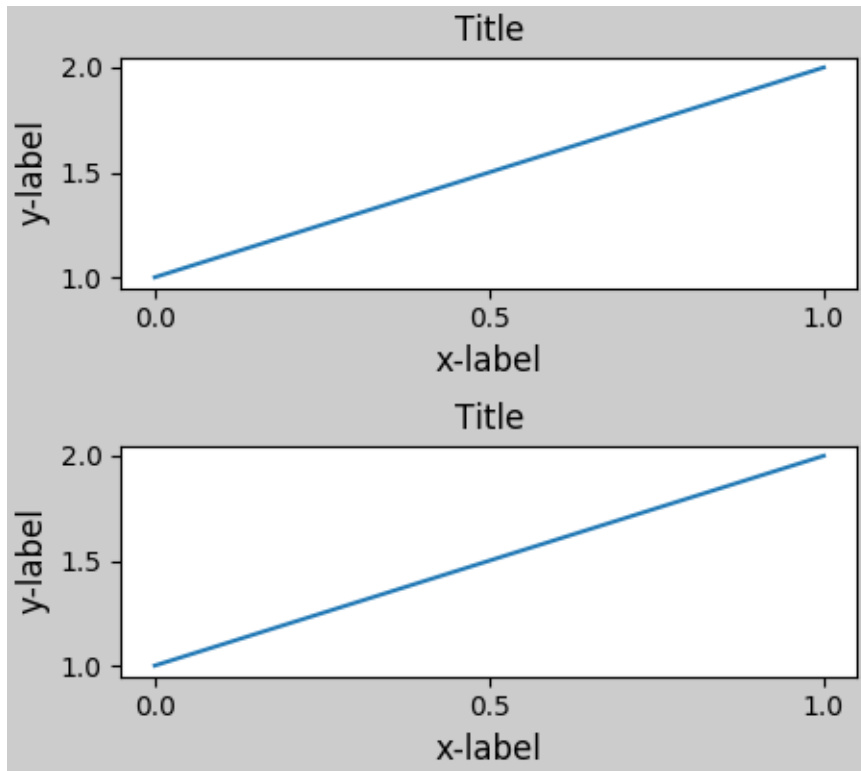
Use with `GridSpec`

`constrained_layout` is meant to be used with `subplots()` or `GridSpec()` and `add_subplot()`.

```
fig = plt.figure(constrained_layout=True)

gs1 = gridspec.GridSpec(2, 1, figure=fig)
ax1 = fig.add_subplot(gs1[0])
ax2 = fig.add_subplot(gs1[1])

example_plot(ax1)
example_plot(ax2)
```



More complicated gridspec layouts are possible...

```
fig = plt.figure(constrained_layout=True)

gs0 = gridspec.GridSpec(1, 2, figure=fig)

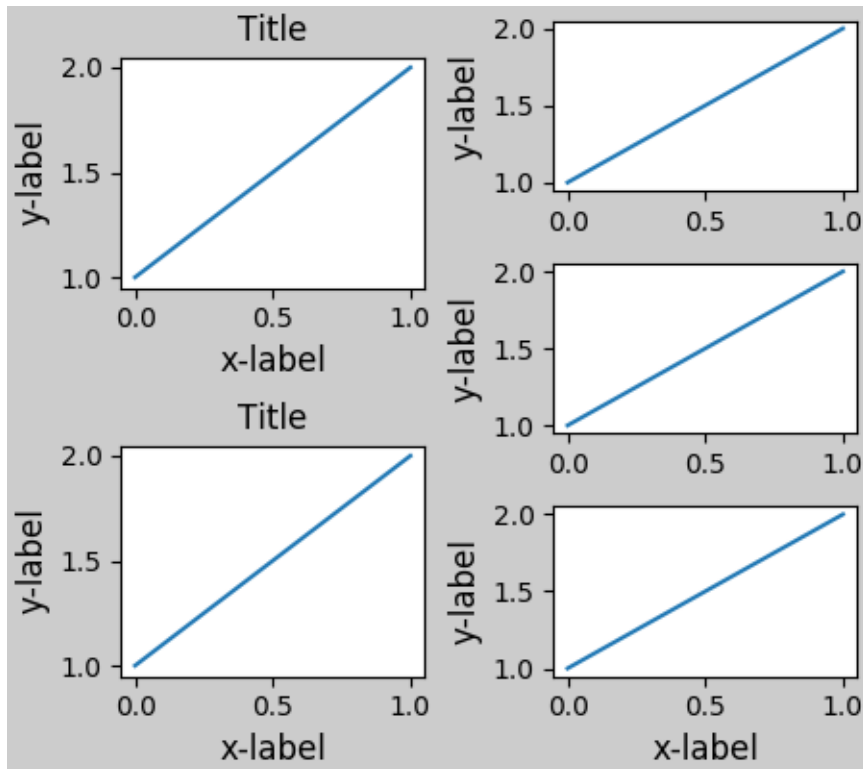
gs1 = gridspec.GridSpecFromSubplotSpec(2, 1, gs0[0])
ax1 = fig.add_subplot(gs1[0])
ax2 = fig.add_subplot(gs1[1])

example_plot(ax1)
example_plot(ax2)

gs2 = gridspec.GridSpecFromSubplotSpec(3, 1, gs0[1])

for ss in gs2:
    ax = fig.add_subplot(ss)
    example_plot(ax)
    ax.set_title("")
    ax.set_xlabel("")

ax.set_xlabel("x-label", fontsize=12)
```



Note that in the above the left and columns don't have the same vertical extent. If we want the top and bottom of the two grids to line up then they need to be in the same gridspec:

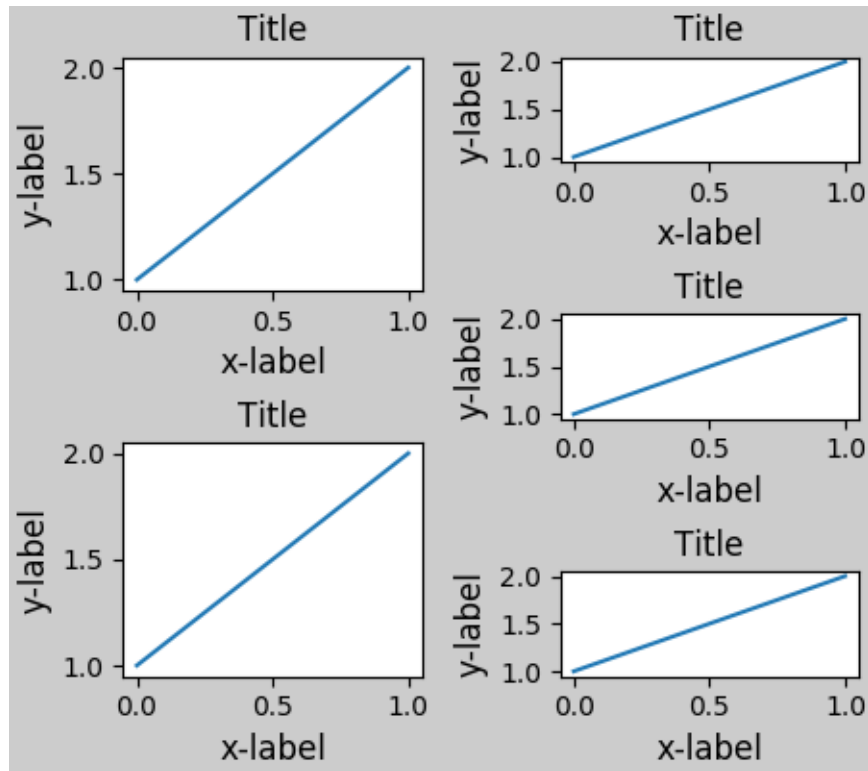
```
fig = plt.figure(constrained_layout=True)

gs0 = gridspec.GridSpec(6, 2, figure=fig)

ax1 = fig.add_subplot(gs0[3, 0])
ax2 = fig.add_subplot(gs0[3:, 0])

example_plot(ax1)
example_plot(ax2)

ax = fig.add_subplot(gs0[0:2, 1])
example_plot(ax)
ax = fig.add_subplot(gs0[2:4, 1])
example_plot(ax)
ax = fig.add_subplot(gs0[4:, 1])
example_plot(ax)
```



This example uses two gridspecs to have the colorbar only pertain to one set of pcors. Note how the left column is wider than the two right-hand columns because of this. Of course, if you wanted the subplots to be the same size you only needed one gridspec.

```
def docomplicated(suptitle=None):
    fig = plt.figure(constrained_layout=True)
    gs0 = gridspec.GridSpec(1, 2, figure=fig, width_ratios=[1., 2.])
    gsl = gridspec.GridSpecFromSubplotSpec(2, 1, gs0[0])
    gsr = gridspec.GridSpecFromSubplotSpec(2, 2, gs0[1])

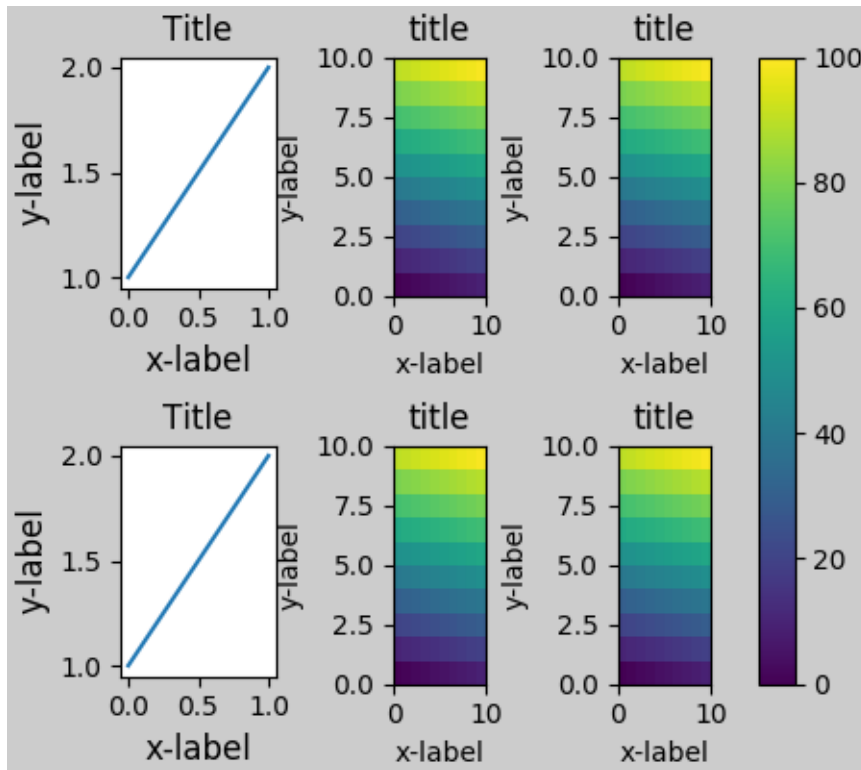
    for gs in gsl:
        ax = fig.add_subplot(gs)
        example_plot(ax)
    axs = []
    for gs in gsr:
        ax = fig.add_subplot(gs)
        pcm = ax.pcolormesh(arr, **pc_kwargs)
        ax.set_xlabel('x-label')
        ax.set_ylabel('y-label')
        ax.set_title('title')

        axs += [ax]
    fig.colorbar(pcm, ax=axs)
    if suptitle is not None:
        fig.suptitle(suptitle)
```

(continues on next page)

(continued from previous page)

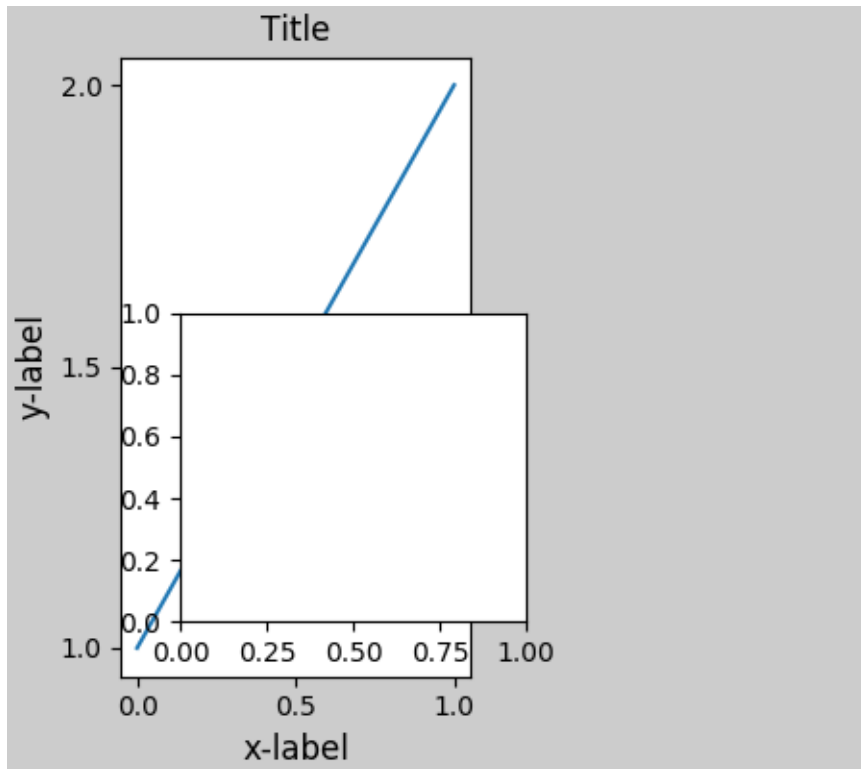
```
docomplicated()
```



Manually setting axes positions

There can be good reasons to manually set an axes position. A manual call to `ax.set_position()` will set the axes so `constrained_layout` has no effect on it anymore. (Note that `constrained_layout` still leaves the space for the axes that is moved).

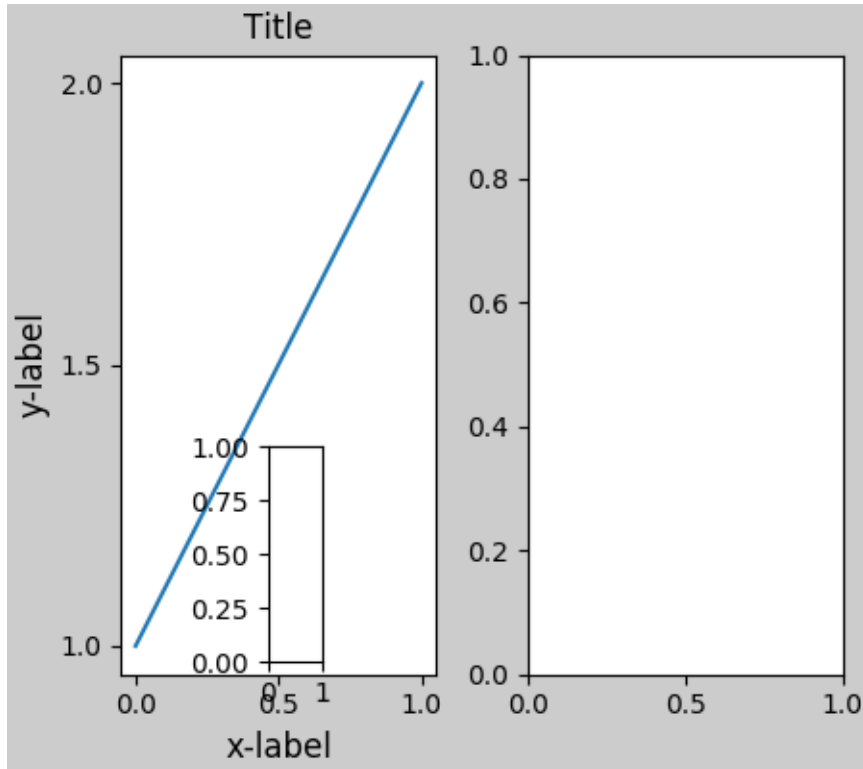
```
fig, axs = plt.subplots(1, 2, constrained_layout=True)
example_plot(axs[0], fontsize=12)
axs[1].set_position([0.2, 0.2, 0.4, 0.4])
```

If you want an inset axes in data-space, you need to manually execute the layout using `fig.execute_constrained_layout()` call. The inset figure will then be properly positioned. However, it will not be properly positioned if the size of the figure is subsequently changed. Similarly, if the figure is printed to another backend, there may be slight changes of location due to small differences in how the backends render fonts.

```
from matplotlib.transforms import Bbox

fig, axs = plt.subplots(1, 2, constrained_layout=True)
example_plot(axs[0], fontsize=12)
fig.execute_constrained_layout()
# put into data-space:
bb_data_ax2 = Bbox.from_bounds(0.5, 1., 0.2, 0.4)
disp_coords = axs[0].transData.transform(bb_data_ax2)
fig_coords_ax2 = fig.transFigure.inverted().transform(disp_coords)
bb_ax2 = Bbox(fig_coords_ax2)
ax2 = fig.add_axes(bb_ax2)
```



Limitations

Incompatible functions

`constrained_layout` will not work on subplots created via the `subplot` command. The reason is that each of these commands creates a separate `GridSpec` instance and `constrained_layout` uses (nested) gridspecs to carry out the layout. So the following fails to yield a nice layout:

```
fig = plt.figure(constrained_layout=True)
```

```
ax1 = plt.subplot(221)
```

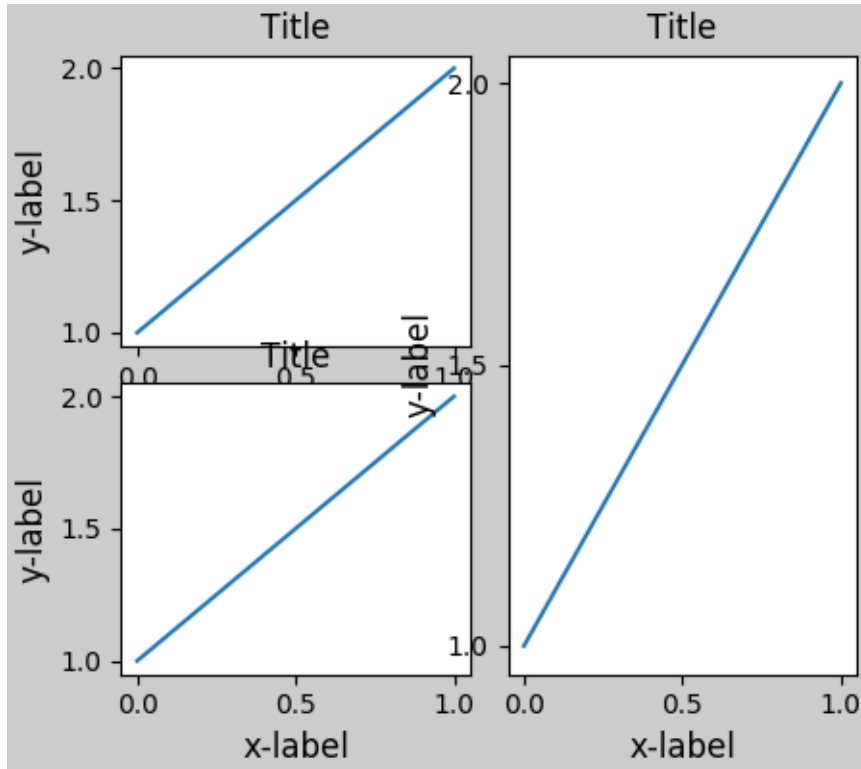
```
ax2 = plt.subplot(223)
```

```
ax3 = plt.subplot(122)
```

```
example_plot(ax1)
```

```
example_plot(ax2)
```

```
example_plot(ax3)
```

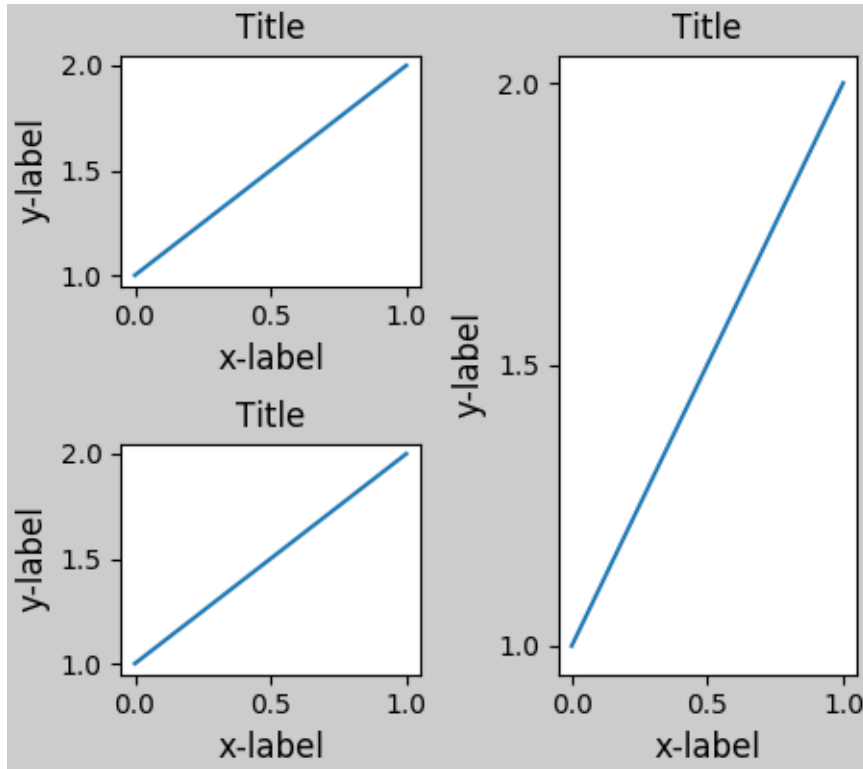


Of course that layout is possible using a gridspec:

```
fig = plt.figure(constrained_layout=True)
gs = gridspec.GridSpec(2, 2, figure=fig)

ax1 = fig.add_subplot(gs[0, 0])
ax2 = fig.add_subplot(gs[1, 0])
ax3 = fig.add_subplot(gs[:, 1])

example_plot(ax1)
example_plot(ax2)
example_plot(ax3)
```

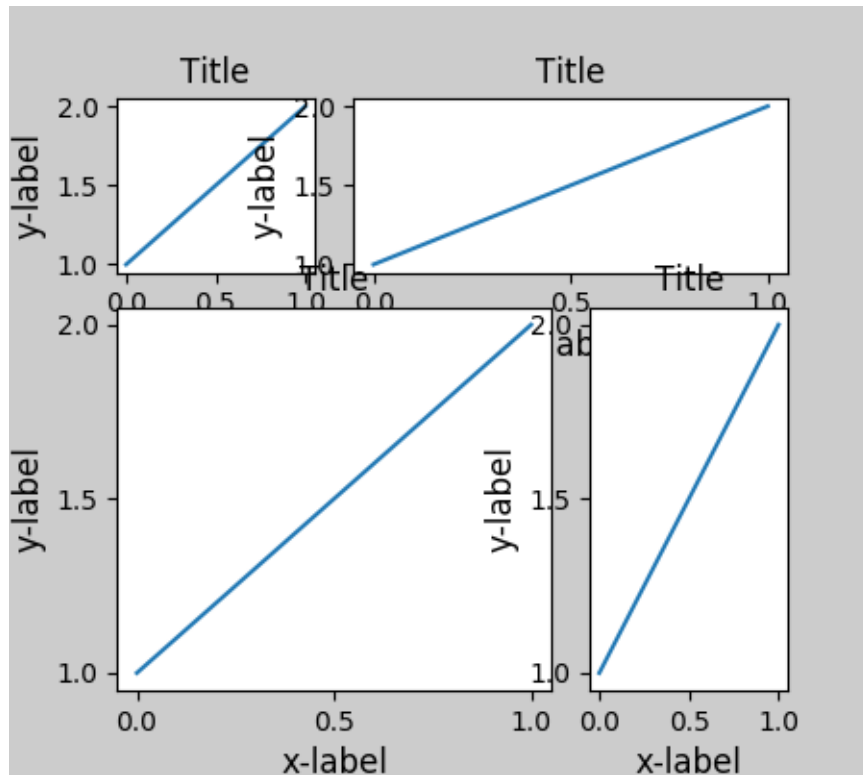


Similarly, `subplot2grid()` doesn't work for the same reason: each call creates a different parent `gridspec`.

```
fig = plt.figure(constrained_layout=True)

ax1 = plt.subplot2grid((3, 3), (0, 0))
ax2 = plt.subplot2grid((3, 3), (0, 1), colspan=2)
ax3 = plt.subplot2grid((3, 3), (1, 0), colspan=2, rowspan=2)
ax4 = plt.subplot2grid((3, 3), (1, 2), rowspan=2)

example_plot(ax1)
example_plot(ax2)
example_plot(ax3)
example_plot(ax4)
```

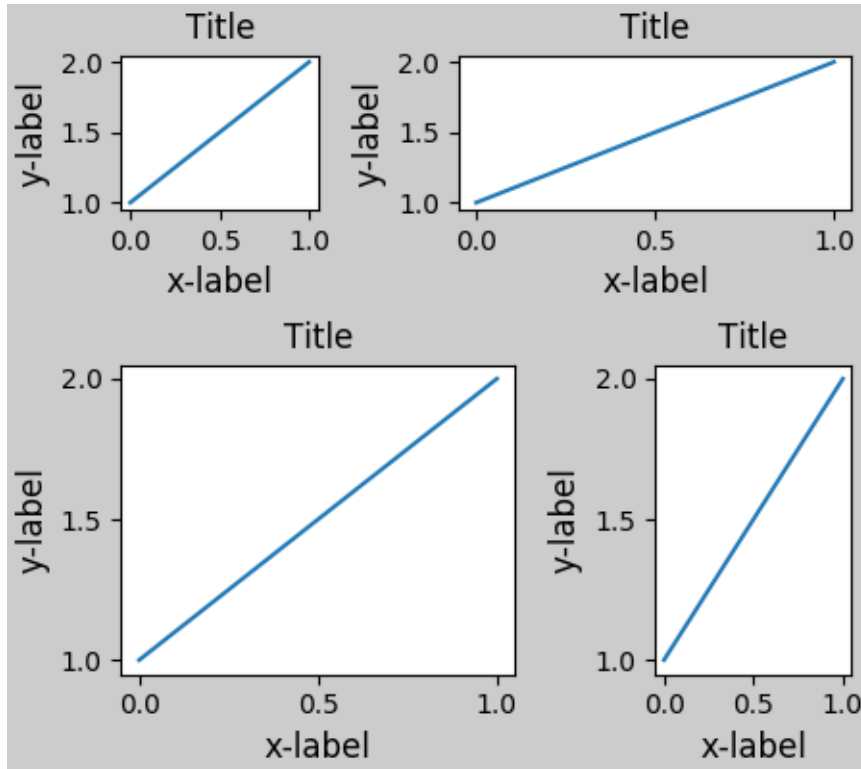


The way to make this plot compatible with `constrained_layout` is again to use `gridspec` directly

```
fig = plt.figure(constrained_layout=True)
gs = gridspec.GridSpec(3, 3, figure=fig)

ax1 = fig.add_subplot(gs[0, 0])
ax2 = fig.add_subplot(gs[0, 1:])
ax3 = fig.add_subplot(gs[1:, 0:2])
ax4 = fig.add_subplot(gs[1:, -1])

example_plot(ax1)
example_plot(ax2)
example_plot(ax3)
example_plot(ax4)
```



Other Caveats

- `constrained_layout` only considers ticklabels, axis labels, titles, and legends. Thus, other artists may be clipped and also may overlap.
- It assumes that the extra space needed for ticklabels, axis labels, and titles is independent of original location of axes. This is often true, but there are rare cases where it is not.
- There are small differences in how the backends handle rendering fonts, so the results will not be pixel-identical.

Debugging

Constrained-layout can fail in somewhat unexpected ways. Because it uses a constraint solver the solver can find solutions that are mathematically correct, but that aren't at all what the user wants. The usual failure mode is for all sizes to collapse to their smallest allowable value. If this happens, it is for one of two reasons:

1. There was not enough room for the elements you were requesting to draw.
2. There is a bug - in which case open an issue at <https://github.com/matplotlib/matplotlib/issues>.

If there is a bug, please report with a self-contained example that does not require outside data or dependencies (other than numpy).

Notes on the algorithm

The algorithm for the constraint is relatively straightforward, but has some complexity due to the complex ways we can layout a figure.

Figure layout

Figures are laid out in a hierarchy:

1. Figure: `fig = plt.figure()`
 - a. Gridspec `gs0 = gridspec.GridSpec(1, 2, figure=fig)`
 - i. Subplotspec: `ss = gs[0, 0]`
 1. Axes: `ax0 = fig.add_subplot(ss)`
 - ii. Subplotspec: `ss = gs[0, 1]`
 1. Gridspec: `gsR = gridspec.GridSpecFromSubplotSpec(2, 1, ss)`
 - Subplotspec: `ss = gsR[0, 0]`
 - Axes: `axR0 = fig.add_subplot(ss)`
 - Subplotspec: `ss = gsR[1, 0]`
 - Axes: `axR1 = fig.add_subplot(ss)`

Each item has a layoutbox associated with it. The nesting of gridspecs created with `GridSpecFromSubplotSpec` can be arbitrarily deep.

Each `~matplotlib.axes.Axes` has *two* layoutboxes. The first one, `ax._layoutbox` represents the outside of the Axes and all its decorations (i.e. ticklabels, axis labels, etc.). The second layoutbox corresponds to the Axes' `ax.position`, which sets where in the figure the spines are placed.

Why so many stacked containers? Ideally, all that would be needed are the Axes layout boxes. For the Gridspec case, a container is needed if the Gridspec is nested via `GridSpecFromSubplotSpec`. At the top level, it is desirable for symmetry, but it also makes room for `suptitle`.

For the Subplotspec/Axes case, Axes often have colorbars or other annotations that need to be packaged inside the Subplotspec, hence the need for the outer layer.

Simple case: one Axes

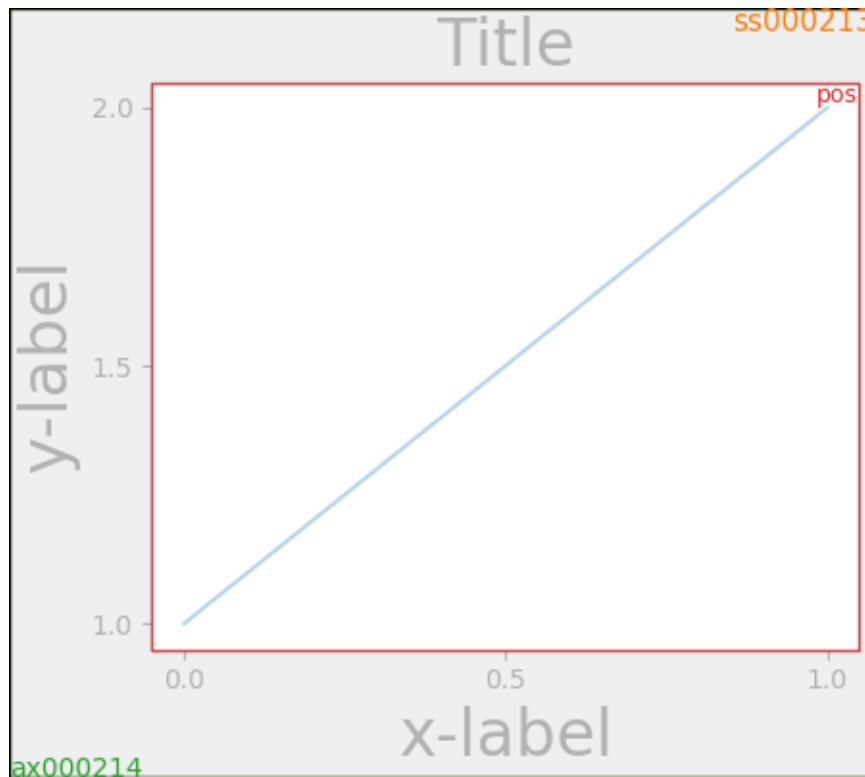
For a single Axes the layout is straight forward. The Figure and outer Gridspec layoutboxes coincide. The Subplotspec and Axes boxes also coincide because the Axes has no colorbar. Note the difference between the red pos box and the green ax box is set by the size of the decorations around the Axes.

In the code, this is accomplished by the entries in `do_constrained_layout` like:

```
ax._poslayoutbox.edit_left_margin_min(-bbox.x0 + pos.x0 + w_padt)
```

```
from matplotlib._layoutbox import plot_children
```

```
fig, ax = plt.subplots(constrained_layout=True)
example_plot(ax, fontsize=24)
plot_children(fig, fig._layoutbox, printit=False)
```



Simple case: two Axes

For this case, the Axes layoutboxes and the Subplotspec boxes still co-incide. However, because the decorations in the right-hand plot are so much smaller than the left-hand, so the right-hand layoutboxes are smaller.

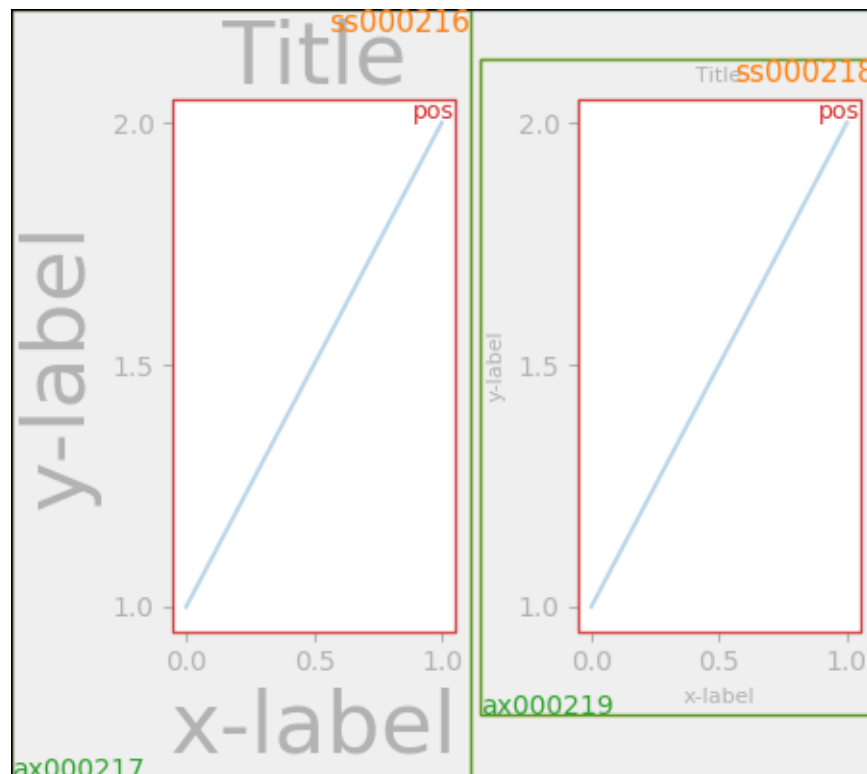
The Subplotspec boxes are laid out in the code in the subroutine `arange_subplotspecs`, which simply checks the subplotspecs in the code against one another and stacks them appropriately.

The two pos axes are lined up. Because they have the same minimum row, they are lined up at the top. Because they have the same maximum row they are lined up at the bottom. In the code this is accomplished via the calls to `layoutbox.align`. If there was more than one row, then the same horizontal alignment would occur between the rows.

The two pos axes are given the same width because the subplotspecs occupy the same number of columns. This is accomplished in the code where `dcols0` is compared to `dcolsC`. If they are equal, then their widths are constrained to be equal.

While it is a bit subtle in this case, note that the division between the Subplotspecs is not centered, but has been moved to the right to make space for the larger labels on the left-hand plot.

```
fig, ax = plt.subplots(1, 2, constrained_layout=True)
example_plot(ax[0], fontsize=32)
example_plot(ax[1], fontsize=8)
plot_children(fig, fig._layoutbox, printit=False)
```



Two Axes and colorbar

Adding a colorbar makes it clear why the Subplotspec layoutboxes must be different from the axes layoutboxes. Here we see the left-hand subplotspec has more room to accommodate the colorbar, and that there are two green ax boxes inside the ss box.

Note that the width of the pos boxes is still the same because of the constraint on their widths because their subplotspecs occupy the same number of columns (one in this example).

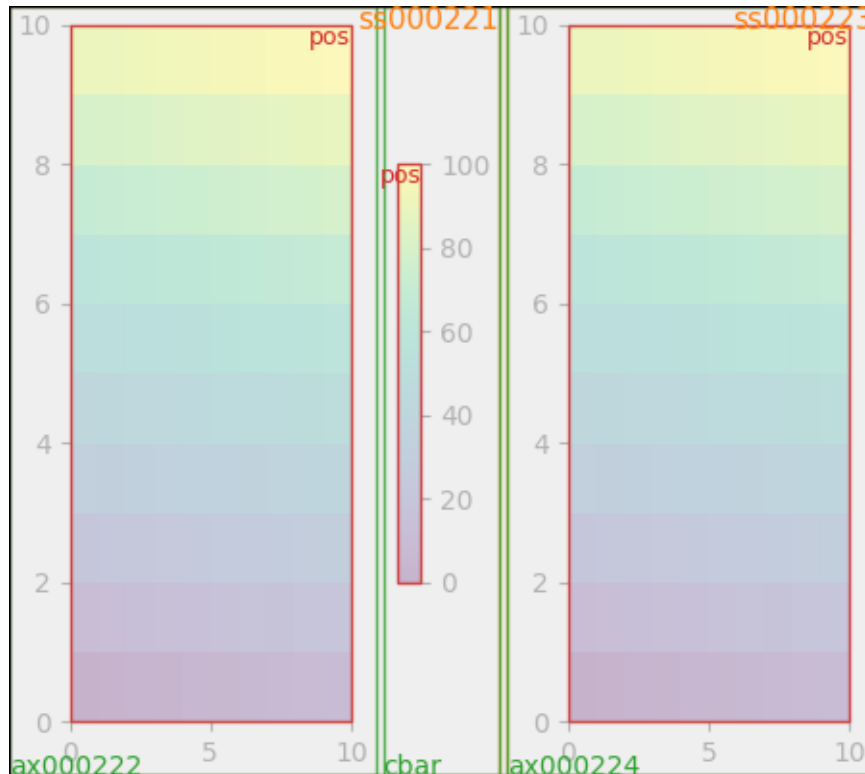
The colorbar layout logic is contained in `make_axes` which calls `__constrained_layout.layoutcolorbarsingle` for cbars attached to a single axes, and `__constrained_layout.layoutcolorbargridspec` if the colorbar is associated with a gridspec.

```
fig, ax = plt.subplots(1, 2, constrained_layout=True)
im = ax[0].pcolormesh(arr, **pc_kwargs)
fig.colorbar(im, ax=ax[0], shrink=0.6)
```

(continues on next page)

(continued from previous page)

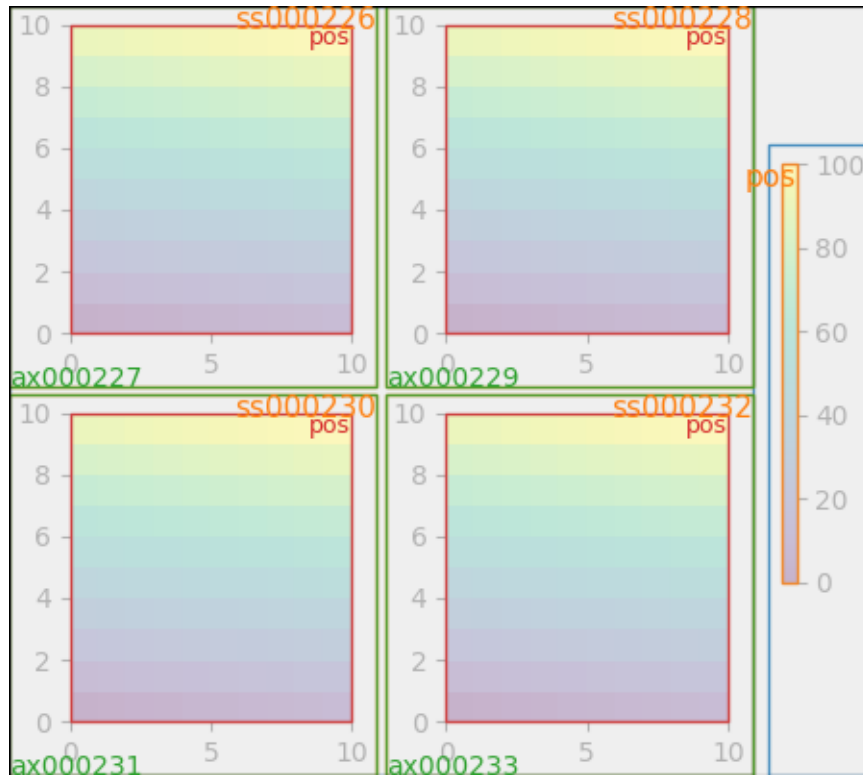
```
im = ax[1].pcolormesh(arr, **pc_kwargs)
plot_children(fig, fig._layoutbox, printit=False)
```



Colorbar associated with a Gridspec

This example shows the Subplotspec layoutboxes being made smaller by a colorbar layoutbox. The size of the colorbar layoutbox is set to be shrink smaller than the vertical extent of the pos layoutboxes in the gridspec, and it is made to be centered between those two points.

```
fig, ax = plt.subplots(2, 2, constrained_layout=True)
for a in ax.flatten():
    im = a.pcolormesh(arr, **pc_kwargs)
fig.colorbar(im, ax=ax, shrink=0.6)
plot_children(fig, fig._layoutbox, printit=False)
```



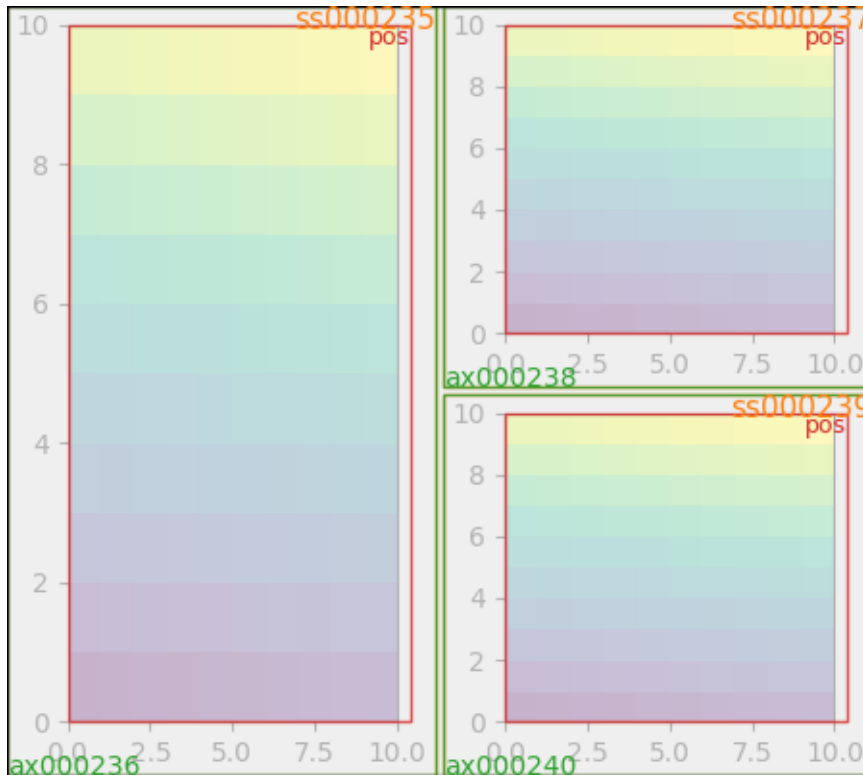
Uneven sized Axes

There are two ways to make axes have an uneven size in a Gridspec layout, either by specifying them to cross Gridspecs rows or columns, or by specifying width and height ratios.

The first method is used here. The constraint that makes the heights be correct is in the code where `drawsC < draws0` which in this case would be 1 is less than 2. So we constrain the height of the 1-row Axes to be less than half the height of the 2-row Axes.

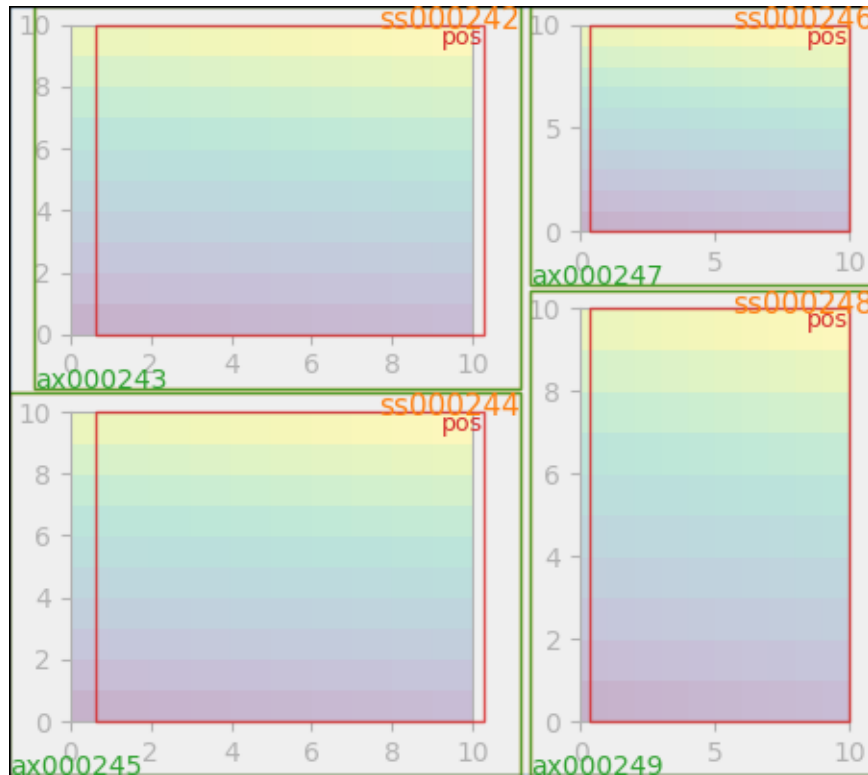
Note: This algorithm can be wrong if the decorations attached to the smaller axes are very large, so there is an unaccounted-for edge case.

```
fig = plt.figure(constrained_layout=True)
gs = gridspec.GridSpec(2, 2, figure=fig)
ax = fig.add_subplot(gs[:, 0])
im = ax.pcolormesh(arr, **pc_kwargs)
ax = fig.add_subplot(gs[0, 1])
im = ax.pcolormesh(arr, **pc_kwargs)
ax = fig.add_subplot(gs[1, 1])
im = ax.pcolormesh(arr, **pc_kwargs)
plot_children(fig, fig._layoutbox, printit=False)
```



Height and width ratios are accommodated with the same part of the code with the smaller axes always constrained to be less in size than the larger.

```
fig = plt.figure(constrained_layout=True)
gs = gridspec.GridSpec(3, 2, figure=fig,
    height_ratios=[1., 0.5, 1.5],
    width_ratios=[1.2, 0.8])
ax = fig.add_subplot(gs[:2, 0])
im = ax.pcolormesh(arr, **pc_kwargs)
ax = fig.add_subplot(gs[2, 0])
im = ax.pcolormesh(arr, **pc_kwargs)
ax = fig.add_subplot(gs[0, 1])
im = ax.pcolormesh(arr, **pc_kwargs)
ax = fig.add_subplot(gs[1:, 1])
im = ax.pcolormesh(arr, **pc_kwargs)
plot_children(fig, fig._layoutbox, printit=False)
```

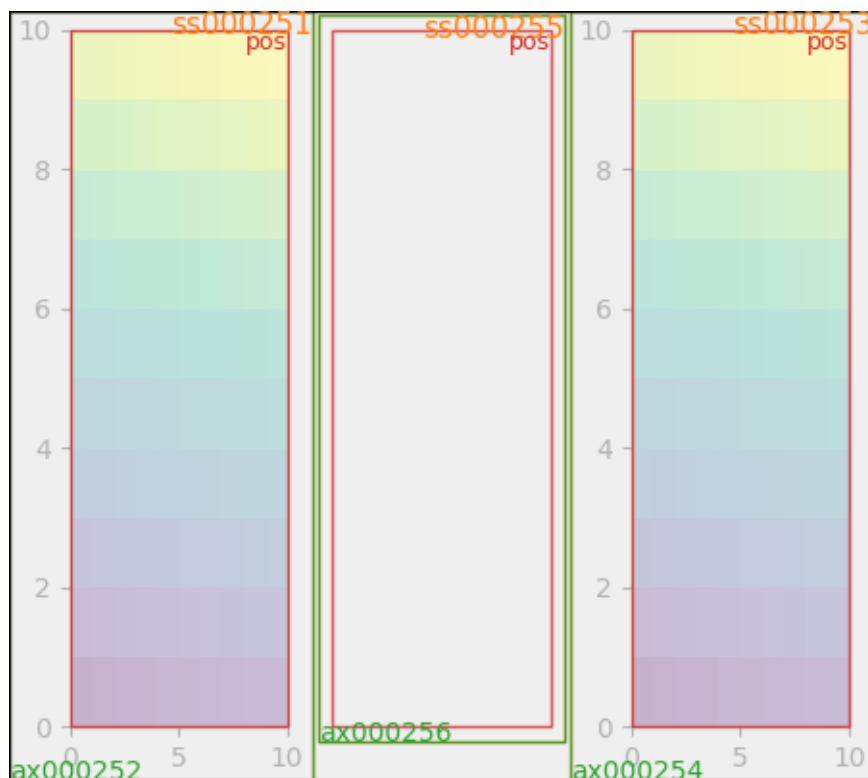


Empty gridspec slots

The final piece of the code that has not been explained is what happens if there is an empty gridspec opening. In that case a fake invisible axes is added and we proceed as before. The empty gridspec has no decorations, but the axes position is made the same size as the occupied Axes positions.

This is done at the start of `do_constrained_layout` (`hassubplotspec`).

```
fig = plt.figure(constrained_layout=True)
gs = gridspec.GridSpec(1, 3, figure=fig)
ax = fig.add_subplot(gs[0])
im = ax.pcolormesh(arr, **pc_kwargs)
ax = fig.add_subplot(gs[-1])
im = ax.pcolormesh(arr, **pc_kwargs)
plot_children(fig, fig._layoutbox, printit=False)
plt.show()
```



Other notes

The layout is called only once. This is OK if the original layout was pretty close (which it should be in most cases). However, if the layout changes a lot from the default layout then the decorators can change size. In particular the x and y tick labels can change. If this happens, then we should probably call the whole routine twice.

Total running time of the script: (0 minutes 1.676 seconds)

3.3 Advanced

These tutorials cover advanced topics for experienced Matplotlib users and developers.

Note: [Click here to download the full example code](#)

3.3.1 Path effects guide

Defining paths that objects follow on a canvas.

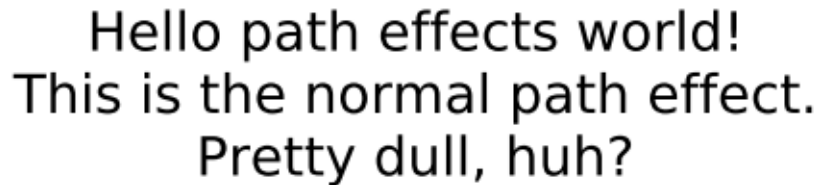
Matplotlib's [patheffects](#) module provides functionality to apply a multiple draw stage to any Artist which can be rendered via a Path.

Artists which can have a path effect applied to them include Patch, Line2D, Collection and even Text. Each artist's path effects can be controlled via the `set_path_effects` method (`set_path_effects`), which takes an iterable of `AbstractPathEffect` instances.

The simplest path effect is the Normal effect, which simply draws the artist without any effect:

```
import matplotlib.pyplot as plt
import matplotlib.path_effects as path_effects

fig = plt.figure(figsize=(5, 1.5))
text = fig.text(0.5, 0.5, 'Hello path effects world!\nThis is the normal '
                  'path effect.\nPretty dull, huh?',
                ha='center', va='center', size=20)
text.set_path_effects([path_effects.Normal()])
plt.show()
```

The image shows a plot with a white background. The text is centered and consists of three lines: "Hello path effects world!", "This is the normal path effect.", and "Pretty dull, huh?". The text is in a black, sans-serif font. The plot area is a small rectangle with a white background.

Whilst the plot doesn't look any different to what you would expect without any path effects, the drawing of the text now been changed to use the path effects framework, opening up the possibilities for more interesting examples.

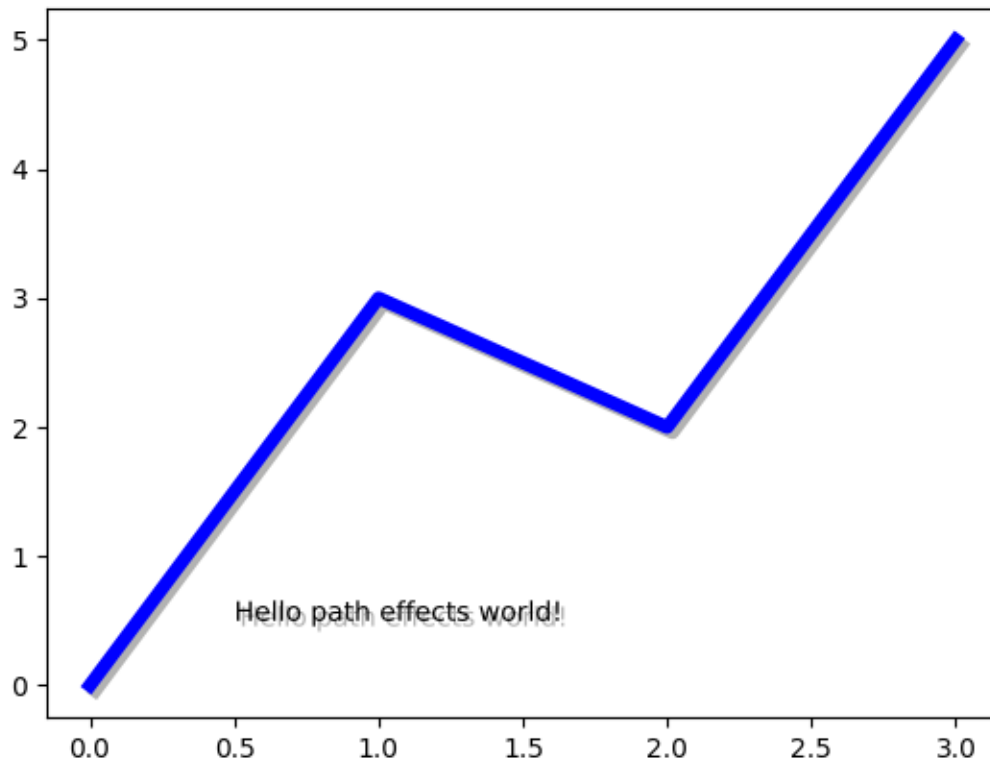
Adding a shadow

A far more interesting path effect than Normal is the drop-shadow, which we can apply to any of our path based artists. The classes `SimplePatchShadow` and `SimpleLineShadow` do precisely this by drawing either a filled patch or a line patch below the original artist:

```
import matplotlib.path_effects as path_effects

text = plt.text(0.5, 0.5, 'Hello path effects world!',
                path_effects=[path_effects.withSimplePatchShadow()])

plt.plot([0, 3, 2, 5], linewidth=5, color='blue',
         path_effects=[path_effects.SimpleLineShadow(),
                       path_effects.Normal()])
plt.show()
```



Notice the two approaches to setting the path effects in this example. The first uses the `with*` classes to include the desired functionality automatically followed with the "normal" effect, whereas the latter explicitly defines the two path effects to draw.

Making an artist stand out

One nice way of making artists visually stand out is to draw an outline in a bold color below the actual artist. The `Stroke` path effect makes this a relatively simple task:

```
fig = plt.figure(figsize=(7, 1))
text = fig.text(0.5, 0.5, 'This text stands out because of\n'
                  'its black border.', color='white',
                  ha='center', va='center', size=30)
text.set_path_effects([path_effects.Stroke(linewidth=3, foreground='black'),
                       path_effects.Normal()])
plt.show()
```


This text stands out because of its black border.

It is important to note that this effect only works because we have drawn the text path twice; once with a thick black line, and then once with the original text path on top.

You may have noticed that the keywords to `Stroke` and `SimplePatchShadow` and `SimpleLineShadow` are not the usual Artist keywords (such as `facecolor` and `edgecolor` etc.). This is because with these path effects we are operating at lower level of matplotlib. In fact, the keywords which are accepted are those for a `matplotlib.backend_bases.GraphicsContextBase` instance, which have been designed for making it easy to create new backends - and not for its user interface.

Greater control of the path effect artist

As already mentioned, some of the path effects operate at a lower level than most users will be used to, meaning that setting keywords such as `facecolor` and `edgecolor` raise an `AttributeError`. Luckily there is a generic `PathPatchEffect` path effect which creates a `PathPatch` class with the original path. The keywords to this effect are identical to those of `PathPatch`:

```
fig = plt.figure(figsize=(8, 1))
t = fig.text(0.02, 0.5, 'Hatch shadow', fontsize=75, weight=1000, va='center')
t.set_path_effects([path_effects.PathPatchEffect(offset=(4, -4), hatch='xxxx',
                                                  facecolor='gray'),
                   path_effects.PathPatchEffect(edgecolor='white', linewidth=1.1,
                                                  facecolor='black')])
plt.show()
```

Hatch shadow

Note: [Click here to download the full example code](#)

3.3.2 Path Tutorial

Defining paths in your Matplotlib visualization.

The object underlying all of the `matplotlib.patch` objects is the `Path`, which supports the standard set of `moveto`, `lineto`, `curveto` commands to draw simple and compound outlines consisting of line segments and splines. The `Path` is instantiated with a `(N,2)` array of `(x,y)` vertices, and a `N`-length array of path codes. For example to draw the unit rectangle from `(0,0)` to `(1,1)`, we could use this code

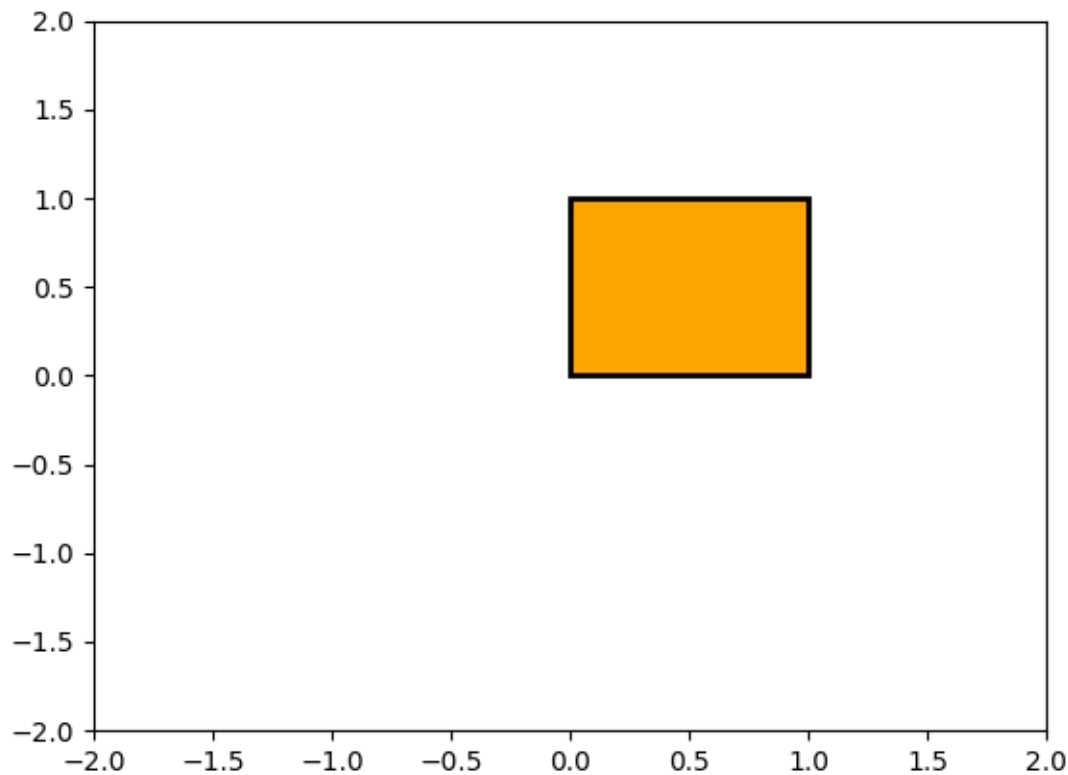
```
import matplotlib.pyplot as plt
from matplotlib.path import Path
import matplotlib.patches as patches

verts = [
    (0., 0.), # left, bottom
    (0., 1.), # left, top
    (1., 1.), # right, top
    (1., 0.), # right, bottom
    (0., 0.), # ignored
]

codes = [
    Path.MOVETO,
    Path.LINETO,
    Path.LINETO,
    Path.LINETO,
    Path.CLOSEPOLY,
]

path = Path(verts, codes)

fig, ax = plt.subplots()
patch = patches.PathPatch(path, facecolor='orange', lw=2)
ax.add_patch(patch)
ax.set_xlim(-2, 2)
ax.set_ylim(-2, 2)
plt.show()
```



The following path codes are recognized

Code	Vertices	Description
STOP	1 (ignored)	A marker for the end of the entire path (currently not required and ignored)
MOVETO	1	Pick up the pen and move to the given vertex.
LINETO	1	Draw a line from the current position to the given vertex.
CURVE3	2 (1 control point, 1 endpoint)	Draw a quadratic Bézier curve from the current position, with the given control point, to the given end point.
CURVE4	3 (2 control points, 1 end-point)	Draw a cubic Bézier curve from the current position, with the given control points, to the given end point.
CLOSE-POLY	1 (point itself is ignored)	Draw a line segment to the start point of the current polyline.

Bézier example

Some of the path components require multiple vertices to specify them: for example CURVE 3 is a [bézier](#) curve with one control point and one end point, and CURVE4 has three vertices for the two control points and the end point. The example below shows a CURVE4 Bézier spline – the [bézier](#)

curve will be contained in the convex hull of the start point, the two control points, and the end point

```
verts = [
    (0., 0.), # P0
    (0.2, 1.), # P1
    (1., 0.8), # P2
    (0.8, 0.), # P3
]

codes = [
    Path.MOVETO,
    Path.CURVE4,
    Path.CURVE4,
    Path.CURVE4,
]

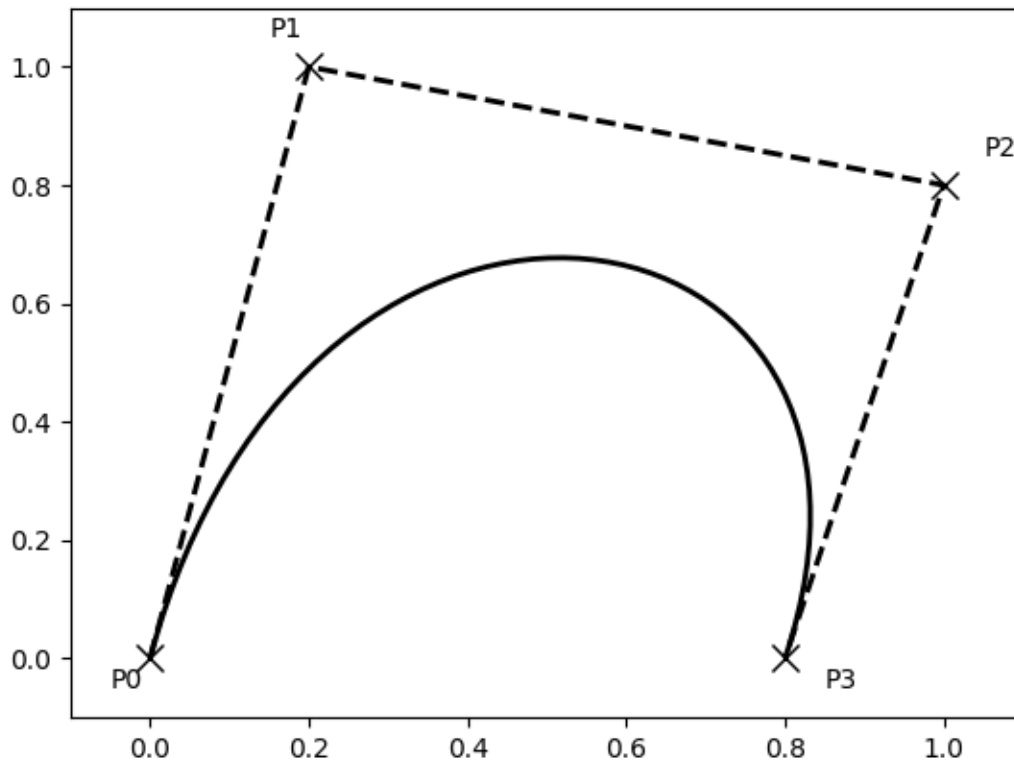
path = Path(verts, codes)

fig, ax = plt.subplots()
patch = patches.PathPatch(path, facecolor='none', lw=2)
ax.add_patch(patch)

xs, ys = zip(*verts)
ax.plot(xs, ys, 'x--', lw=2, color='black', ms=10)

ax.text(-0.05, -0.05, 'P0')
ax.text(0.15, 1.05, 'P1')
ax.text(1.05, 0.85, 'P2')
ax.text(0.85, -0.05, 'P3')

ax.set_xlim(-0.1, 1.1)
ax.set_ylim(-0.1, 1.1)
plt.show()
```



Compound paths

All of the simple patch primitives in matplotlib, `Rectangle`, `Circle`, `Polygon`, etc, are implemented with simple path. Plotting functions like `hist()` and `bar()`, which create a number of primitives, e.g., a bunch of `Rectangles`, can usually be implemented more efficiently using a compound path. The reason `bar` creates a list of rectangles and not a compound path is largely historical: the `Path` code is comparatively new and `bar` predates it. While we could change it now, it would break old code, so here we will cover how to create compound paths, replacing the functionality in `bar`, in case you need to do so in your own code for efficiency reasons, e.g., you are creating an animated bar plot.

We will make the histogram chart by creating a series of rectangles for each histogram bar: the rectangle width is the bin width and the rectangle height is the number of datapoints in that bin. First we'll create some random normally distributed data and compute the histogram. Because `numpy` returns the bin edges and not centers, the length of bins is 1 greater than the length of `n` in the example below:

```
# histogram our data with numpy
data = np.random.randn(1000)
n, bins = np.histogram(data, 100)
```

We'll now extract the corners of the rectangles. Each of the left, bottom, etc, arrays below is `len(n)`, where `n` is the array of counts for each histogram bar:

```
# get the corners of the rectangles for the histogram
left = np.array(bins[:-1])
right = np.array(bins[1:])
bottom = np.zeros(len(left))
top = bottom + n
```

Now we have to construct our compound path, which will consist of a series of MOVETO, LINETO and CLOSEPOLY for each rectangle. For each rectangle, we need 5 vertices: 1 for the MOVETO, 3 for the LINETO, and 1 for the CLOSEPOLY. As indicated in the table above, the vertex for the closepoly is ignored but we still need it to keep the codes aligned with the vertices:

```
nverts = nrects*(1+3+1)
verts = np.zeros((nverts, 2))
codes = np.ones(nverts, int) * path.Path.LINETO
codes[0::5] = path.Path.MOVETO
codes[4::5] = path.Path.CLOSEPOLY
verts[0::5,0] = left
verts[0::5,1] = bottom
verts[1::5,0] = left
verts[1::5,1] = top
verts[2::5,0] = right
verts[2::5,1] = top
verts[3::5,0] = right
verts[3::5,1] = bottom
```

All that remains is to create the path, attach it to a PathPatch, and add it to our axes:

```
barpath = path.Path(verts, codes)
patch = patches.PathPatch(barpath, facecolor='green',
    edgecolor='yellow', alpha=0.5)
ax.add_patch(patch)
```

```
import numpy as np
import matplotlib.patches as patches
import matplotlib.path as path

fig, ax = plt.subplots()
# Fixing random state for reproducibility
np.random.seed(19680801)

# histogram our data with numpy
data = np.random.randn(1000)
n, bins = np.histogram(data, 100)
```

(continues on next page)

(continued from previous page)

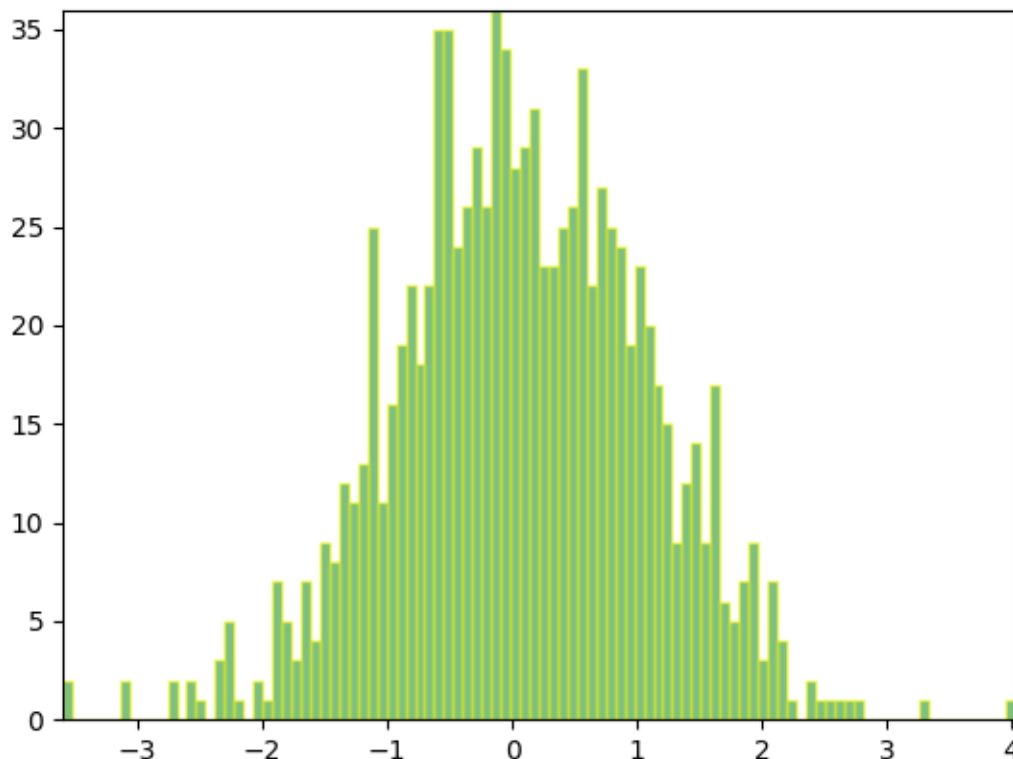
```
# get the corners of the rectangles for the histogram
left = np.array(bins[:-1])
right = np.array(bins[1:])
bottom = np.zeros(len(left))
top = bottom + n
nrects = len(left)

nverts = nrects*(1+3+1)
verts = np.zeros((nverts, 2))
codes = np.ones(nverts, int) * path.Path.LINETO
codes[0::5] = path.Path.MOVETO
codes[4::5] = path.Path.CLOSEPOLY
verts[0::5, 0] = left
verts[0::5, 1] = bottom
verts[1::5, 0] = left
verts[1::5, 1] = top
verts[2::5, 0] = right
verts[2::5, 1] = top
verts[3::5, 0] = right
verts[3::5, 1] = bottom

barpath = path.Path(verts, codes)
patch = patches.PathPatch(barpath, facecolor='green',
                           edgecolor='yellow', alpha=0.5)
ax.add_patch(patch)

ax.set_xlim(left[0], right[-1])
ax.set_ylim(bottom.min(), top.max())

plt.show()
```



Note: [Click here to download the full example code](#)

3.3.3 Transformations Tutorial

Like any graphics packages, Matplotlib is built on top of a transformation framework to easily move between coordinate systems, the userland data coordinate system, the axes coordinate system, the figure coordinate system, and the display coordinate system. In 95% of your plotting, you won't need to think about this, as it happens under the hood, but as you push the limits of custom figure generation, it helps to have an understanding of these objects so you can reuse the existing transformations Matplotlib makes available to you, or create your own (see `matplotlib.transforms`). The table below summarizes the some useful coordinate systems, the transformation object you should use to work in that coordinate system, and the description of that system. In the Transformation Object column, `ax` is a `Axes` instance, and `fig` is a `Figure` instance.

Co-ordinates	Transformation object	Description
"data"	ax.transData	The coordinate system for the data, controlled by xlim and ylim.
"axes"	ax.transAxes	The coordinate system of the Axes; (0, 0) is bottom left of the axes, and (1, 1) is top right of the axes.
"figure"	fig.transFigure	The coordinate system of the Figure; (0, 0) is bottom left of the figure, and (1, 1) is top right of the figure.
"display"	None, or IdentityTransform()	The pixel coordinate system of the display; (0, 0) is bottom left of the display, and (width, height) is top right of the display in pixels.
"xaxis", "yaxis"	ax. get_xaxis_transform(), ax. get_yaxis_transform()	Blended coordinate systems; use data coordinates on one of the axis and axes coordinates on the other.

All of the transformation objects in the table above take inputs in their coordinate system, and transform the input to the display coordinate system. That is why the display coordinate system has `None` for the Transformation Object column – it already is in display coordinates. The transformations also know how to invert themselves, to go from display back to the native coordinate system. This is particularly useful when processing events from the user interface, which typically occur in display space, and you want to know where the mouse click or key-press occurred in your data coordinate system.

Data coordinates

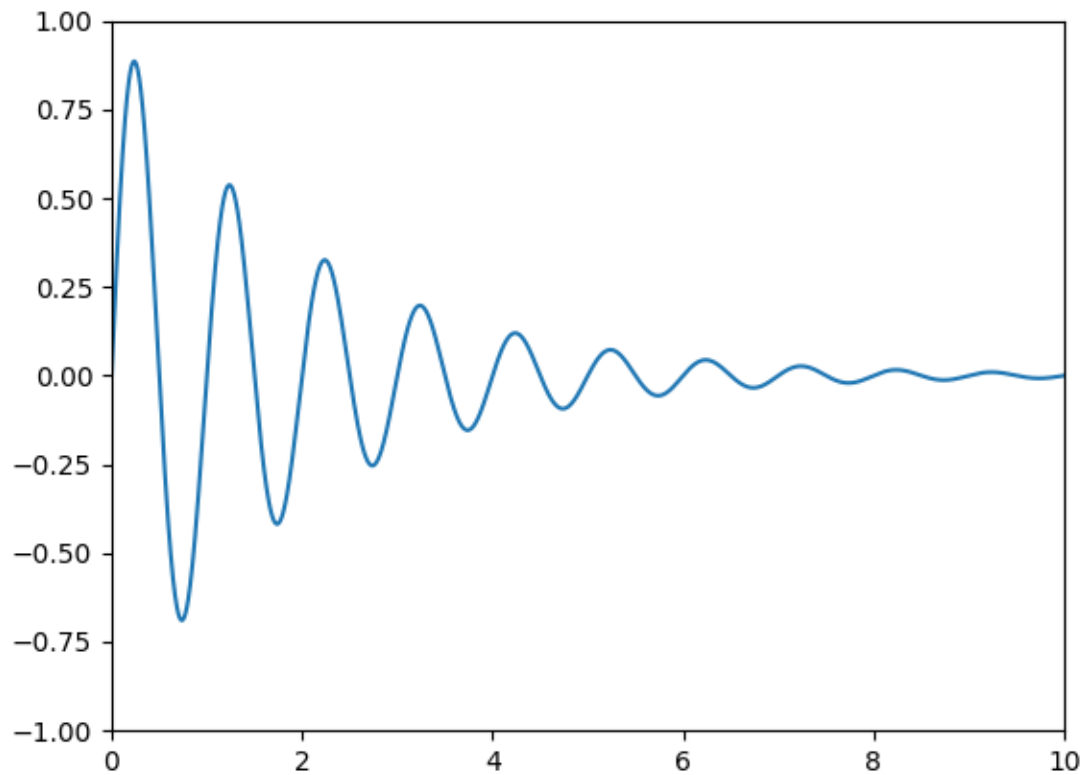
Let's start with the most commonly used coordinate, the data coordinate system. Whenever you add data to the axes, Matplotlib updates the datalimits, most commonly updated with the `set_xlim()` and `set_ylim()` methods. For example, in the figure below, the data limits stretch from 0 to 10 on the x-axis, and -1 to 1 on the y-axis.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 10, 0.005)
y = np.exp(-x/2.) * np.sin(2*np.pi*x)

fig, ax = plt.subplots()
ax.plot(x, y)
ax.set_xlim(0, 10)
ax.set_ylim(-1, 1)

plt.show()
```



You can use the `ax.transData` instance to transform from your data to your display coordinate system, either a single point or a sequence of points as shown below:

```
In [14]: type(ax.transData)
Out[14]: <class 'matplotlib.transforms.CompositeGenericTransform'>

In [15]: ax.transData.transform((5, 0))
Out[15]: array([ 335.175, 247.  ])

In [16]: ax.transData.transform([(5, 0), (1, 2)])
Out[16]:
array([[ 335.175, 247.  ],
       [ 132.435, 642.2 ]])
```

You can use the `inverted()` method to create a transform which will take you from display to data coordinates:

```
In [41]: inv = ax.transData.inverted()

In [42]: type(inv)
Out[42]: <class 'matplotlib.transforms.CompositeGenericTransform'>
```

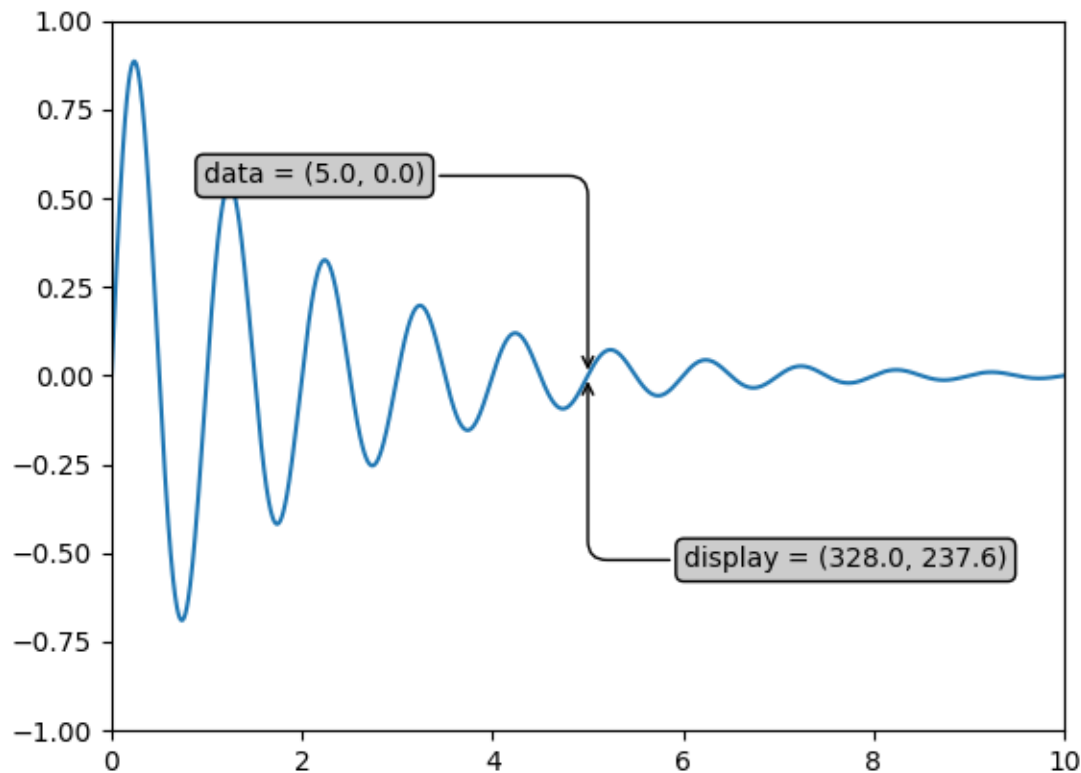
(continues on next page)

(continued from previous page)

```
In [43]: inv.transform((335.175, 247.))  
Out[43]: array([ 5.,  0.]
```

If you are typing along with this tutorial, the exact values of the display coordinates may differ if you have a different window size or dpi setting. Likewise, in the figure below, the display labeled points are probably not the same as in the ipython session because the documentation figure size defaults are different.

```
x = np.arange(0, 10, 0.005)  
y = np.exp(-x/2.) * np.sin(2*np.pi*x)  
  
fig, ax = plt.subplots()  
ax.plot(x, y)  
ax.set_xlim(0, 10)  
ax.set_ylim(-1, 1)  
  
xdata, ydata = 5, 0  
xdisplay, ydisplay = ax.transData.transform_point((xdata, ydata))  
  
bbox = dict(boxstyle="round", fc="0.8")  
arrowprops = dict(  
    arrowstyle="->",  
    connectionstyle="angle,angleA=0,angleB=90,rad=10")  
  
offset = 72  
ax.annotate('data = (%.1f, %.1f)' % (xdata, ydata),  
            (xdata, ydata), xytext=(-2*offset, offset), textcoords='offset points',  
            bbox=bbox, arrowprops=arrowprops)  
  
disp = ax.annotate('display = (%.1f, %.1f)' % (xdisplay, ydisplay),  
                   (xdisplay, ydisplay), xytext=(0.5*offset, -offset),  
                   xycoords='figure pixels',  
                   textcoords='offset points',  
                   bbox=bbox, arrowprops=arrowprops)  
  
plt.show()
```



Note: If you run the source code in the example above in a GUI backend, you may also find that the two arrows for the data and display annotations do not point to exactly the same point. This is because the display point was computed before the figure was displayed, and the GUI backend may slightly resize the figure when it is created. The effect is more pronounced if you resize the figure yourself. This is one good reason why you rarely want to work in display space, but you can connect to the 'on_draw' Event to update figure coordinates on figure draws; see [Event handling and picking](#).

When you change the x or y limits of your axes, the data limits are updated so the transformation yields a new display point. Note that when we just change the ylim, only the y-display coordinate is altered, and when we change the xlim too, both are altered. More on this later when we talk about the Bbox.

```
In [54]: ax.transData.transform((5, 0))
```

```
Out[54]: array([ 335.175, 247.  ])
```

```
In [55]: ax.set_ylim(-1, 2)
```

```
Out[55]: (-1, 2)
```

(continues on next page)

(continued from previous page)

```
In [56]: ax.transData.transform((5, 0))
Out[56]: array([ 335.175    , 181.13333333])

In [57]: ax.set_xlim(10, 20)
Out[57]: (10, 20)

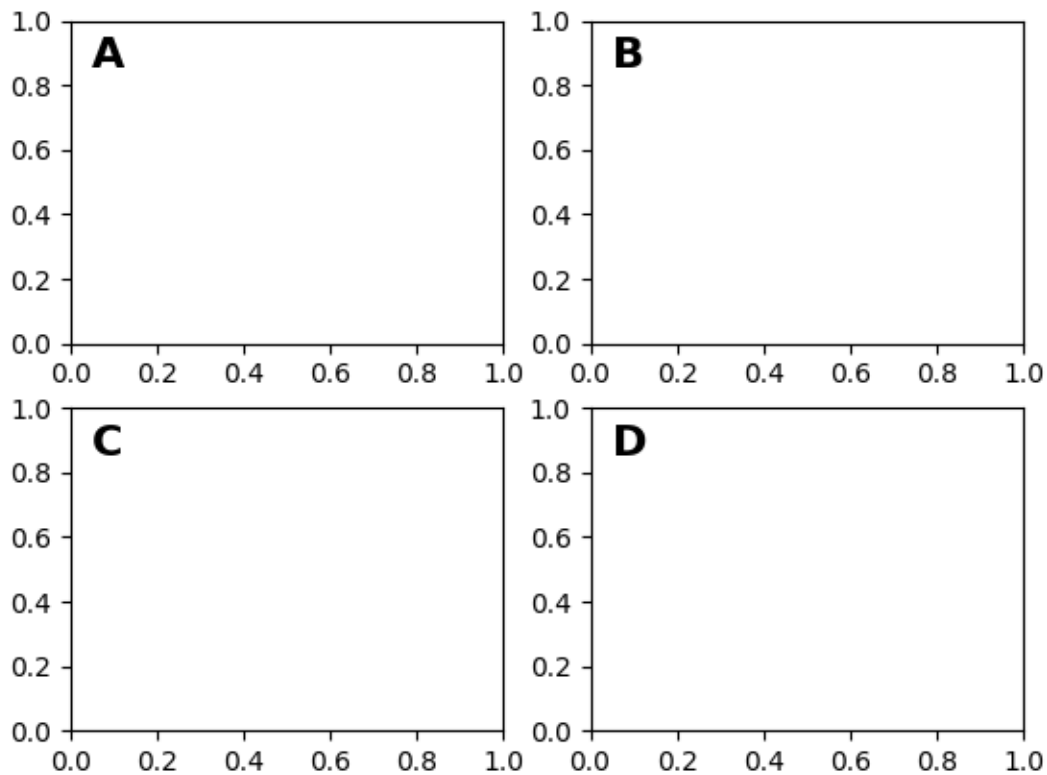
In [58]: ax.transData.transform((5, 0))
Out[58]: array([-171.675    , 181.13333333])
```

Axes coordinates

After the data coordinate system, axes is probably the second most useful coordinate system. Here the point (0, 0) is the bottom left of your axes or subplot, (0.5, 0.5) is the center, and (1.0, 1.0) is the top right. You can also refer to points outside the range, so (-0.1, 1.1) is to the left and above your axes. This coordinate system is extremely useful when placing text in your axes, because you often want a text bubble in a fixed, location, e.g., the upper left of the axes pane, and have that location remain fixed when you pan or zoom. Here is a simple example that creates four panels and labels them 'A', 'B', 'C', 'D' as you often see in journals.

```
fig = plt.figure()
for i, label in enumerate(('A', 'B', 'C', 'D')):
    ax = fig.add_subplot(2, 2, i+1)
    ax.text(0.05, 0.95, label, transform=ax.transAxes,
           fontsize=16, fontweight='bold', va='top')

plt.show()
```



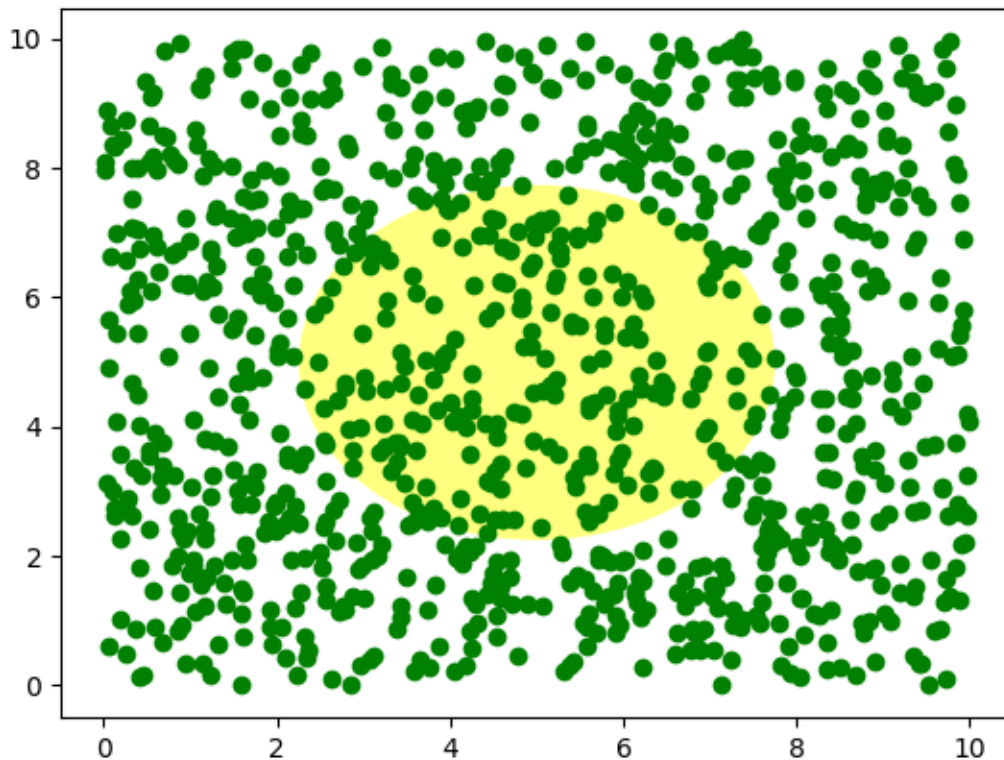
You can also make lines or patches in the axes coordinate system, but this is less useful in my experience than using `ax.transAxes` for placing text. Nonetheless, here is a silly example which plots some random dots in data space, and overlays a semi-transparent Circle centered in the middle of the axes with a radius one quarter of the axes – if your axes does not preserve aspect ratio (see `set_aspect()`), this will look like an ellipse. Use the pan/zoom tool to move around, or manually change the data `xlim` and `ylim`, and you will see the data move, but the circle will remain fixed because it is not in data coordinates and will always remain at the center of the axes.

```
import matplotlib.patches as patches

fig, ax = plt.subplots()

x, y = 10*np.random.rand(2, 1000)
ax.plot(x, y, 'go') # plot some data in data coordinates

circ = patches.Circle((0.5, 0.5), 0.25, transform=ax.transAxes,
                      facecolor='yellow', alpha=0.5)
ax.add_patch(circ)
plt.show()
```



Blended transformations

Drawing in blended coordinate spaces which mix axes with data coordinates is extremely useful, for example to create a horizontal span which highlights some region of the y-data but spans across the x-axis regardless of the data limits, pan or zoom level, etc. In fact these blended lines and spans are so useful, we have built in functions to make them easy to plot (see `axhline()`, `axvline()`, `axhspan()`, `axvspan()`) but for didactic purposes we will implement the horizontal span here using a blended transformation. This trick only works for separable transformations, like you see in normal Cartesian coordinate systems, but not on inseparable transformations like the `PolarTransform`.

```
import matplotlib.transforms as transforms

fig, ax = plt.subplots()
x = np.random.randn(1000)

ax.hist(x, 30)
ax.set_title(r'$\sigma=1 \vee \dots \vee \sigma=2$', fontsize=16)

# the x coords of this transformation are data, and the
# y coord are axes
```

(continues on next page)

(continued from previous page)

```

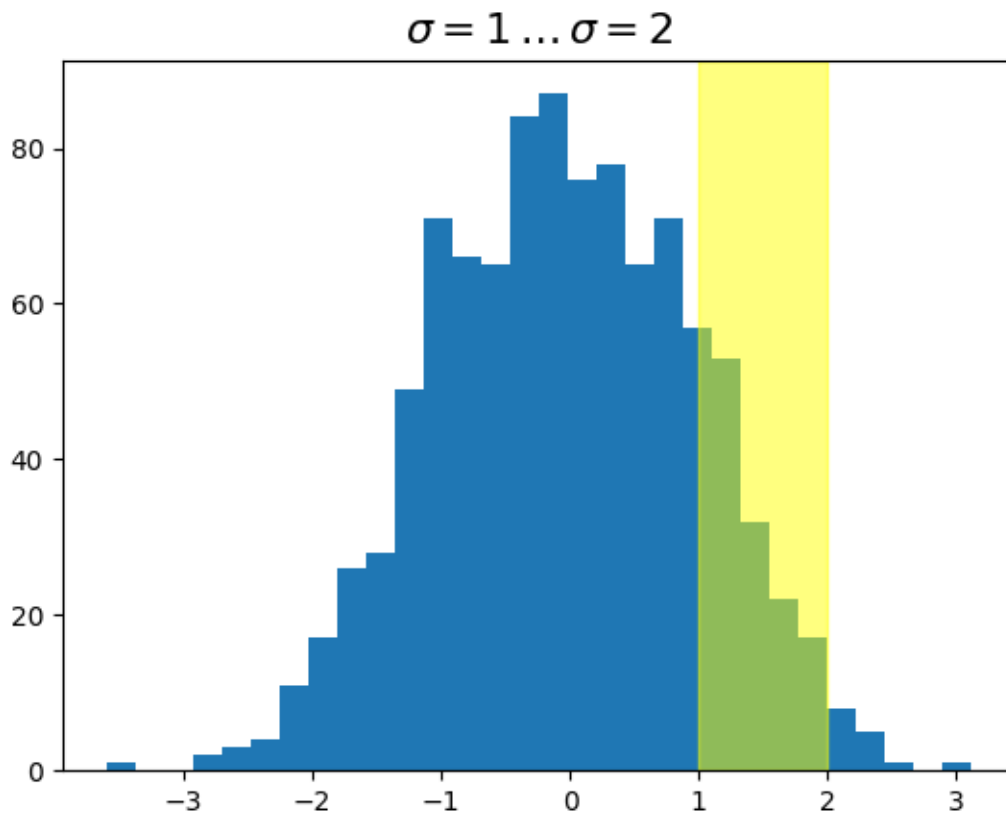
trans = transforms.blended_transform_factory(
    ax.transData, ax.transAxes)

# highlight the 1.2 stddev region with a span.
# We want x to be in data coordinates and y to
# span from 0..1 in axes coords
rect = patches.Rectangle((1, 0), width=1, height=1,
                        transform=trans, color='yellow',
                        alpha=0.5)

ax.add_patch(rect)

plt.show()

```



Note: The blended transformations where x is in data coords and y in axes coordinates is so useful that we have helper methods to return the versions mpl uses internally for drawing ticks, ticklabels, etc. The methods are `matplotlib.axes.Axes.get_xaxis_transform()` and `matplotlib.axes.Axes.get_yaxis_transform()`. So in the example above, the call to `blended_transform_factory()` can be replaced by `get_xaxis_transform`:


```
trans = ax.get_xaxis_transform()
```

Using offset transforms to create a shadow effect

One use of transformations is to create a new transformation that is offset from another transformation, e.g., to place one object shifted a bit relative to another object. Typically you want the shift to be in some physical dimension, like points or inches rather than in data coordinates, so that the shift effect is constant at different zoom levels and dpi settings.

One use for an offset is to create a shadow effect, where you draw one object identical to the first just to the right of it, and just below it, adjusting the zorder to make sure the shadow is drawn first and then the object it is shadowing above it. The transforms module has a helper transformation `ScaledTranslation`. It is instantiated with:

```
trans = ScaledTranslation(xt, yt, scale_trans)
```

where `xt` and `yt` are the translation offsets, and `scale_trans` is a transformation which scales `xt` and `yt` at transformation time before applying the offsets. A typical use case is to use the figure `fig.dpi_scale_trans` transformation for the `scale_trans` argument, to first scale `xt` and `yt` specified in points to display space before doing the final offset. The dpi and inches offset is a common-enough use case that we have a special helper function to create it in `matplotlib.transforms.offset_copy()`, which returns a new transform with an added offset. But in the example below, we'll create the offset transform ourselves. Note the use of the plus operator in:

```
offset = transforms.ScaledTranslation(dx, dy,  
    fig.dpi_scale_trans)  
shadow_transform = ax.transData + offset
```

showing that can chain transformations using the addition operator. This code says: first apply the data transformation `ax.transData` and then translate the data by `dx` and `dy` points. In typography, a ‘point’ https://en.wikipedia.org/wiki/Point_%28typography%29 is 1/72 inches, and by specifying your offsets in points, your figure will look the same regardless of the dpi resolution it is saved in.

```
fig = plt.figure()  
ax = fig.add_subplot(111)  
  
# make a simple sine wave  
x = np.arange(0., 2., 0.01)  
y = np.sin(2*np.pi*x)  
line, = ax.plot(x, y, lw=3, color='blue')  
  
# shift the object over 2 points, and down 2 points  
dx, dy = 2/72., -2/72.  
offset = transforms.ScaledTranslation(dx, dy, fig.dpi_scale_trans)
```

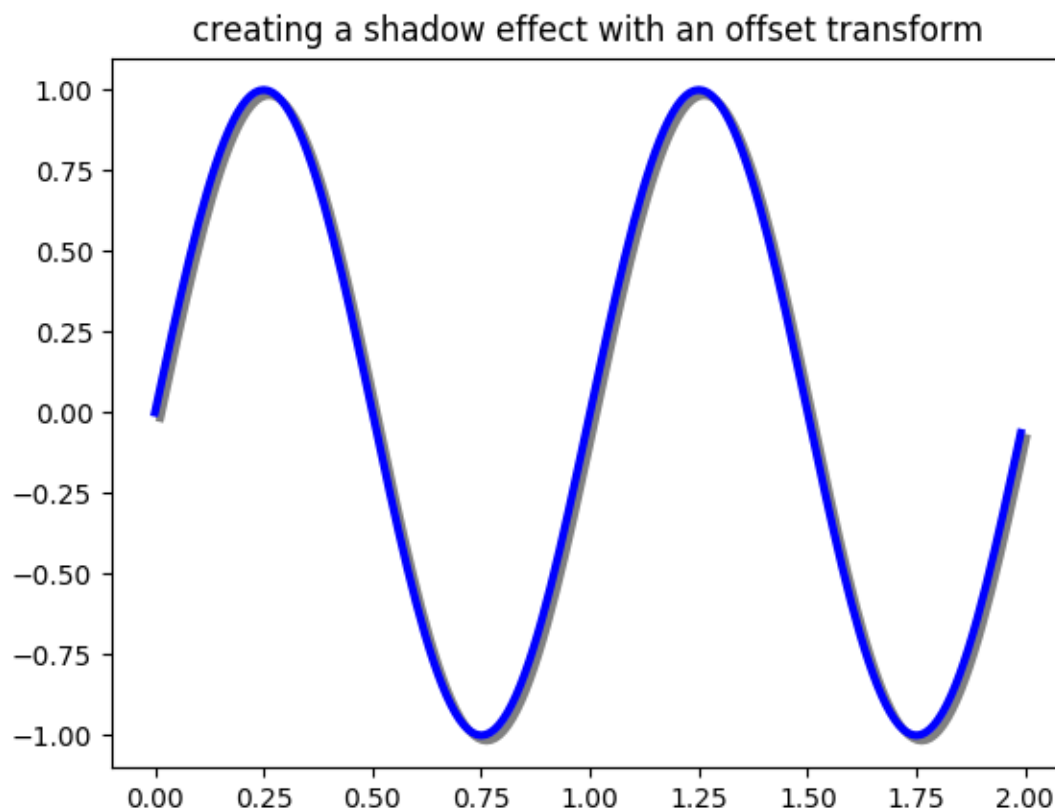
(continues on next page)

(continued from previous page)

```
shadow_transform = ax.transData + offset

# now plot the same data with our offset transform;
# use the zorder to make sure we are below the line
ax.plot(x, y, lw=3, color='gray',
        transform=shadow_transform,
        zorder=0.5*line.get_zorder())

ax.set_title('creating a shadow effect with an offset transform')
plt.show()
```



The transformation pipeline

The `ax.transData` transform we have been working with in this tutorial is a composite of three different transformations that comprise the transformation pipeline from data -> display coordinates. Michael Droettboom implemented the transformations framework, taking care to provide a clean API that segregated the nonlinear projections and scales that happen in polar and logarithmic plots, from the linear affine transformations that happen when you pan and zoom. There is an efficiency here, because you can pan and zoom in your axes which affects the affine transformation,

but you may not need to compute the potentially expensive nonlinear scales or projections on simple navigation events. It is also possible to multiply affine transformation matrices together, and then apply them to coordinates in one step. This is not true of all possible transformations.

Here is how the `ax.transData` instance is defined in the basic separable axis `Axes` class:

```
self.transData = self.transScale + (self.transLimits + self.transAxes)
```

We've been introduced to the `transAxes` instance above in [Axes coordinates](#), which maps the (0, 0), (1, 1) corners of the axes or subplot bounding box to display space, so let's look at these other two pieces.

`self.transLimits` is the transformation that takes you from data to axes coordinates; i.e., it maps your view `xlim` and `ylim` to the unit space of the axes (and `transAxes` then takes that unit space to display space). We can see this in action here

```
In [80]: ax = subplot(111)

In [81]: ax.set_xlim(0, 10)
Out[81]: (0, 10)

In [82]: ax.set_ylim(-1, 1)
Out[82]: (-1, 1)

In [84]: ax.transLimits.transform((0, -1))
Out[84]: array([ 0.,  0.])

In [85]: ax.transLimits.transform((10, -1))
Out[85]: array([ 1.,  0.])

In [86]: ax.transLimits.transform((10, 1))
Out[86]: array([ 1.,  1.])

In [87]: ax.transLimits.transform((5, 0))
Out[87]: array([ 0.5,  0.5])
```

and we can use this same inverted transformation to go from the unit axes coordinates back to data coordinates.

```
In [90]: inv.transform((0.25, 0.25))
Out[90]: array([ 2.5, -0.5])
```

The final piece is the `self.transScale` attribute, which is responsible for the optional non-linear scaling of the data, e.g., for logarithmic axes. When an `Axes` is initially setup, this is just set to the identity transform, since the basic Matplotlib axes has linear scale, but when you call a logarithmic scaling function like `semilogx()` or explicitly set the scale to logarithmic with `set_xscale()`, then the `ax.transScale` attribute is set to handle the nonlinear projection. The scales transforms are properties of the respective `xaxis` and `yaxis` `Axis` instances. For example, when you call `ax.set_xscale('log')`, the `xaxis` updates its scale to a `matplotlib.scale.LogScale` instance.

For non-separable axes the PolarAxes, there is one more piece to consider, the projection transformation. The `transData` `matplotlib.projections.polar.PolarAxes` is similar to that for the typical separable `matplotlib.Axes`, with one additional piece `transProjection`:

```
self.transData = self.transScale + self.transProjection + \
    (self.transProjectionAffine + self.transAxes)
```

`transProjection` handles the projection from the space, e.g., latitude and longitude for map data, or radius and theta for polar data, to a separable Cartesian coordinate system. There are several projection examples in the `matplotlib.projections` package, and the best way to learn more is to open the source for those packages and see how to make your own, since Matplotlib supports extensible axes and projections. Michael Droettboom has provided a nice tutorial example of creating a Hammer projection axes; see /gallery/misc/custom_projection.

3.4 Colors

Matplotlib has support for visualizing information with a wide array of colors and colormaps. These tutorials cover the basics of how these colormaps look, how you can create your own, and how you can customize colormaps for your use case.

For even more information see the [examples page](#).

Note: [Click here to download the full example code](#)

3.4.1 Specifying Colors

Matplotlib recognizes the following formats to specify a color:

- an RGB or RGBA tuple of float values in $[0, 1]$ (e.g., `(0.1, 0.2, 0.5)` or `(0.1, 0.2, 0.5, 0.3)`). RGBA is short for Red, Green, Blue, Alpha;
- a hex RGB or RGBA string (e.g., `'#0F0F0F'` or `'#0F0F0F0F'`);
- a string representation of a float value in $[0, 1]$ inclusive for gray level (e.g., `'0.5'`);
- one of `{'b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'}`;
- a X11/CSS4 color name;
- a name from the [xkcd color survey](#); prefixed with `'xkcd:'` (e.g., `'xkcd:sky blue'`);
- one of `{'tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown', 'tab:pink', 'tab:gray', 'tab:olive', 'tab:cyan'}` which are the Tableau Colors from the 'T10' categorical palette (which is the default color cycle);
- a "CN" color spec, i.e. `'C'` followed by a single digit, which is an index into the default property cycle (`matplotlib.rcParams['axes.prop_cycle']`); the indexing occurs at artist creation time and defaults to black if the cycle does not include color.

"Red", "Green" and "Blue", are the intensities of those colors, the combination of which span the colorspace.

How "Alpha" behaves depends on the zorder of the Artist. Higher zorder Artists are drawn on top of lower Artists, and "Alpha" determines whether the lower artist is covered by the higher. If the old RGB of a pixel is RGBold and the RGB of the pixel of the Artist being added is RGBnew with Alpha alpha, then the RGB of the pixel is updated to: $RGB = RGBold * (1 - Alpha) + RGBnew * Alpha$. Alpha of 1 means the old color is completely covered by the new Artist, Alpha of 0 means that pixel of the Artist is transparent.

All string specifications of color, other than "CN", are case-insensitive.

"CN" color selection

"CN" colors are converted to RGBA as soon as the artist is created. For example,

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl

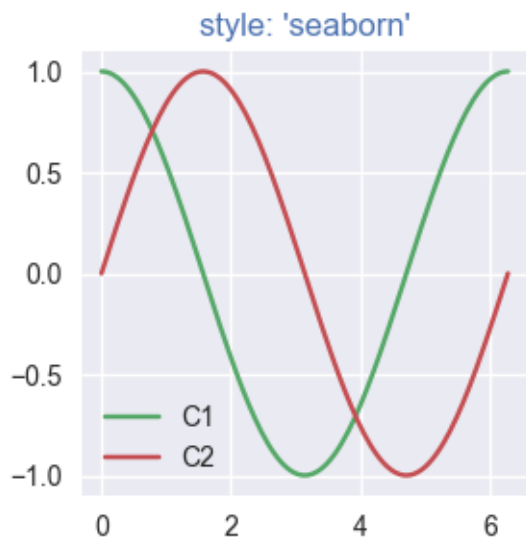
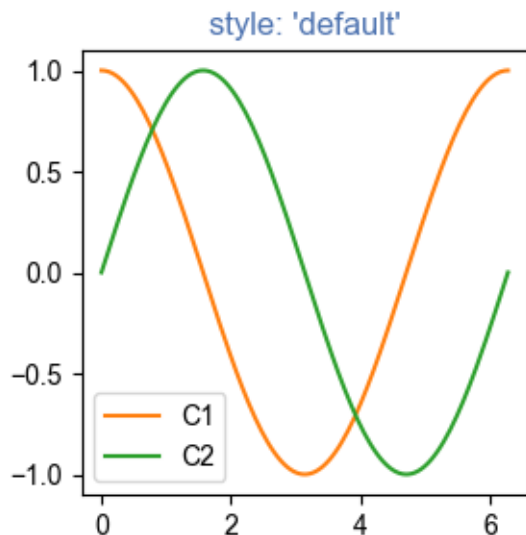
th = np.linspace(0, 2*np.pi, 128)

def demo(sty):
    mpl.style.use(sty)
    fig, ax = plt.subplots(figsize=(3, 3))

    ax.set_title('style: {!r}'.format(sty), color='C0')

    ax.plot(th, np.cos(th), 'C1', label='C1')
    ax.plot(th, np.sin(th), 'C2', label='C2')
    ax.legend()

demo('default')
demo('seaborn')
```



will use the first color for the title and then plot using the second and third colors of each style's `mpl.rcParams['axes.prop_cycle']`.

xkcd v X11/CSS4

The xkcd colors are derived from a user survey conducted by the webcomic xkcd. [Details of the survey are available on the xkcd blog.](#)

Out of 148 colors in the CSS color list, there are 95 name collisions between the X11/CSS4 names and the xkcd names, all but 3 of which have different hex values. For example 'blue' maps to '#0000FF' where as 'xkcd:blue' maps to '#0343DF'. Due to these name collisions all of the xkcd colors have 'xkcd:' prefixed. As noted in the blog post, while it might be interesting to re-define the X11/CSS4 names based on such a survey, we do not do so unilaterally.

The name collisions are shown in the table below; the color names where the hex values agree are

shown in bold.

```
import matplotlib._color_data as mcd
import matplotlib.patches as mpatch

overlap = {name for name in mcd.CSS4_COLORS
            if "xkcd:" + name in mcd.XKCD_COLORS}

fig = plt.figure(figsize=[4.8, 16])
ax = fig.add_axes([0, 0, 1, 1])

for j, n in enumerate(sorted(overlap, reverse=True)):
    weight = None
    cn = mcd.CSS4_COLORS[n]
    xkcd = mcd.XKCD_COLORS["xkcd:" + n].upper()
    if cn == xkcd:
        weight = 'bold'

    r1 = mpatch.Rectangle((0, j), 1, 1, color=cn)
    r2 = mpatch.Rectangle((1, j), 1, 1, color=xkcd)
    txt = ax.text(2, j+.5, ' ' + n, va='center', fontsize=10,
                  weight=weight)
    ax.add_patch(r1)
    ax.add_patch(r2)
    ax.axhline(j, color='k')

ax.text(.5, j + 1.5, 'X11', ha='center', va='center')
ax.text(1.5, j + 1.5, 'xkcd', ha='center', va='center')
ax.set_xlim(0, 3)
ax.set_ylim(0, j + 2)
ax.axis('off')
```

X11	xkcd	
		aqua
		aquamarine
		azure
		beige
		black
		blue
		brown
		chartreuse
		chocolate
		coral
		crimson
		cyan
		darkblue
		darkgreen
		fuchsia
		gold
		goldenrod
		green
		grey
		indigo
		ivory
		khaki
		lavender
		lightblue
		lightgreen
		lime
		magenta
		maroon
		navy
		olive

Note: [Click here to download the full example code](#)

3.4.2 Customized Colorbars Tutorial

This tutorial shows how to build colorbars without an attached plot.

Customized Colorbars

ColorbarBase derives from ScalarMappable and puts a colorbar in a specified axes, so it has everything needed for a standalone colorbar. It can be used as is to make a colorbar for a given colormap and does not need a mappable object like an image. In this tutorial we will explore what can be done with standalone colorbar.

Basic continuous colorbar

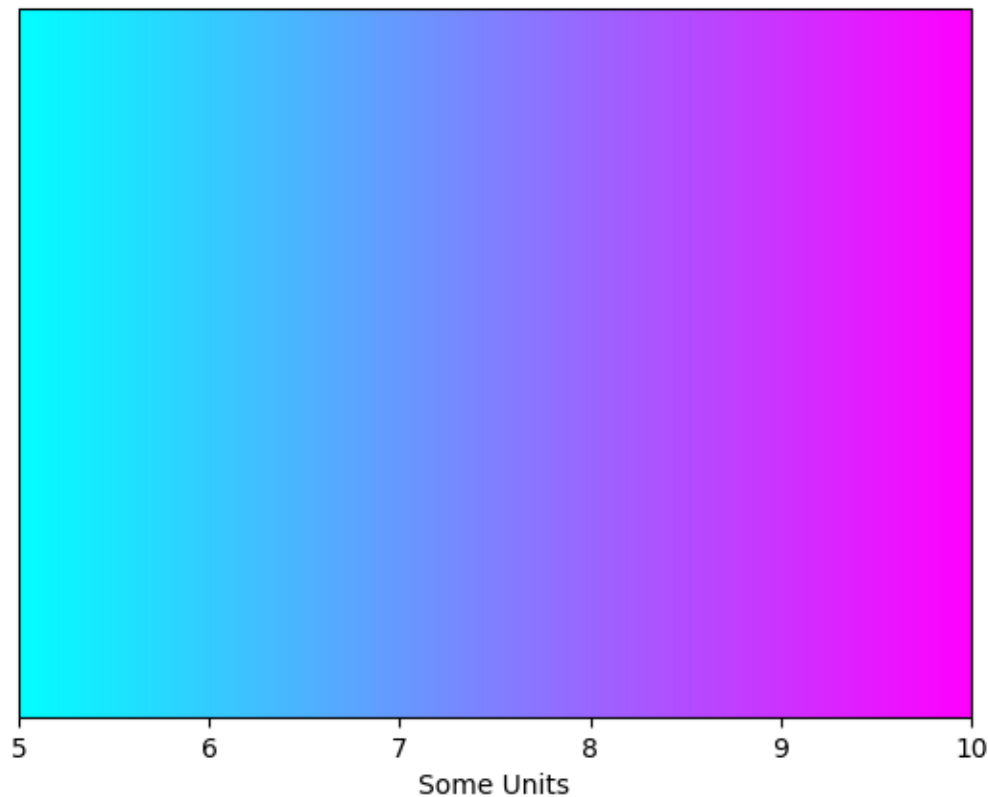
Set the colormap and norm to correspond to the data for which the colorbar will be used. Then create the colorbar by calling ColorbarBase and specify axis, colormap, norm and orientation as parameters. Here we create a basic continuous colorbar with ticks and labels. More information on colorbar api can be found [here](#).

```
import matplotlib.pyplot as plt
import matplotlib as mpl

fig, ax = plt.subplots()

cmap = mpl.cm.cool
norm = mpl.colors.Normalize(vmin=5, vmax=10)

cb1 = mpl.colorbar.ColorbarBase(ax, cmap=cmap,
                                norm=norm,
                                orientation='horizontal')
cb1.set_label('Some Units')
fig.show()
```



Discrete intervals colorbar

The second example illustrates the use of a `ListedColormap` which generates a colormap from a set of listed colors, `colors.BoundaryNorm()` which generates a colormap index based on discrete intervals and extended ends to show the "over" and "under" value colors. Over and under are used to display data outside of the normalized `[0,1]` range. Here we pass colors as gray shades as a string encoding a float in the 0-1 range.

If a `ListedColormap` is used, the length of the bounds array must be one greater than the length of the color list. The bounds must be monotonically increasing.

This time we pass some more arguments in addition to previous arguments to `ColorbarBase`. For the out-of-range values to display on the colorbar, we have to use the `extend` keyword argument. To use `extend`, you must specify two extra boundaries. Finally `spacing` argument ensures that intervals are shown on colorbar proportionally.

```
fig, ax = plt.subplots()

cmap = mpl.colors.ListedColormap(['red', 'green', 'blue', 'cyan'])
cmap.set_over('0.25')
```

(continues on next page)

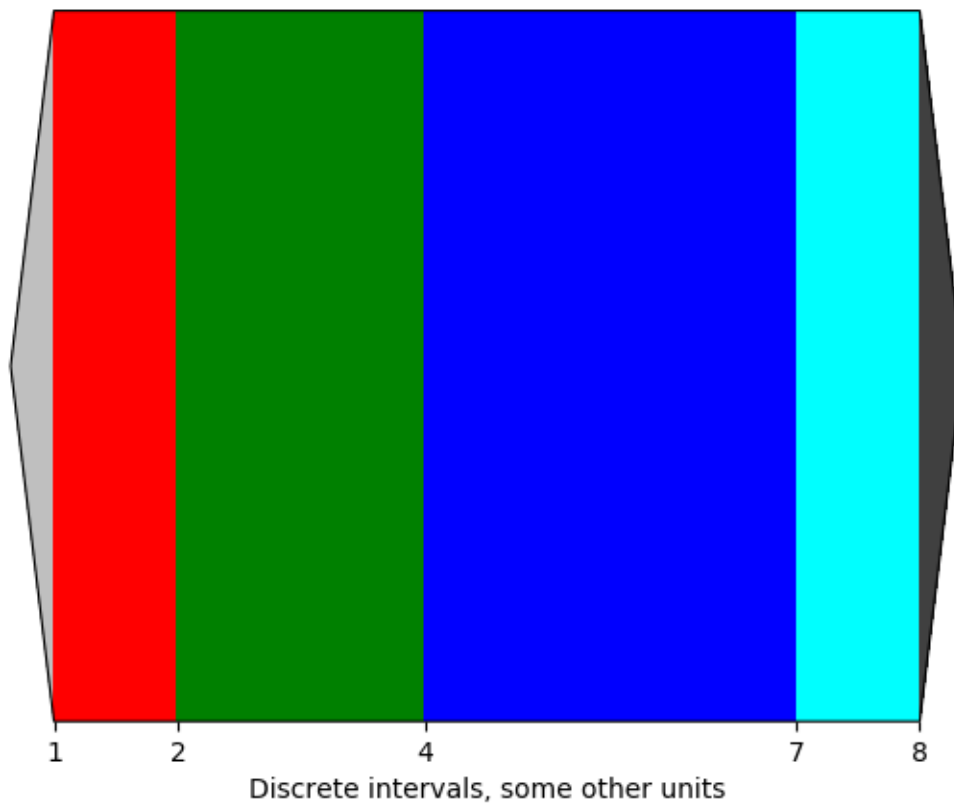
(continued from previous page)

```

cmap.set_under('0.75')

bounds = [1, 2, 4, 7, 8]
norm = mpl.colors.BoundaryNorm(bounds, cmap.N)
cb2 = mpl.colorbar.ColorbarBase(ax, cmap=cmap,
                                norm=norm,
                                boundaries=[0] + bounds + [13],
                                extend='both',
                                ticks=bounds,
                                spacing='proportional',
                                orientation='horizontal')
cb2.set_label('Discrete intervals, some other units')
fig.show()

```



Colorbar with custom extension lengths

Here we illustrate the use of custom length colorbar extensions, used on a colorbar with discrete intervals. To make the length of each extension same as the length of the interior colors, use `extendfrac='auto'`.

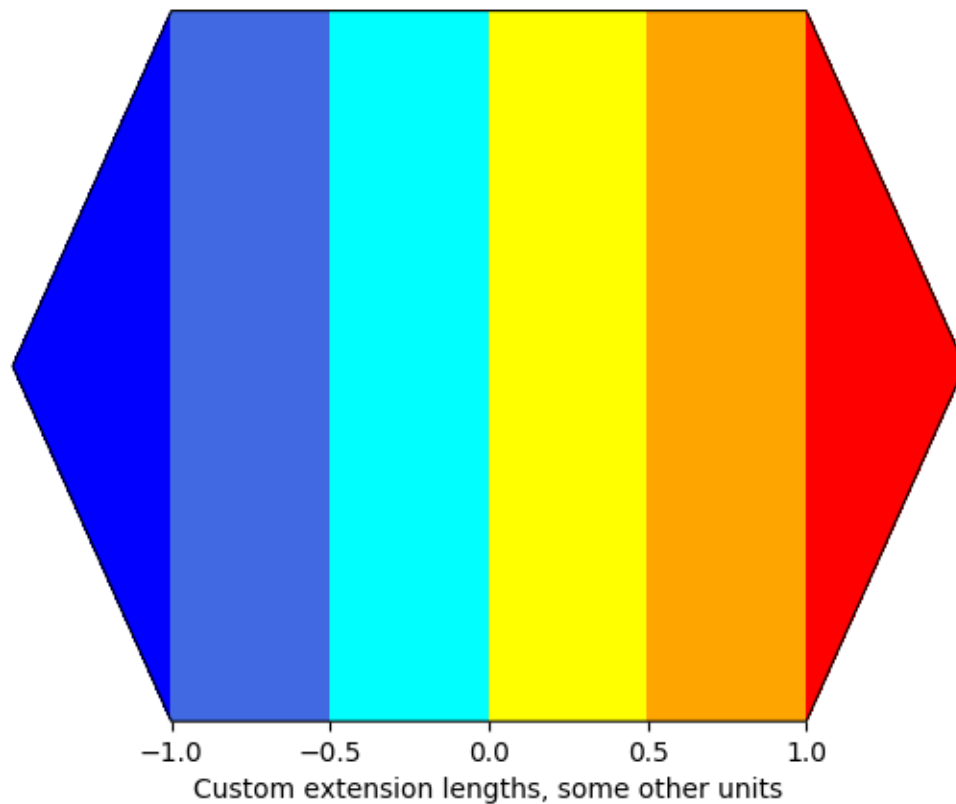
```

fig, ax = plt.subplots()

cmap = mpl.colors.ListedColormap(['royalblue', 'cyan',
                                  'yellow', 'orange'])
cmap.set_over('red')
cmap.set_under('blue')

bounds = [-1.0, -0.5, 0.0, 0.5, 1.0]
norm = mpl.colors.BoundaryNorm(bounds, cmap.N)
cb3 = mpl.colorbar.ColorbarBase(ax, cmap=cmap,
                                norm=norm,
                                boundaries=[-10] + bounds + [10],
                                extend='both',
                                extendfrac='auto',
                                ticks=bounds,
                                spacing='uniform',
                                orientation='horizontal')
cb3.set_label('Custom extension lengths, some other units')
fig.show()

```



Note: [Click here to download the full example code](#)

3.4.3 Colormap Normalization

Objects that use colormaps by default linearly map the colors in the colormap from data values `vmin` to `vmax`. For example:

```
pcm = ax.pcolormesh(x, y, Z, vmin=-1., vmax=1., cmap='RdBu_r')
```

will map the data in `Z` linearly from -1 to +1, so `Z=0` will give a color at the center of the colormap `RdBu_r` (white in this case).

Matplotlib does this mapping in two steps, with a normalization from `[0,1]` occurring first, and then mapping onto the indices in the colormap. Normalizations are classes defined in the `matplotlib.colors` module. The default, linear normalization is `matplotlib.colors.Normalize()`.

Artists that map data to color pass the arguments `vmin` and `vmax` to construct a `matplotlib.colors.Normalize()` instance, then call it:

```
In [1]: import matplotlib as mpl
In [2]: norm = mpl.colors.Normalize(vmin=-1.,vmax=1.)
In [3]: norm(0.)
Out[3]: 0.5
```

However, there are sometimes cases where it is useful to map data to colormaps in a non-linear fashion.

Logarithmic

One of the most common transformations is to plot data by taking its logarithm (to the base-10). This transformation is useful to display changes across disparate scales. Using `colors.LogNorm()` normalizes the data via \log_{10} . In the example below, there are two bumps, one much smaller than the other. Using `colors.LogNorm()`, the shape and location of each bump can clearly be seen:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as colors

N = 100
X, Y = np.mgrid[-3:3:complex(0, N), -2:2:complex(0, N)]

# A low hump with a spike coming out of the top right. Needs to have
# z/colour axis on a log scale so we see both hump and spike. linear
```

(continues on next page)

(continued from previous page)

```

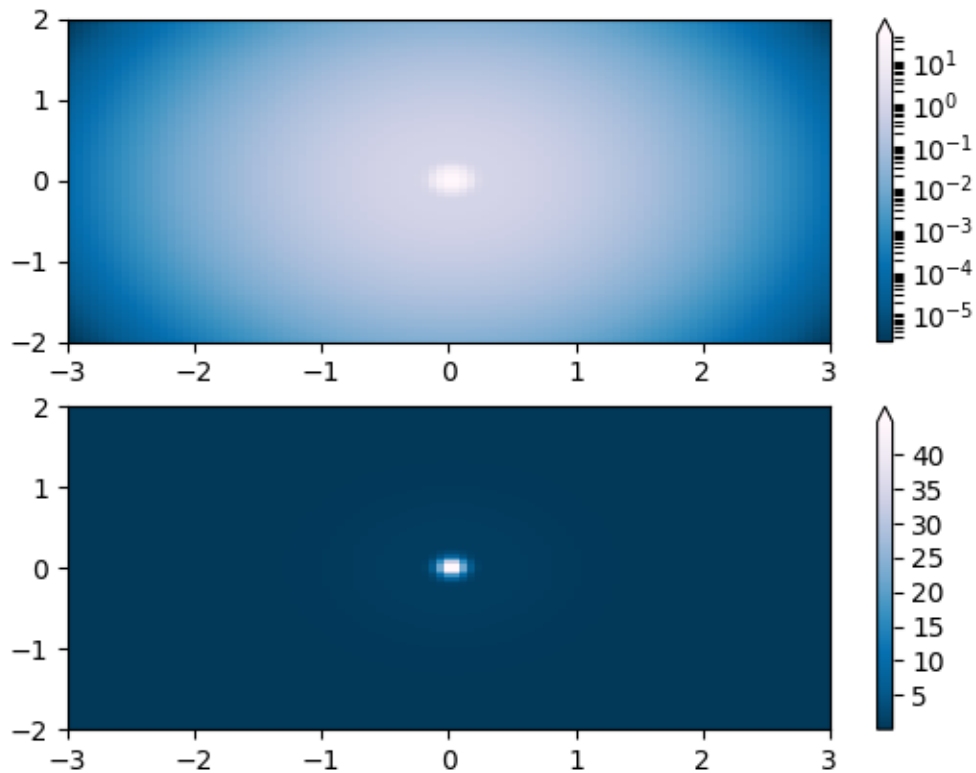
# scale only shows the spike.
Z1 = np.exp(-(X)**2 - (Y)**2)
Z2 = np.exp(-(X * 10)**2 - (Y * 10)**2)
Z = Z1 + 50 * Z2

fig, ax = plt.subplots(2, 1)

pcm = ax[0].pcolor(X, Y, Z,
                  norm=colors.LogNorm(vmin=Z.min(), vmax=Z.max()),
                  cmap='PuBu_r')
fig.colorbar(pcm, ax=ax[0], extend='max')

pcm = ax[1].pcolor(X, Y, Z, cmap='PuBu_r')
fig.colorbar(pcm, ax=ax[1], extend='max')
fig.show()

```



Symmetric logarithmic

Similarly, it sometimes happens that there is data that is positive and negative, but we would still like a logarithmic scaling applied to both. In this case, the negative numbers are also scaled logarithmically, and mapped to smaller numbers; e.g., if `vmin=-vmax`, then they the negative numbers are mapped from 0 to 0.5 and the positive from 0.5 to 1.

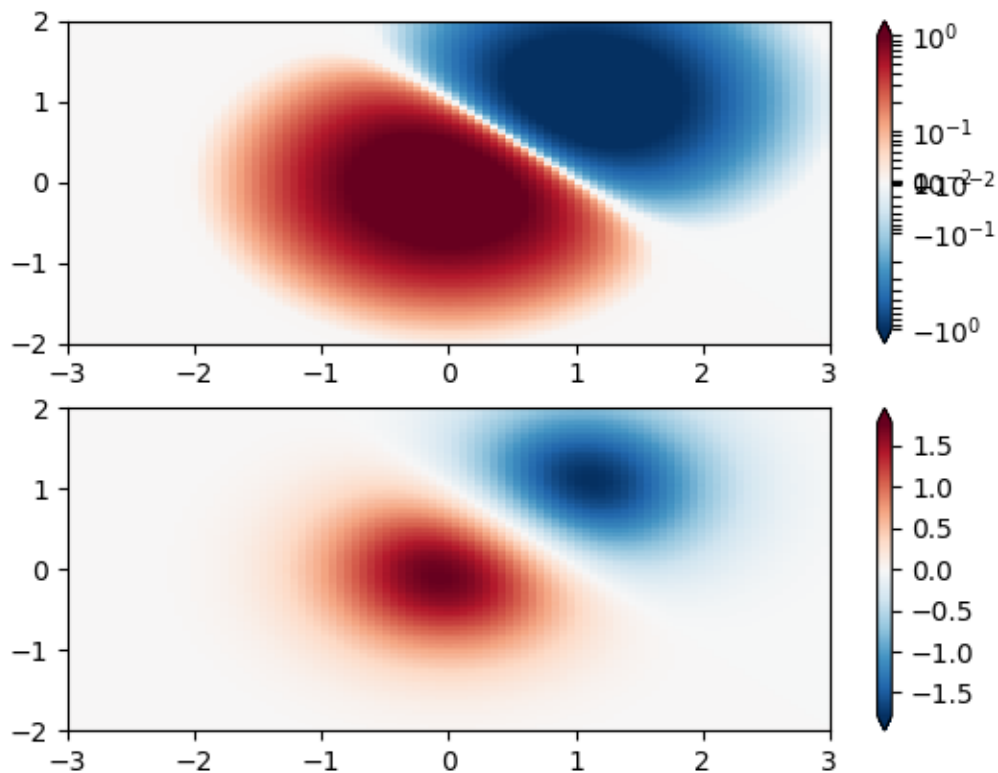
Since the logarithm of values close to zero tends toward infinity, a small range around zero needs to be mapped linearly. The parameter `linthresh` allows the user to specify the size of this range (`-linthresh, linthresh`). The size of this range in the colormap is set by `linscale`. When `linscale == 1.0` (the default), the space used for the positive and negative halves of the linear range will be equal to one decade in the logarithmic range.

```
N = 100
X, Y = np.mgrid[-3:3:complex(0, N), -2:2:complex(0, N)]
Z1 = np.exp(-X**2 - Y**2)
Z2 = np.exp(-(X - 1)**2 - (Y - 1)**2)
Z = (Z1 - Z2) * 2

fig, ax = plt.subplots(2, 1)

pcm = ax[0].pcolormesh(X, Y, Z,
                      norm=colors.SymLogNorm(linthresh=0.03, linscale=0.03,
                                              vmin=-1.0, vmax=1.0),
                      cmap='RdBu_r')
fig.colorbar(pcm, ax=ax[0], extend='both')

pcm = ax[1].pcolormesh(X, Y, Z, cmap='RdBu_r', vmin=-np.max(Z))
fig.colorbar(pcm, ax=ax[1], extend='both')
fig.show()
```



Power-law

Sometimes it is useful to remap the colors onto a power-law relationship (i.e. $y = x^\gamma$, where γ is the power). For this we use the `colors.PowerNorm()`. It takes as an argument `gamma` (`gamma == 1.0` will just yield the default linear normalization):

Note: There should probably be a good reason for plotting the data using this type of transformation. Technical viewers are used to linear and logarithmic axes and data transformations. Power laws are less common, and viewers should explicitly be made aware that they have been used.

```
N = 100
X, Y = np.mgrid[0:3:complex(0, N), 0:2:complex(0, N)]
Z1 = (1 + np.sin(Y * 10.)) * X**(2.)

fig, ax = plt.subplots(2, 1)

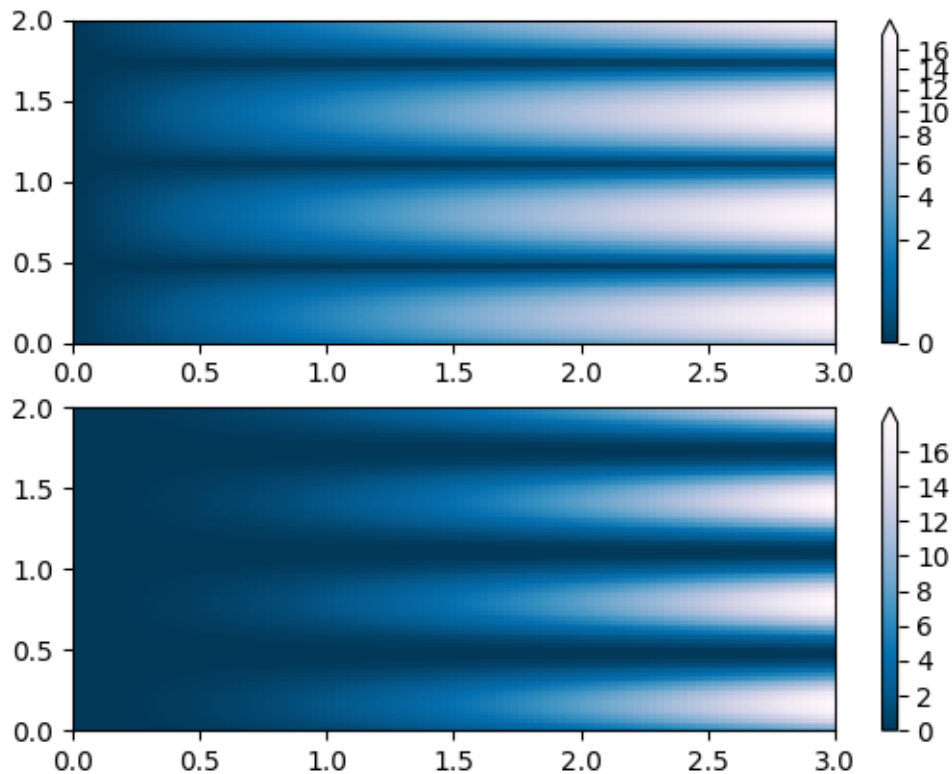
pcm = ax[0].pcolormesh(X, Y, Z1, norm=colors.PowerNorm(gamma=0.5),
                      cmap='PuBu_r')
```

(continues on next page)

(continued from previous page)

```
fig.colorbar(pcm, ax=ax[0], extend='max')

pcm = ax[1].pcolormesh(X, Y, Z1, cmap='PuBu_r')
fig.colorbar(pcm, ax=ax[1], extend='max')
fig.show()
```



Discrete bounds

Another normalization that comes with matplotlib is `colors.BoundaryNorm()`. In addition to `vmin` and `vmax`, this takes as arguments boundaries between which data is to be mapped. The colors are then linearly distributed between these "bounds". For instance:

```
In [4]: import matplotlib.colors as colors

In [5]: bounds = np.array([-0.25, -0.125, 0, 0.5, 1])

In [6]: norm = colors.BoundaryNorm(boundaries=bounds, ncolors=4)
```

(continues on next page)

(continued from previous page)

```
In [7]: print(norm([-0.2,-0.15,-0.02, 0.3, 0.8, 0.99]))
[0 0 1 2 3 3]
```

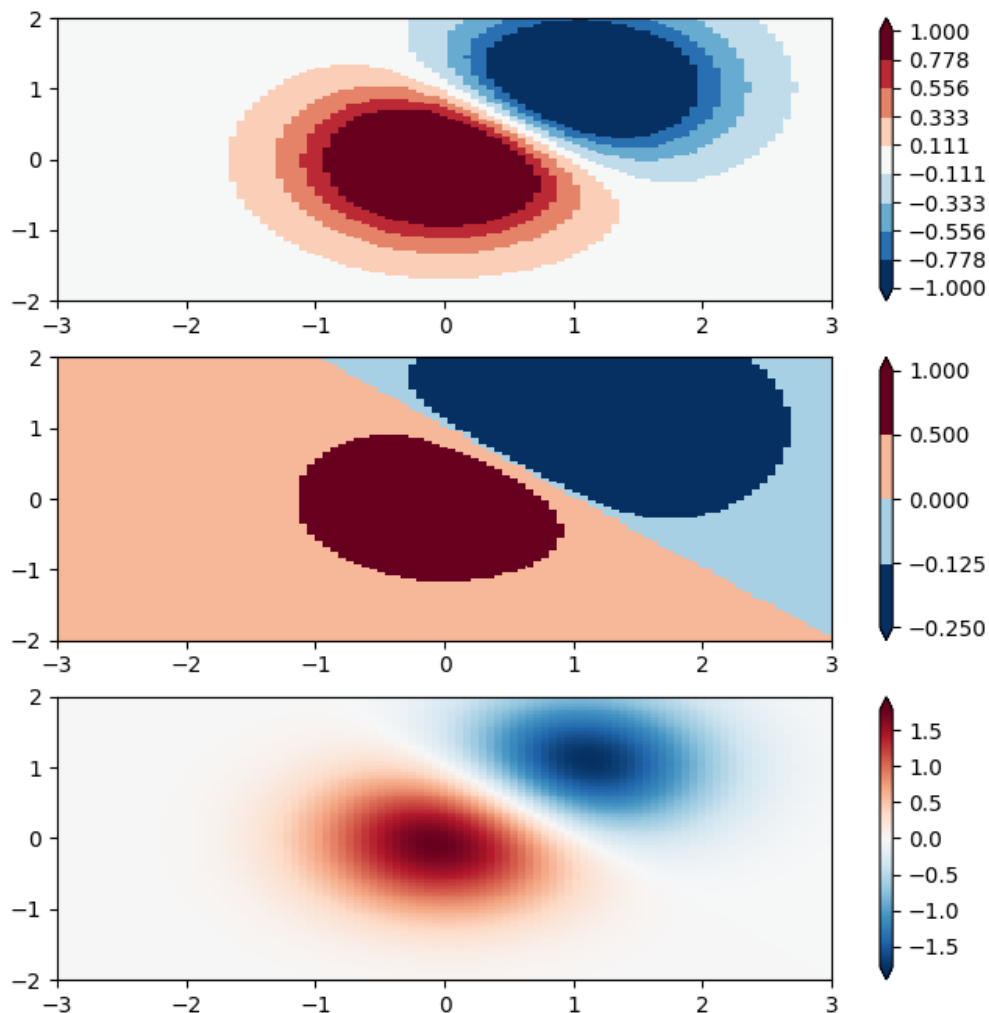
Note unlike the other norms, this norm returns values from 0 to `ncolors-1`.

```
N = 100
X, Y = np.mgrid[-3:3:complex(0, N), -2:2:complex(0, N)]
Z1 = np.exp(-X**2 - Y**2)
Z2 = np.exp(-(X - 1)**2 - (Y - 1)**2)
Z = (Z1 - Z2) * 2

fig, ax = plt.subplots(3, 1, figsize=(8, 8))
ax = ax.flatten()
# even bounds gives a contour-like effect
bounds = np.linspace(-1, 1, 10)
norm = colors.BoundaryNorm(boundaries=bounds, ncolors=256)
pcm = ax[0].pcolormesh(X, Y, Z,
                      norm=norm,
                      cmap='RdBu_r')
fig.colorbar(pcm, ax=ax[0], extend='both', orientation='vertical')

# uneven bounds changes the colormapping:
bounds = np.array([-0.25, -0.125, 0, 0.5, 1])
norm = colors.BoundaryNorm(boundaries=bounds, ncolors=256)
pcm = ax[1].pcolormesh(X, Y, Z, norm=norm, cmap='RdBu_r')
fig.colorbar(pcm, ax=ax[1], extend='both', orientation='vertical')

pcm = ax[2].pcolormesh(X, Y, Z, cmap='RdBu_r', vmin=-np.max(Z))
fig.colorbar(pcm, ax=ax[2], extend='both', orientation='vertical')
fig.show()
```



Custom normalization: Two linear ranges

It is possible to define your own normalization. In the following example, we modify `colors.SymLogNorm()` to use different linear maps for the negative data values and the positive. (Note that this example is simple, and does not validate inputs or account for complex cases such as masked data)

Note: This may appear soon as `colors.OffsetNorm()`.

As above, non-symmetric mapping of data to color is non-standard practice for quantitative data, and should only be used advisedly. A practical example is having an ocean/land colormap where

the land and ocean data span different ranges.

```
N = 100
X, Y = np.mgrid[-3:3:complex(0, N), -2:2:complex(0, N)]
Z1 = np.exp(-X**2 - Y**2)
Z2 = np.exp(-(X - 1)**2 - (Y - 1)**2)
Z = (Z1 - Z2) * 2

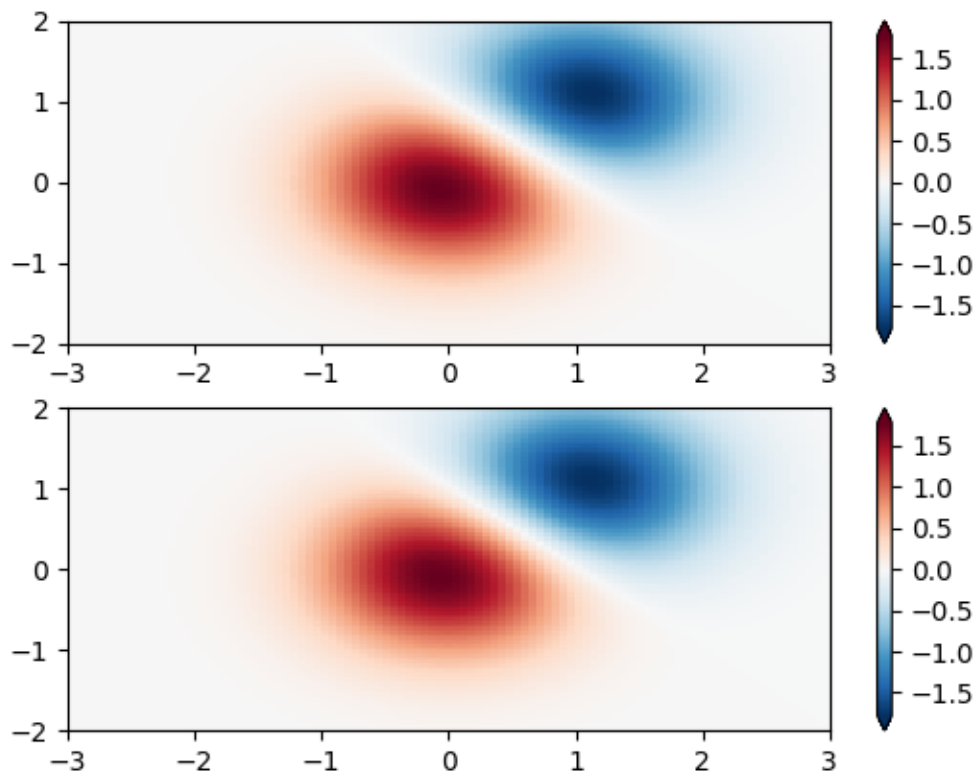
class MidpointNormalize(colors.Normalize):
    def __init__(self, vmin=None, vmax=None, midpoint=None, clip=False):
        self.midpoint = midpoint
        colors.Normalize.__init__(self, vmin, vmax, clip)

    def __call__(self, value, clip=None):
        # I'm ignoring masked values and all kinds of edge cases to make a
        # simple example...
        x, y = [self.vmin, self.midpoint, self.vmax], [0, 0.5, 1]
        return np.ma.masked_array(np.interp(value, x, y))

fig, ax = plt.subplots(2, 1)

pcm = ax[0].pcolormesh(X, Y, Z,
                      norm=MidpointNormalize(midpoint=0.),
                      cmap='RdBu_r')
fig.colorbar(pcm, ax=ax[0], extend='both')

pcm = ax[1].pcolormesh(X, Y, Z, cmap='RdBu_r', vmin=-np.max(Z))
fig.colorbar(pcm, ax=ax[1], extend='both')
fig.show()
```



Note: [Click here to download the full example code](#)

3.4.4 Colormaps in Matplotlib

How (and why) to choose a particular colormap.

Overview

The idea behind choosing a good colormap is to find a good representation in 3D colorspace for your data set. The best colormap for any given data set depends on many things including:

- Whether representing form or metric data ([?])
- Your knowledge of the data set (e.g., is there a critical value from which the other values deviate?)
- If there is an intuitive color scheme for the parameter you are plotting
- If there is a standard in the field the audience may be expecting

For many applications, a perceptually uniform colormap is the best choice — one in which equal steps in data are perceived as equal steps in the color space. Researchers have found that the human brain perceives changes in the lightness parameter as changes in the data much better than, for example, changes in hue. Therefore, colormaps which have monotonically increasing lightness through the colormap will be better interpreted by the viewer. A wonderful example of perceptually uniform colormaps is [?].

Color can be represented in 3D space in various ways. One way to represent color is using CIELAB. In CIELAB, color space is represented by lightness, L^* ; red-green, a^* ; and yellow-blue, b^* . The lightness parameter L^* can then be used to learn more about how the matplotlib colormaps will be perceived by viewers.

An excellent starting resource for learning about human perception of colormaps is from [?].

Classes of colormaps

Colormaps are often split into several categories based on their function (see, e.g., [?]):

1. Sequential: change in lightness and often saturation of color incrementally, often using a single hue; should be used for representing information that has ordering.
2. Diverging: change in lightness and possibly saturation of two different colors that meet in the middle at an unsaturated color; should be used when the information being plotted has a critical middle value, such as topography or when the data deviates around zero.
3. Qualitative: often are miscellaneous colors; should be used to represent information which does not have ordering or relationships.

```
# sphinx_gallery_thumbnail_number = 2

import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib import cm
from colorspacious import cspace_converter
from collections import OrderedDict

cmaps = OrderedDict()
```

Sequential

For the Sequential plots, the lightness value increases monotonically through the colormaps. This is good. Some of the L^* values in the colormaps span from 0 to 100 (binary and the other grayscale), and others start around $L^* = 20$. Those that have a smaller range of L^* will accordingly have a smaller perceptual range. Note also that the L^* function varies amongst the colormaps: some are approximately linear in L^* and others are more curved.

```
cmaps['Perceptually Uniform Sequential'] = [  
    'viridis', 'plasma', 'inferno', 'magma', 'cividis']  
  
cmaps['Sequential'] = [  
    'Greys', 'Purples', 'Blues', 'Greens', 'Oranges', 'Reds',  
    'YlOrBr', 'YlOrRd', 'OrRd', 'PuRd', 'RdPu', 'BuPu',  
    'GnBu', 'PuBu', 'YlGnBu', 'PuBuGn', 'BuGn', 'YlGn']
```

Sequential2

Many of the L^* values from the Sequential2 plots are monotonically increasing, but some (autumn, cool, spring, and winter) plateau or even go both up and down in L^* space. Others (afmhot, copper, gist_heat, and hot) have kinks in the L^* functions. Data that is being represented in a region of the colormap that is at a plateau or kink will lead to a perception of banding of the data in those values in the colormap (see [?] for an excellent example of this).

```
cmaps['Sequential (2)'] = [  
    'binary', 'gist_yarg', 'gist_gray', 'gray', 'bone', 'pink',  
    'spring', 'summer', 'autumn', 'winter', 'cool', 'Wistia',  
    'hot', 'afmhot', 'gist_heat', 'copper']
```

Diverging

For the Diverging maps, we want to have monotonically increasing L^* values up to a maximum, which should be close to $L^* = 100$, followed by monotonically decreasing L^* values. We are looking for approximately equal minimum L^* values at opposite ends of the colormap. By these measures, BrBG and RdBu are good options. coolwarm is a good option, but it doesn't span a wide range of L^* values (see grayscale section below).

```
cmaps['Diverging'] = [  
    'PiYG', 'PRGn', 'BrBG', 'PuOr', 'RdGy', 'RdBu',  
    'RdYlBu', 'RdYlGn', 'Spectral', 'coolwarm', 'bwr', 'seismic']
```

Qualitative

Qualitative colormaps are not aimed at being perceptual maps, but looking at the lightness parameter can verify that for us. The L^* values move all over the place throughout the colormap, and are clearly not monotonically increasing. These would not be good options for use as perceptual colormaps.

```
cmaps['Qualitative'] = ['Pastel1', 'Pastel2', 'Paired', 'Accent',  
    'Dark2', 'Set1', 'Set2', 'Set3',  
    'tab10', 'tab20', 'tab20b', 'tab20c']
```

Miscellaneous

Some of the miscellaneous colormaps have particular uses for which they have been created. For example, `gist_earth`, `ocean`, and `terrain` all seem to be created for plotting topography (green/brown) and water depths (blue) together. We would expect to see a divergence in these colormaps, then, but multiple kinks may not be ideal, such as in `gist_earth` and `terrain`. `CMRmap` was created to convert well to grayscale, though it does appear to have some small kinks in L^* . `cubehelix` was created to vary smoothly in both lightness and hue, but appears to have a small hump in the green hue area.

The often-used `jet` colormap is included in this set of colormaps. We can see that the L^* values vary widely throughout the colormap, making it a poor choice for representing data for viewers to see perceptually. See an extension on this idea at [?].

```
cmaps['Miscellaneous'] = [
    'flag', 'prism', 'ocean', 'gist_earth', 'terrain', 'gist_stern',
    'gnuplot', 'gnuplot2', 'CMRmap', 'cubehelix', 'brg', 'hsv',
    'gist_rainbow', 'rainbow', 'jet', 'nipy_spectral', 'gist_ncar']
```

First, we'll show the range of each colormap. Note that some seem to change more "quickly" than others.

```
nrows = max(len(cmap_list) for cmap_category, cmap_list in cmaps.items())
gradient = np.linspace(0, 1, 256)
gradient = np.vstack((gradient, gradient))

def plot_color_gradients(cmap_category, cmap_list, nrows):
    fig, axes = plt.subplots(nrows=nrows)
    fig.subplots_adjust(top=0.95, bottom=0.01, left=0.2, right=0.99)
    axes[0].set_title(cmap_category + ' colormaps', fontsize=14)

    for ax, name in zip(axes, cmap_list):
        ax.imshow(gradient, aspect='auto', cmap=plt.get_cmap(name))
        pos = list(ax.get_position().bounds)
        x_text = pos[0] - 0.01
        y_text = pos[1] + pos[3]/2.
        fig.text(x_text, y_text, name, va='center', ha='right', fontsize=10)

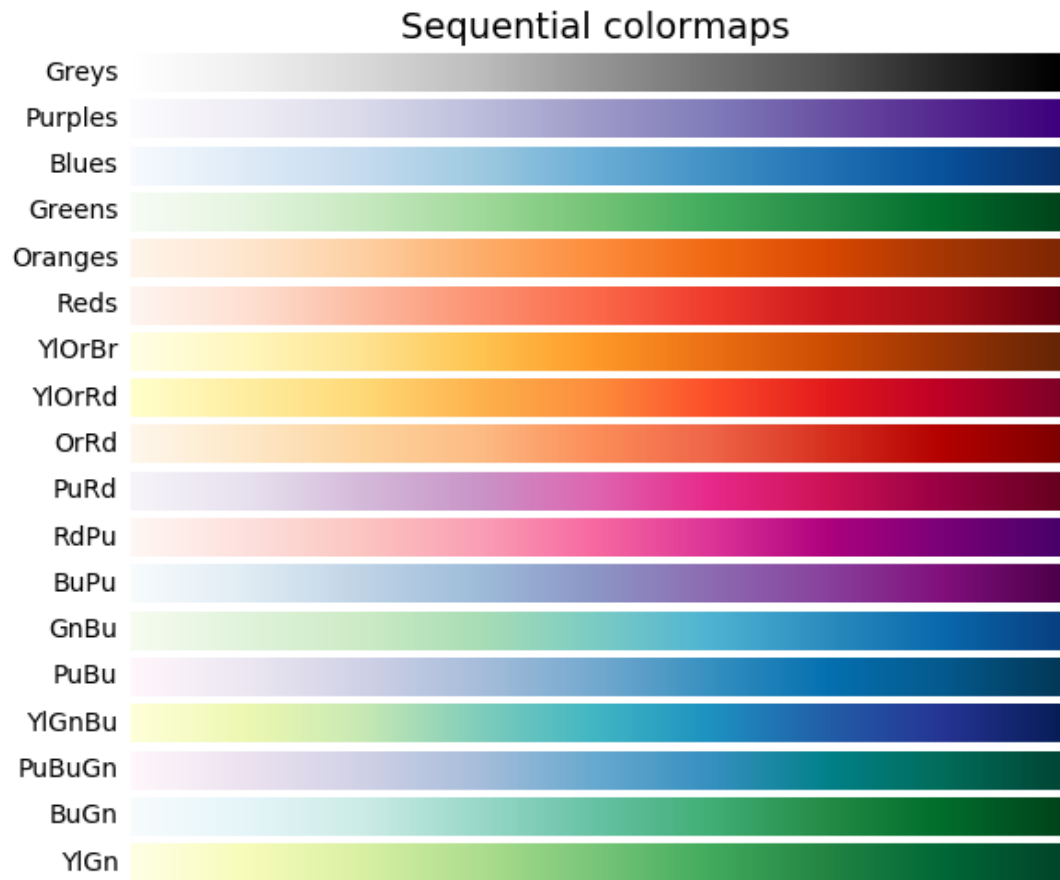
    # Turn off *all* ticks & spines, not just the ones with colormaps.
    for ax in axes:
        ax.set_axis_off()

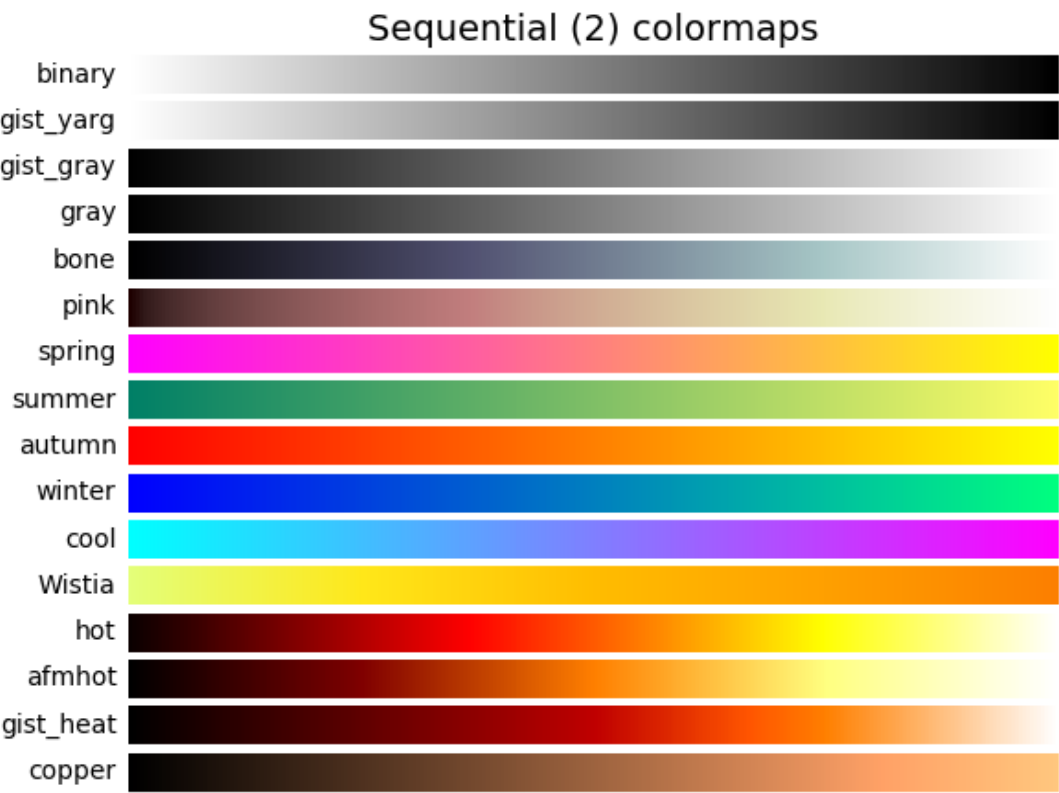
for cmap_category, cmap_list in cmaps.items():
    plot_color_gradients(cmap_category, cmap_list, nrows)

plt.show()
```

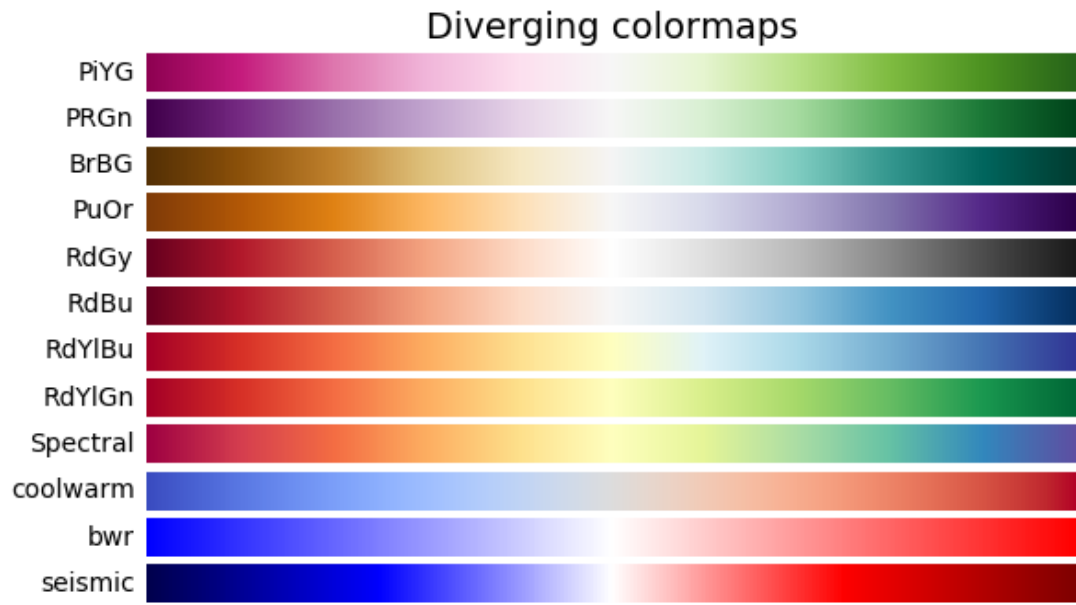


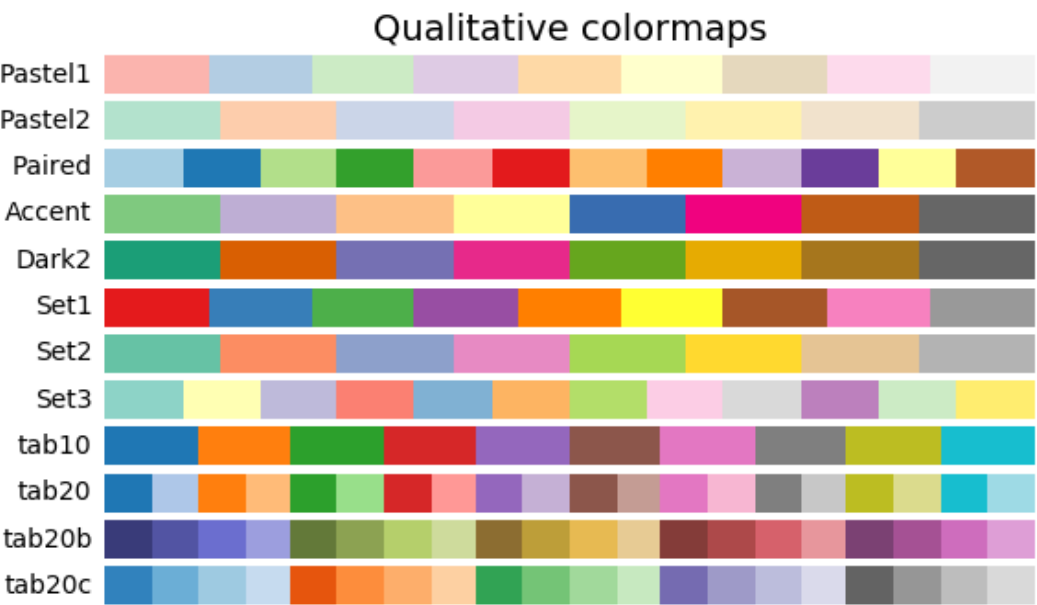

•

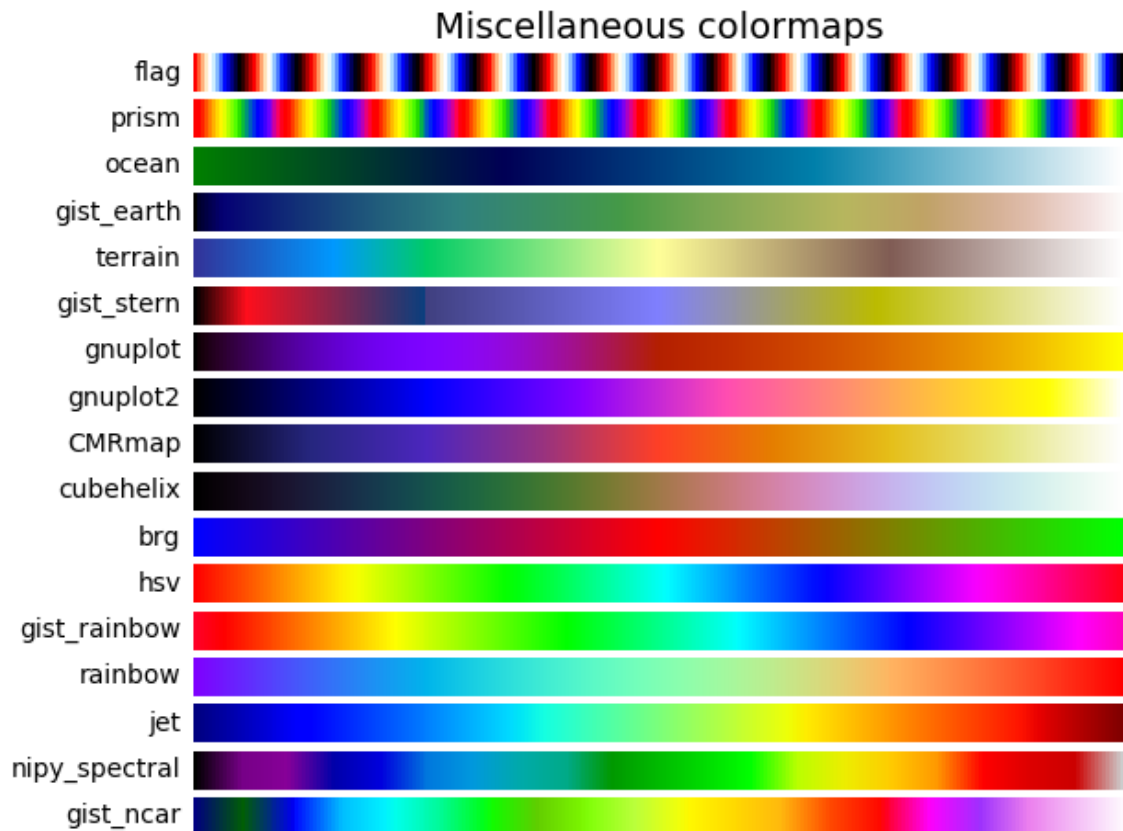




•







•

Lightness of matplotlib colormaps

Here we examine the lightness values of the matplotlib colormaps. Note that some documentation on the colormaps is available ([\[?\]](#)).

```
mpl.rcParams.update({'font.size': 12})

# Number of colormap per subplot for particular cmap categories
_DSUBS = {'Perceptually Uniform Sequential': 5, 'Sequential': 6,
          'Sequential (2)': 6, 'Diverging': 6, 'Qualitative': 4,
          'Miscellaneous': 6}

# Spacing between the colormaps of a subplot
_DC = {'Perceptually Uniform Sequential': 1.4, 'Sequential': 0.7,
       'Sequential (2)': 1.4, 'Diverging': 1.4, 'Qualitative': 1.4,
       'Miscellaneous': 1.4}

# Indices to step through colormap
x = np.linspace(0.0, 1.0, 100)

# Do plot
```

(continues on next page)

(continued from previous page)

```

for cmap_category, cmap_list in cmaps.items():

    # Do subplots so that colormaps have enough space.
    # Default is 6 colormaps per subplot.
    dsub = _DSUBS.get(cmap_category, 6)
    nsubplots = int(np.ceil(len(cmap_list) / dsub))

    # squeeze=False to handle similarly the case of a single subplot
    fig, axes = plt.subplots(nrows=nsubplots, squeeze=False,
                             figsize=(7, 2.6*nsubplots))

    for i, ax in enumerate(axes.flat):

        locs = [] # locations for text labels

        for j, cmap in enumerate(cmap_list[i*dsub:(i+1)*dsub]):

            # Get RGB values for colormap and convert the colormap in
            # CAM02-UCS colorspace. lab[0, :, 0] is the lightness.
            rgb = cm.get_cmap(cmap)(x)[np.newaxis, :, :3]
            lab = cspace_converter("sRGB1", "CAM02-UCS")(rgb)

            # Plot colormap L values. Do separately for each category
            # so each plot can be pretty. To make scatter markers change
            # color along plot:
            # http://stackoverflow.com/questions/8202605/matplotlib-scatterplot-colour-as-a-function-of-a-third-variable

            if cmap_category == 'Sequential':
                # These colormaps all start at high lightness but we want them
                # reversed to look nice in the plot, so reverse the order.
                y_ = lab[0, ::-1, 0]
                c_ = x[::-1]
            else:
                y_ = lab[0, :, 0]
                c_ = x

            dc = _DC.get(cmap_category, 1.4) # cmaps horizontal spacing
            ax.scatter(x + j*dc, y_, c=c_, cmap=cmap, s=300, linewidths=0.0)

            # Store locations for colormap labels
            if cmap_category in ('Perceptually Uniform Sequential',
                                'Sequential'):
                locs.append(x[-1] + j*dc)
            elif cmap_category in ('Diverging', 'Qualitative',

```

(continues on next page)

(continued from previous page)

```

'Miscellaneous', 'Sequential (2)'):
    locs.append(x[int(x.size/2.)] + j*dc)

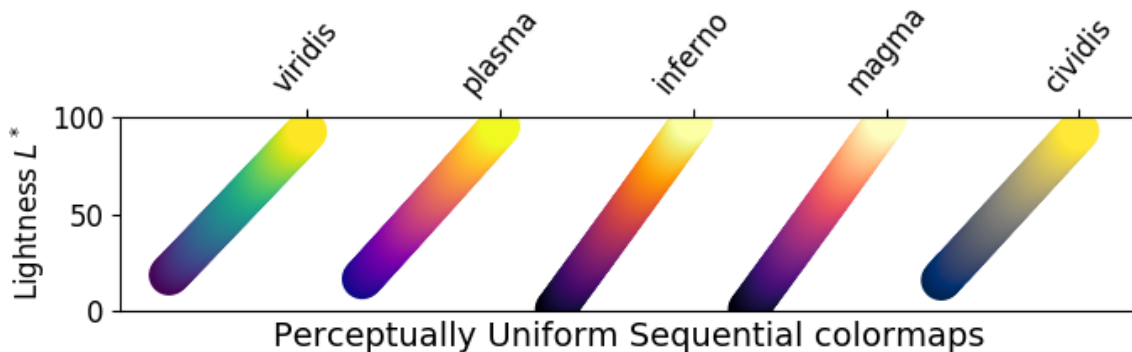
# Set up the axis limits:
# * the 1st subplot is used as a reference for the x-axis limits
# * lightness values goes from 0 to 100 (y-axis limits)
ax.set_xlim(axes[0, 0].get_xlim())
ax.set_ylim(0.0, 100.0)

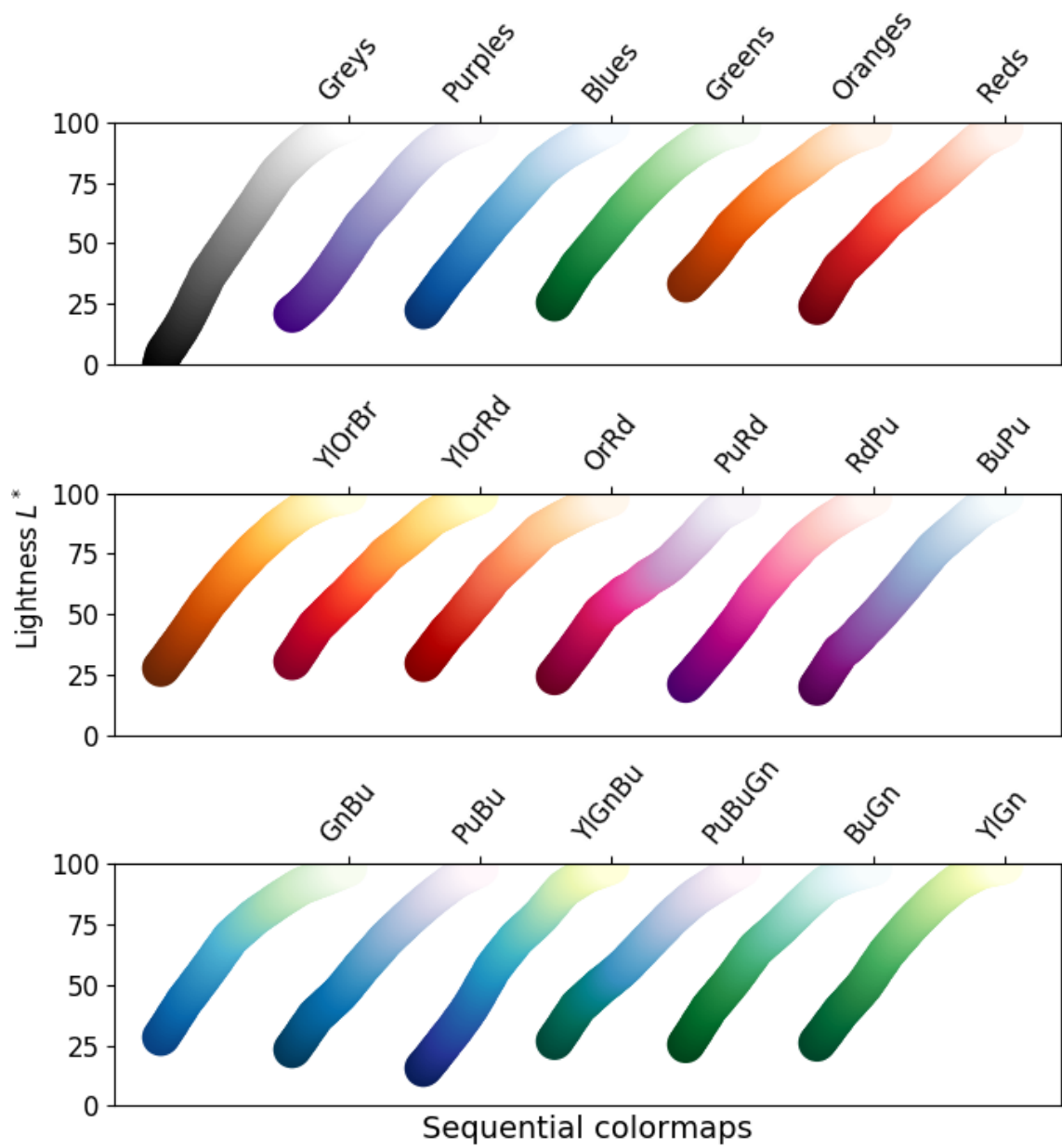
# Set up labels for colormaps
ax.xaxis.set_ticks_position('top')
ticker = mpl.ticker.FixedLocator(locs)
ax.xaxis.set_major_locator(ticker)
formatter = mpl.ticker.FixedFormatter(cmap_list[i*dsub:(i+1)*dsub])
ax.xaxis.set_major_formatter(formatter)
ax.xaxis.set_tick_params(rotation=50)

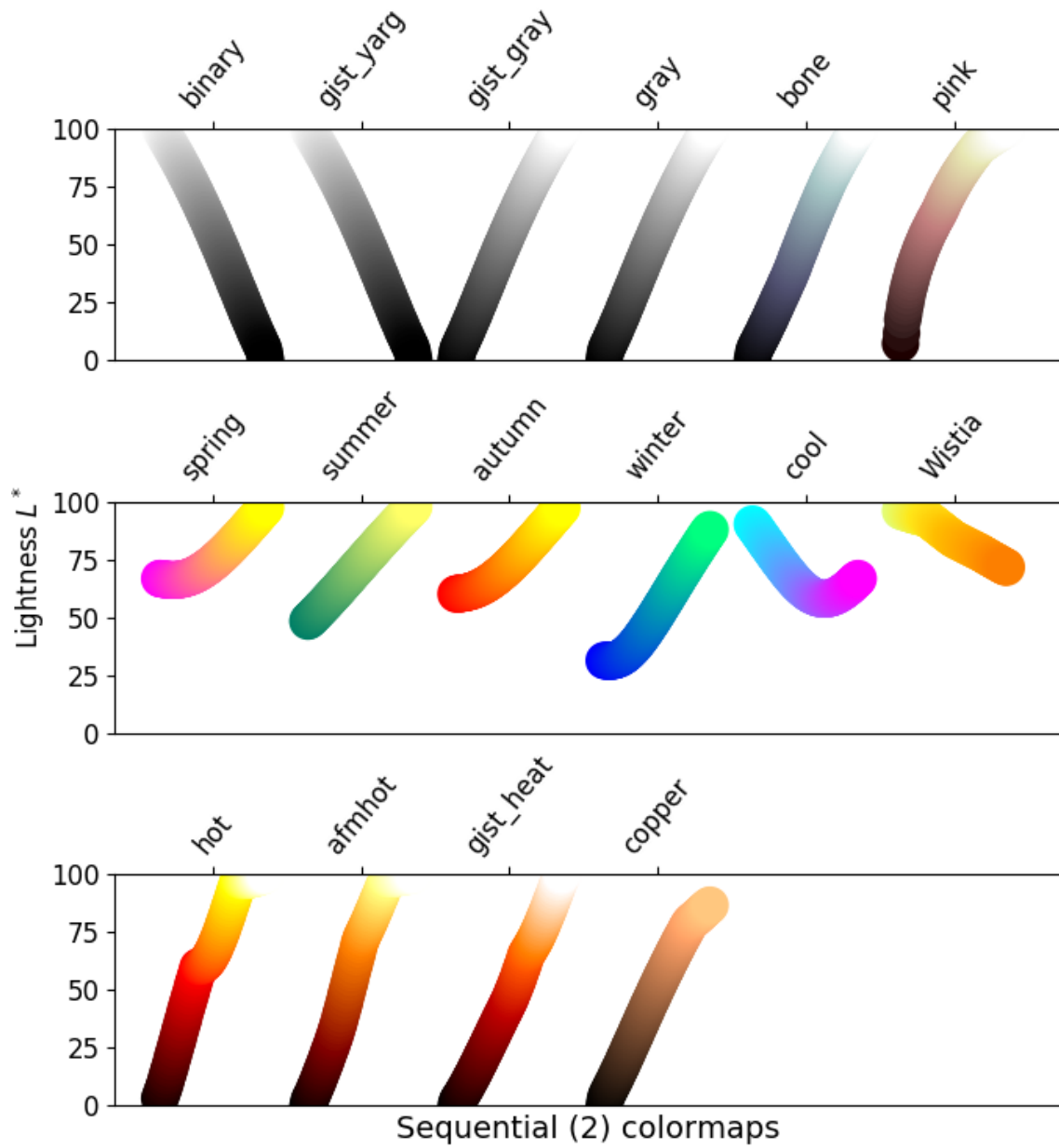
ax.set_xlabel(cmap_category + ' colormaps', fontsize=14)
fig.text(0.0, 0.55, 'Lightness  $L^*$ ', fontsize=12,
        transform=fig.transFigure, rotation=90)

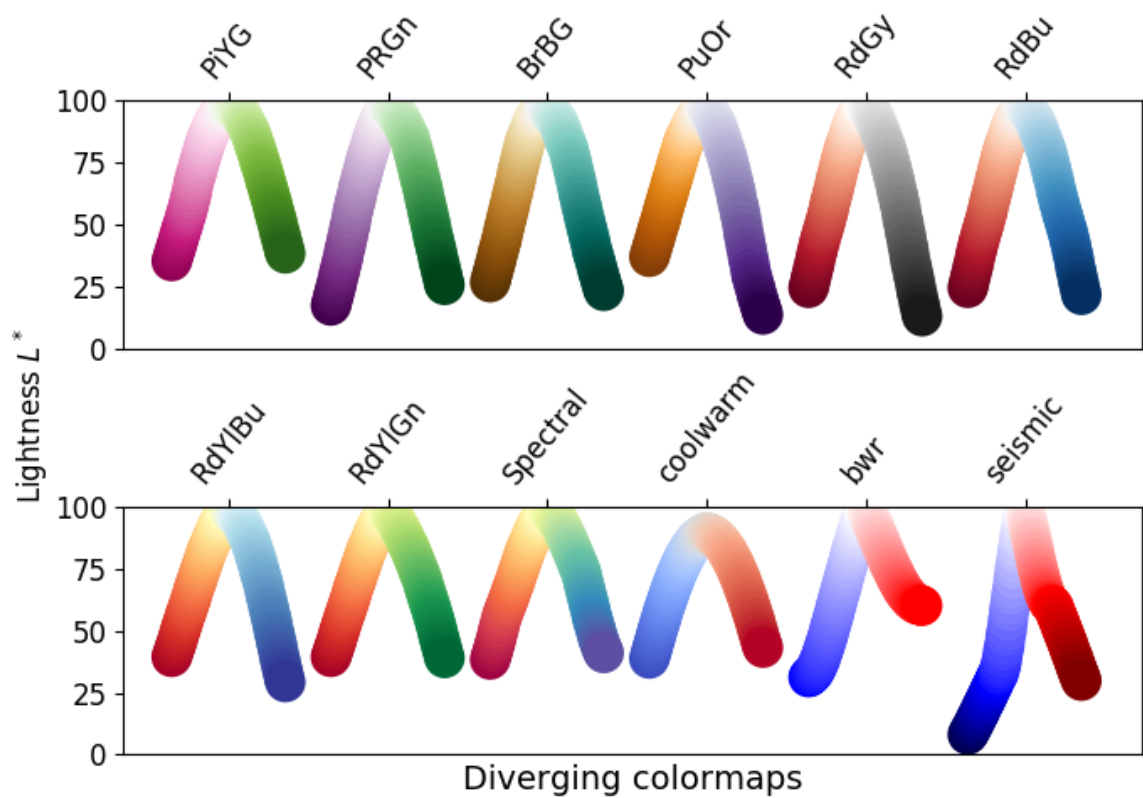
fig.tight_layout(h_pad=0.0, pad=1.5)
plt.show()

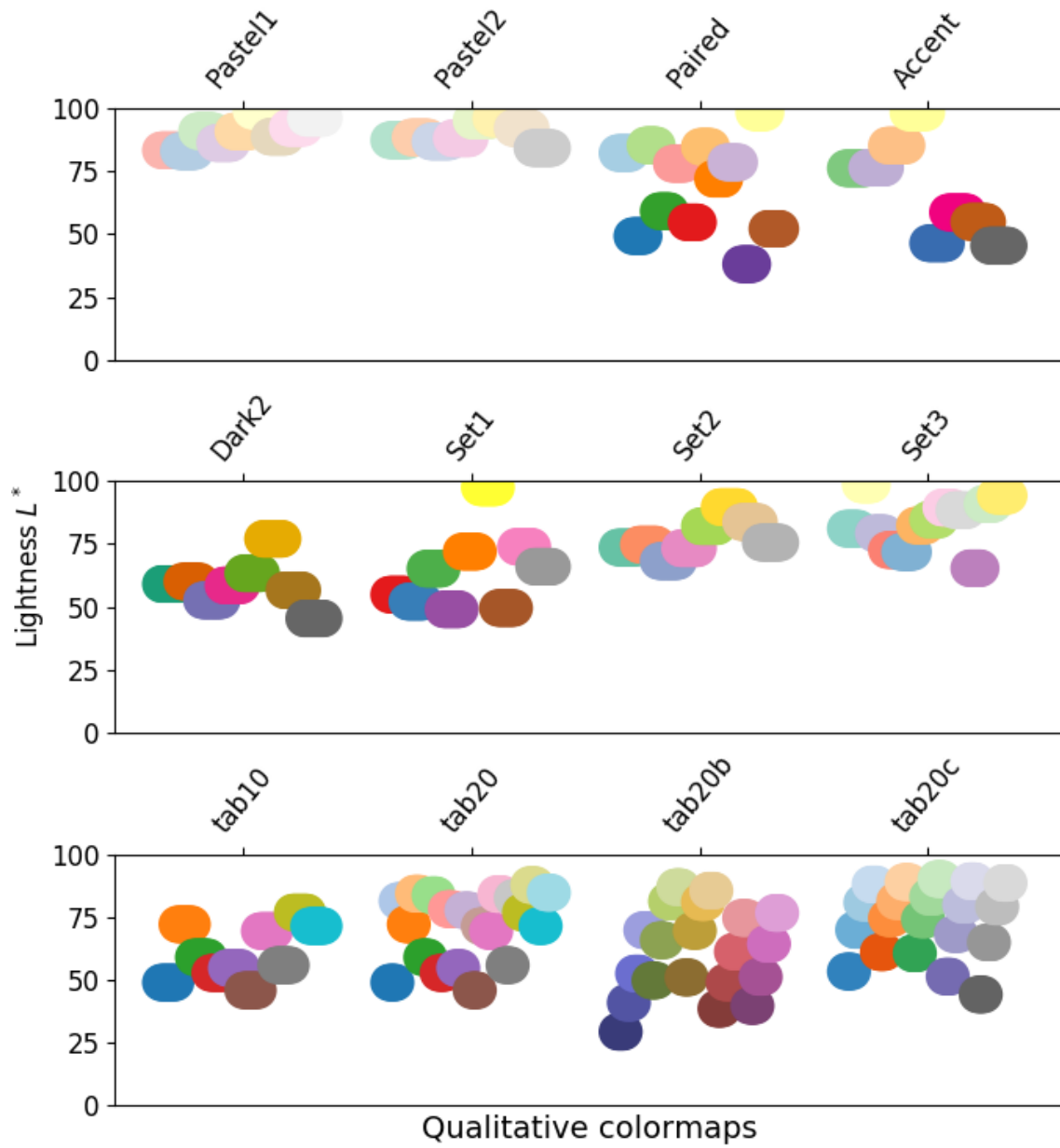
```

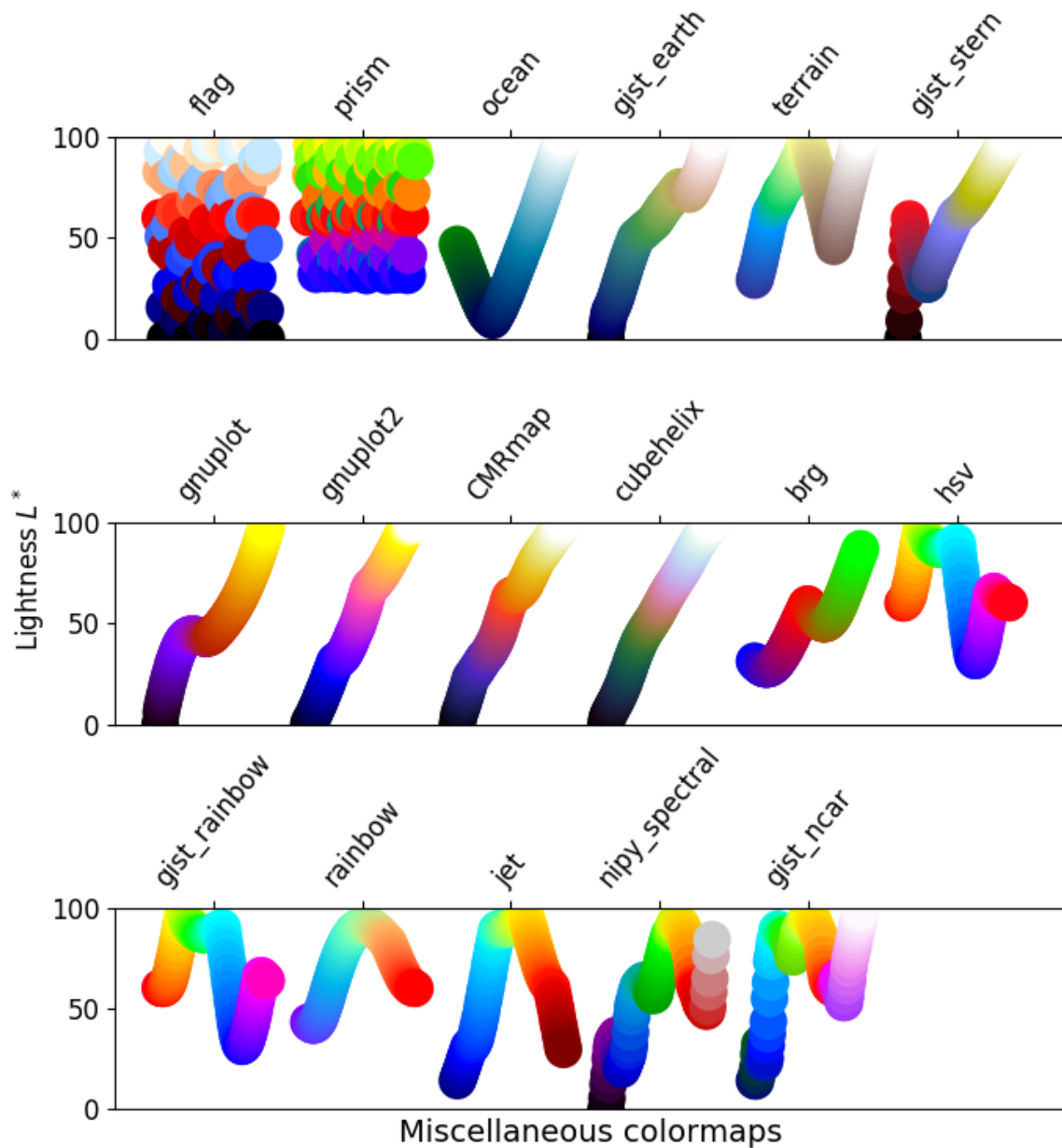












Grayscale conversion

It is important to pay attention to conversion to grayscale for color plots, since they may be printed on black and white printers. If not carefully considered, your readers may end up with indecipherable plots because the grayscale changes unpredictably through the colormap.

Conversion to grayscale is done in many different ways [?]. Some of the better ones use a linear combination of the rgb values of a pixel, but weighted according to how we perceive color intensity. A nonlinear method of conversion to grayscale is to use the L^* values of the pixels. In general, similar principles apply for this question as they do for presenting one's information perceptually;

that is, if a colormap is chosen that is monotonically increasing in L^* values, it will print in a reasonable manner to grayscale.

With this in mind, we see that the Sequential colormaps have reasonable representations in grayscale. Some of the Sequential2 colormaps have decent enough grayscale representations, though some (autumn, spring, summer, winter) have very little grayscale change. If a colormap like this was used in a plot and then the plot was printed to grayscale, a lot of the information may map to the same gray values. The Diverging colormaps mostly vary from darker gray on the outer edges to white in the middle. Some (PuOr and seismic) have noticeably darker gray on one side than the other and therefore are not very symmetric. coolwarm has little range of gray scale and would print to a more uniform plot, losing a lot of detail. Note that overlaid, labeled contours could help differentiate between one side of the colormap vs. the other since color cannot be used once a plot is printed to grayscale. Many of the Qualitative and Miscellaneous colormaps, such as Accent, hsv, and jet, change from darker to lighter and back to darker gray throughout the colormap. This would make it impossible for a viewer to interpret the information in a plot once it is printed in grayscale.

```
mpl.rcParams.update({'font.size': 14})

# Indices to step through colormap.
x = np.linspace(0.0, 1.0, 100)

gradient = np.linspace(0, 1, 256)
gradient = np.vstack((gradient, gradient))

def plot_color_gradients(cmap_category, cmap_list):
    fig, axes = plt.subplots(nrows=len(cmap_list), ncols=2)
    fig.subplots_adjust(top=0.95, bottom=0.01, left=0.2, right=0.99,
                        wspace=0.05)
    fig.suptitle(cmap_category + ' colormaps', fontsize=14, y=1.0, x=0.6)

    for ax, name in zip(axes, cmap_list):

        # Get RGB values for colormap.
        rgb = cm.get_cmap(plt.get_cmap(name))(x)[np.newaxis, :, :3]

        # Get colormap in CAM02-UCS colorspace. We want the lightness.
        lab = cspace_converter("sRGB1", "CAM02-UCS")(rgb)
        L = lab[0, :, 0]
        L = np.float32(np.vstack((L, L, L)))

        ax[0].imshow(gradient, aspect='auto', cmap=plt.get_cmap(name))
        ax[1].imshow(L, aspect='auto', cmap='binary_r', vmin=0., vmax=100.)
        pos = list(ax[0].get_position().bounds)
        x_text = pos[0] - 0.01
        y_text = pos[1] + pos[3]/2.
```

(continues on next page)

(continued from previous page)

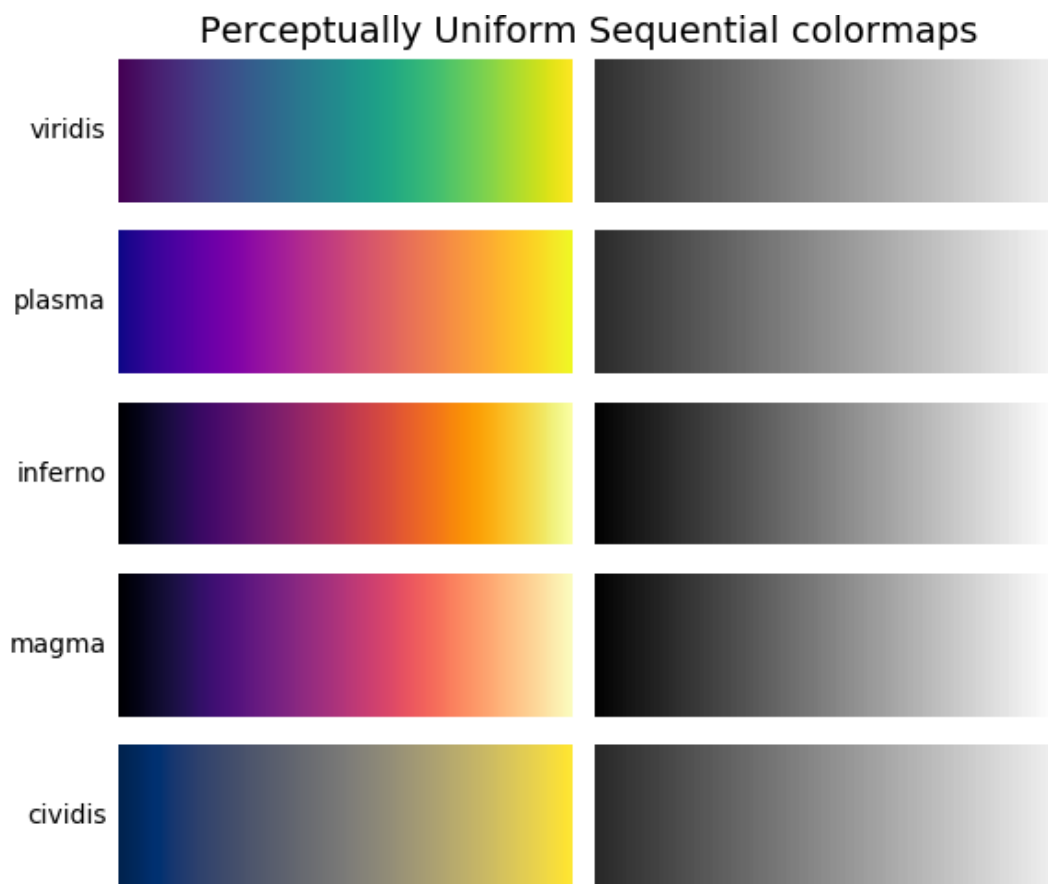
```
fig.text(x_text, y_text, name, va='center', ha='right', fontsize=10)

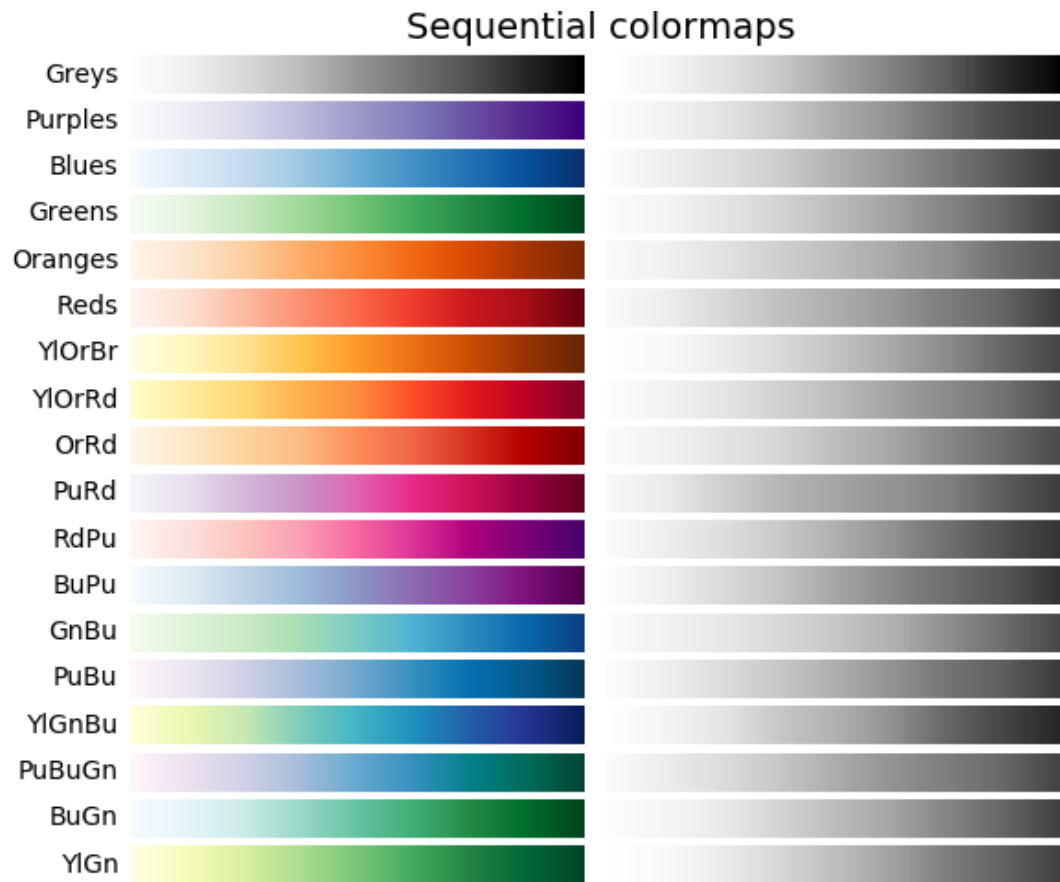
# Turn off *all* ticks & spines, not just the ones with colormaps.
for ax in axes.flat:
    ax.set_axis_off()

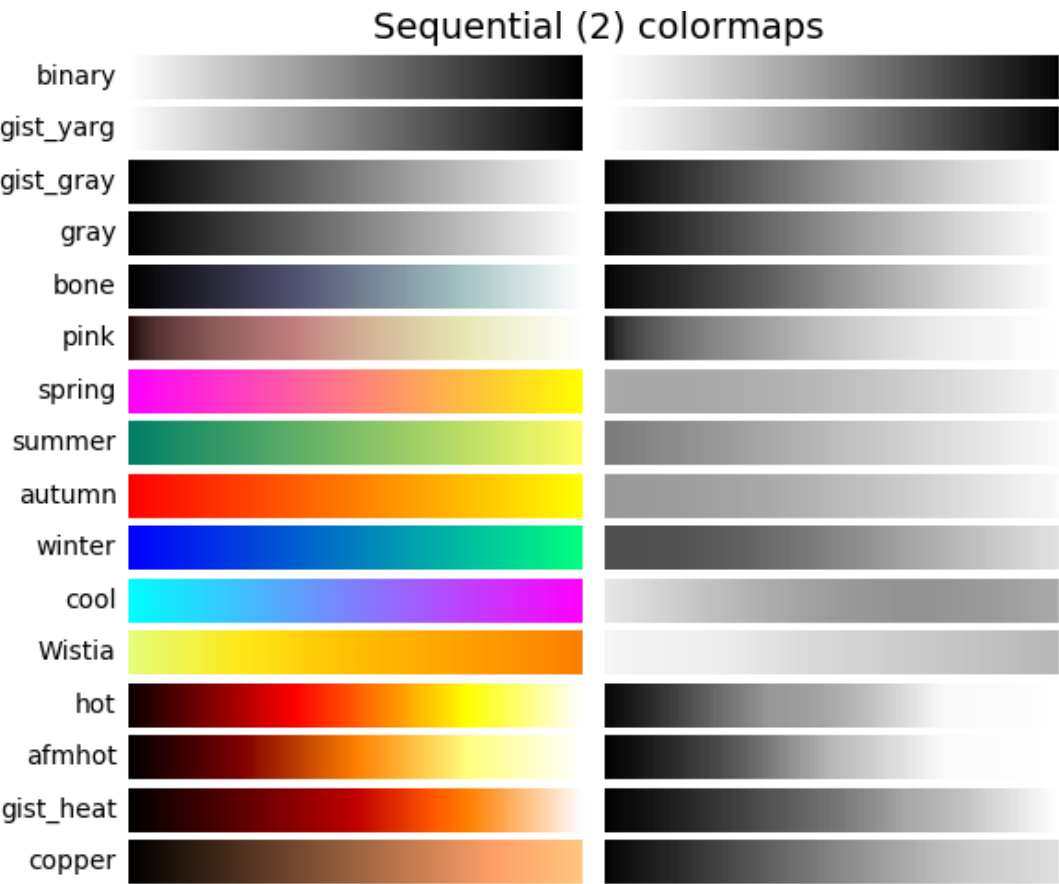
plt.show()

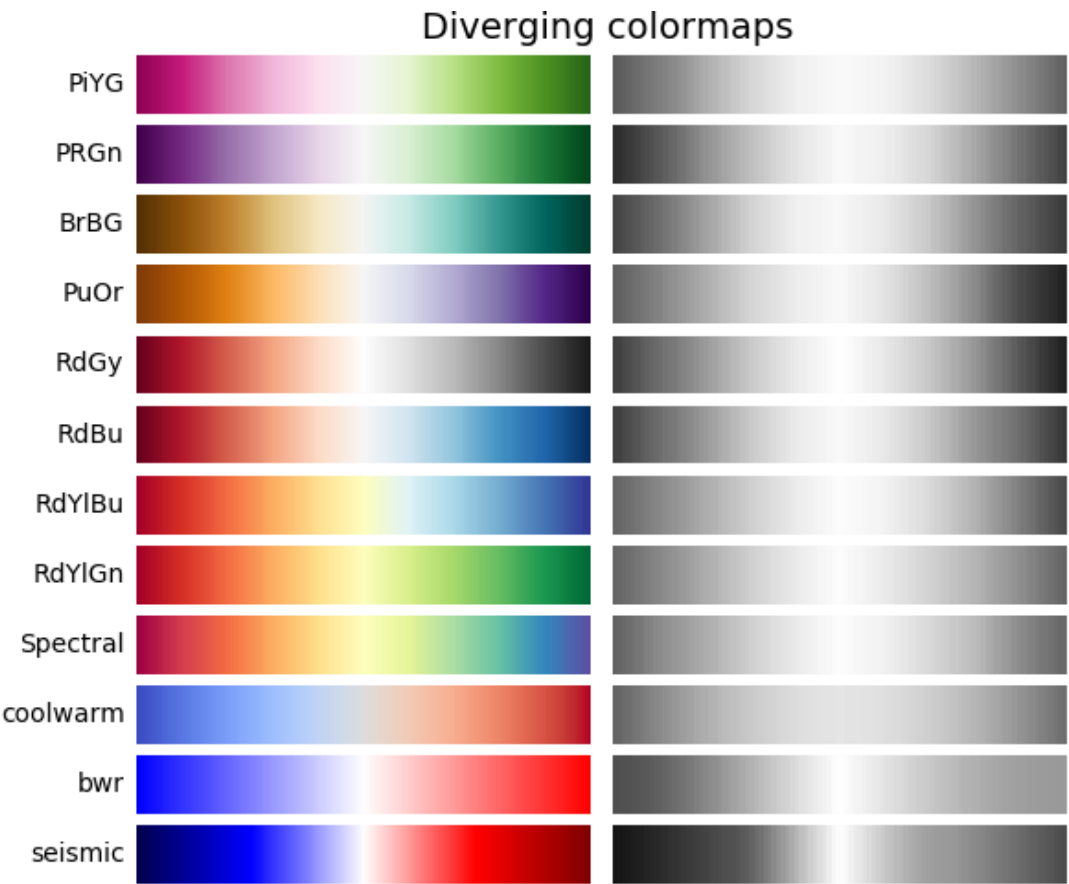
for cmap_category, cmap_list in cmaps.items():

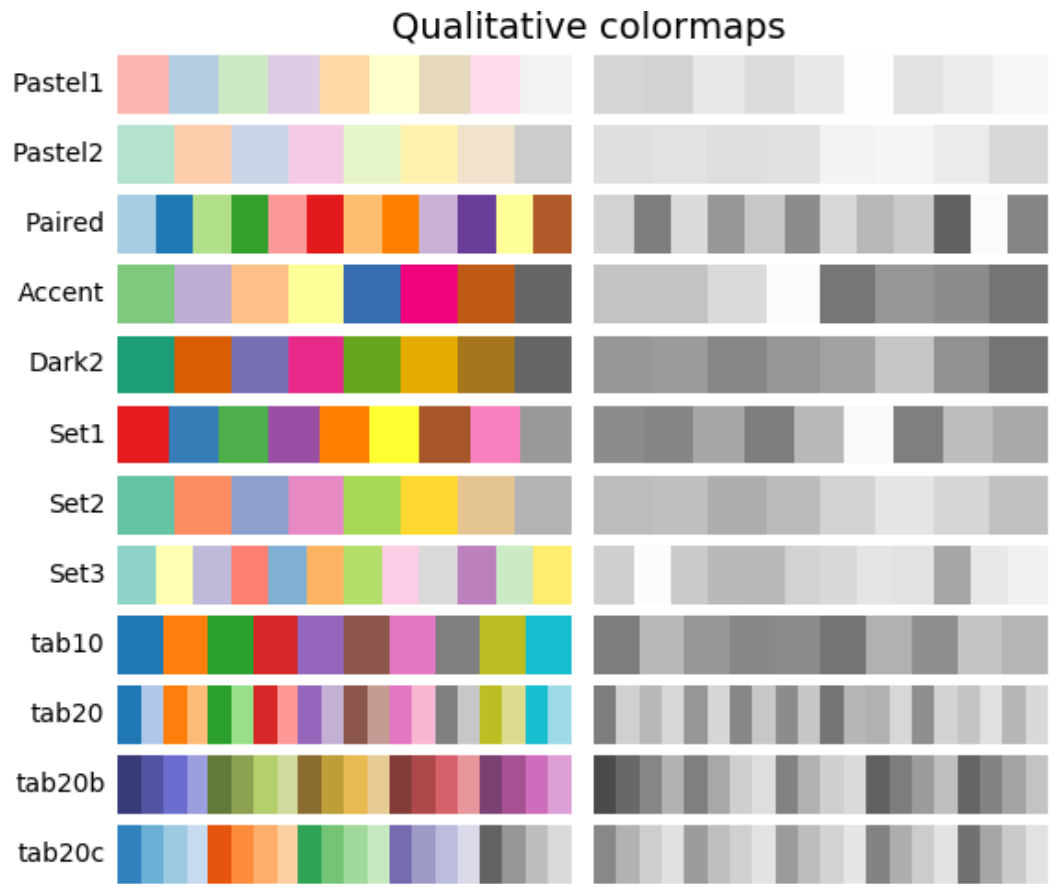
    plot_color_gradients(cmap_category, cmap_list)
```

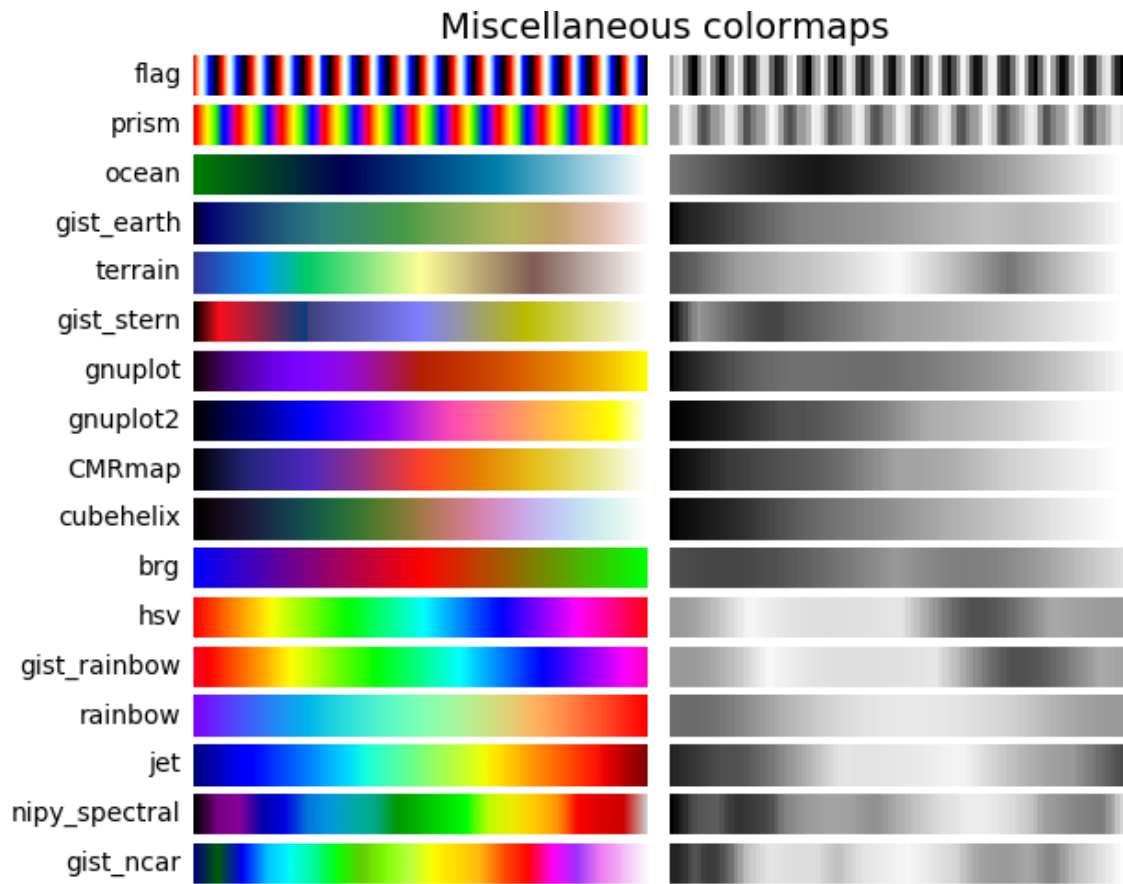












Color vision deficiencies

There is a lot of information available about color blindness (e.g., [?]). Additionally, there are tools available to convert images to how they look for different types of color vision deficiencies (e.g., [?]).

The most common form of color vision deficiency involves differentiating between red and green. Thus, avoiding colormaps with both red and green will avoid many problems in general.

References

Total running time of the script: (0 minutes 2.512 seconds)

3.5 Text

matplotlib has extensive text support, including support for mathematical expressions, truetype support for raster and vector outputs, newline separated text with arbitrary rotations, and unicode support. These tutorials cover the basics of working with text in Matplotlib.

Note: [Click here to download the full example code](#)

3.5.1 Annotations

Annotating text with Matplotlib.

Table of Contents

- [Annotations](#)
- [Basic annotation](#)
- [Advanced Annotation](#)
 - [Annotating with Text with Box](#)
 - [Annotating with Arrow](#)
 - [Placing Artist at the anchored location of the Axes](#)
 - [Using Complex Coordinates with Annotations](#)
 - [Using ConnectionPatch](#)
 - * [Advanced Topics](#)
 - [Zoom effect between Axes](#)
 - [Define Custom BoxStyle](#)

3.5.2 Basic annotation

The uses of the basic `text()` will place text at an arbitrary position on the Axes. A common use case of text is to annotate some feature of the plot, and the `annotate()` method provides helper functionality to make annotations easy. In an annotation, there are two points to consider: the location being annotated represented by the argument `xy` and the location of the text `xytext`. Both of these arguments are `(x,y)` tuples.

In this example, both the `xy` (arrow tip) and `xytext` locations (text location) are in data coordinates. There are a variety of other coordinate systems one can choose – you can specify the coordinate system of `xy` and `xytext` with one of the following strings for `xycoords` and `textcoords` (default is `'data'`)

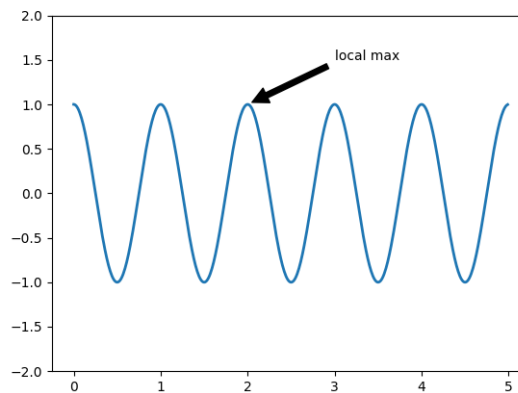


Fig. 23: Annotation Basic

argument	coordinate system
'figure points'	points from the lower left corner of the figure
'figure pixels'	pixels from the lower left corner of the figure
'figure fraction'	0,0 is lower left of figure and 1,1 is upper right
'axes points'	points from lower left corner of axes
'axes pixels'	pixels from lower left corner of axes
'axes fraction'	0,0 is lower left of axes and 1,1 is upper right
'data'	use the axes data coordinate system

For example to place the text coordinates in fractional axes coordinates, one could do:

```
ax.annotate('local max', xy=(3, 1), xycoords='data',
            xytext=(0.8, 0.95), textcoords='axes fraction',
            arrowprops=dict(facecolor='black', shrink=0.05),
            horizontalalignment='right', verticalalignment='top',
            )
```

For physical coordinate systems (points or pixels) the origin is the bottom-left of the figure or axes.

Optionally, you can enable drawing of an arrow from the text to the annotated point by giving a dictionary of arrow properties in the optional keyword argument `arrowprops`.

arrowprops key	description
width	the width of the arrow in points
frac	the fraction of the arrow length occupied by the head
headwidth	the width of the base of the arrow head in points
shrink	move the tip and base some percent away from the annotated point and text
**kwargs	any key for <code>matplotlib.patches.Polygon</code> , e.g., <code>facecolor</code>

In the example below, the `xy` point is in native coordinates (`xycoords` defaults to `'data'`). For a polar axes, this is in (theta, radius) space. The text in this example is placed in the fractional figure

coordinate system. `matplotlib.text.Text` keyword args like `horizontalalignment`, `verticalalignment` and `fontsize` are passed from `annotate` to the `Text` instance.

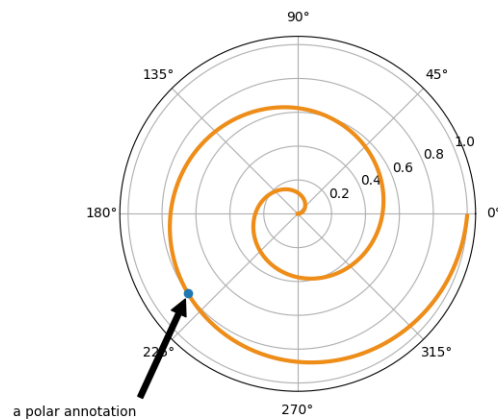


Fig. 24: Annotation Polar

For more on all the wild and wonderful things you can do with annotations, including fancy arrows, see [Advanced Annotation](#) and `/gallery/text_labels_and_annotations/annotation_demo`.

Do not proceed unless you have already read [Basic annotation](#), `text()` and `annotate()`!

3.5.3 Advanced Annotation

Annotating with Text with Box

Let's start with a simple example.

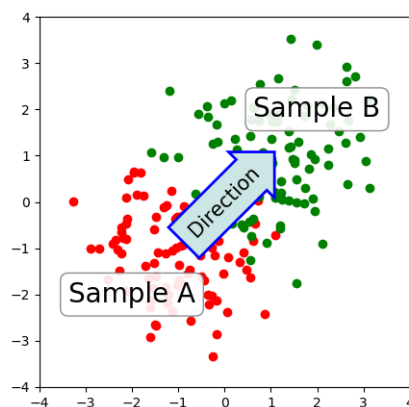


Fig. 25: Annotate Text Arrow

The `text()` function in the `pyplot` module (or `text` method of the `Axes` class) takes `bbox` keyword argument, and when given, a box around the text is drawn.

```

bbox_props = dict(boxstyle="rarrow,pad=0.3", fc="cyan", ec="b", lw=2)
t = ax.text(0, 0, "Direction", ha="center", va="center", rotation=45,
            size=15,
            bbox=bbox_props)

```

The patch object associated with the text can be accessed by:

```
bb = t.get_bbox_patch()
```

The return value is an instance of FancyBboxPatch and the patch properties like facecolor, edgewidth, etc. can be accessed and modified as usual. To change the shape of the box, use the set_boxstyle method.

```
bb.set_boxstyle("rarrow", pad=0.6)
```

The arguments are the name of the box style with its attributes as keyword arguments. Currently, following box styles are implemented.

Class	Name	Attrs
Circle	circle	pad=0.3
DArrow	darrow	pad=0.3
LArrow	larrow	pad=0.3
RArrow	rarrow	pad=0.3
Round	round	pad=0.3,rounding_size=None
Round4	round4	pad=0.3,rounding_size=None
Roundtooth	roundtooth	pad=0.3,tooth_size=None
Sawtooth	sawtooth	pad=0.3,tooth_size=None
Square	square	pad=0.3

Note that the attribute arguments can be specified within the style name with separating comma (this form can be used as "boxstyle" value of bbox argument when initializing the text instance)

```
bb.set_boxstyle("rarrow,pad=0.6")
```

Annotating with Arrow

The annotate() function in the pyplot module (or annotate method of the Axes class) is used to draw an arrow connecting two points on the plot.

```

ax.annotate("Annotation",
            xy=(x1, y1), xycoords='data',
            xytext=(x2, y2), textcoords='offset points',
            )

```

This annotates a point at xy in the given coordinate (xycoords) with the text at xytext given in textcoords. Often, the annotated point is specified in the data coordinate and the annotating text

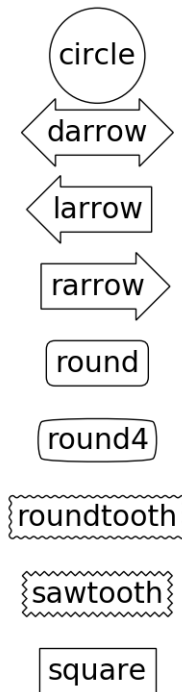


Fig. 26: Fancybox Demo

in offset points. See `annotate()` for available coordinate systems.

An arrow connecting two points (`xy` & `xytext`) can be optionally drawn by specifying the `arrowprops` argument. To draw only an arrow, use empty string as the first argument.

```
ax.annotate("",
            xy=(0.2, 0.2), xycoords='data',
            xytext=(0.8, 0.8), textcoords='data',
            arrowprops=dict(arrowstyle="->",
                           connectionstyle="arc3"),
            )
```

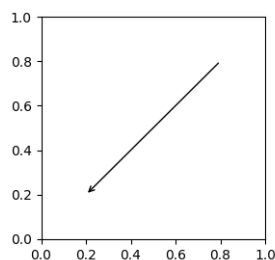


Fig. 27: Annotate Simple01

The arrow drawing takes a few steps.

1. a connecting path between two points are created. This is controlled by `connectionstyle` key

value.

2. If patch object is given (patchA & patchB), the path is clipped to avoid the patch.
3. The path is further shrunk by given amount of pixels (shrinkA & shrinkB)
4. The path is transmuted to arrow patch, which is controlled by the arrowstyle key value.

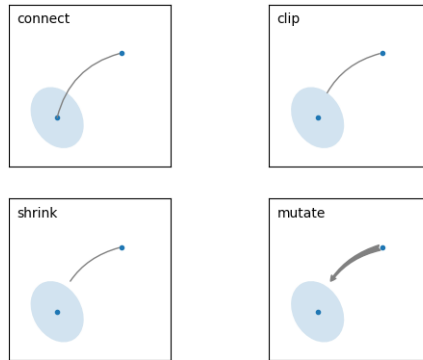


Fig. 28: Annotate Explain

The creation of the connecting path between two points is controlled by `connectionstyle` key and the following styles are available.

Name	Attrs
<code>angle</code>	<code>angleA=90,angleB=0,rad=0.0</code>
<code>angle3</code>	<code>angleA=90,angleB=0</code>
<code>arc</code>	<code>angleA=0,angleB=0,armA=None,armB=None,rad=0.0</code>
<code>arc3</code>	<code>rad=0.0</code>
<code>bar</code>	<code>armA=0.0,armB=0.0,fraction=0.3,angle=None</code>

Note that "3" in `angle3` and `arc3` is meant to indicate that the resulting path is a quadratic spline segment (three control points). As will be discussed below, some arrow style options can only be used when the connecting path is a quadratic spline.

The behavior of each connection style is (limitedly) demonstrated in the example below. (Warning : The behavior of the `bar` style is currently not well defined, it may be changed in the future).

The connecting path (after clipping and shrinking) is then mutated to an arrow patch, according to the given `arrowstyle`.

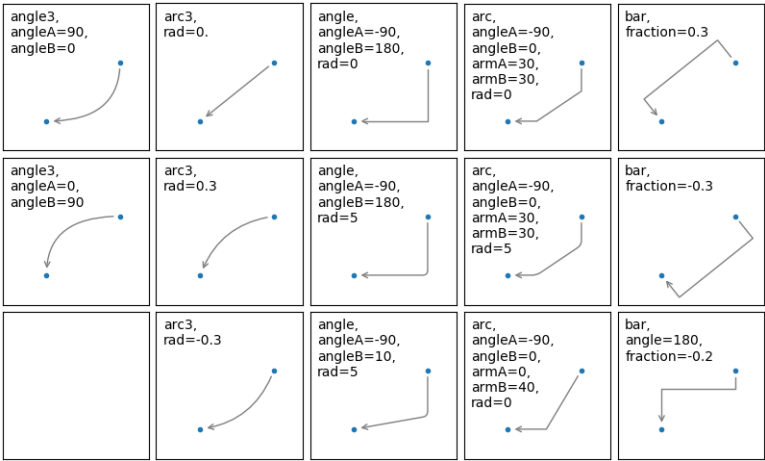


Fig. 29: Connectionstyle Demo

Name	Attrs
-	None
->	head_length=0.4,head_width=0.2
-[widthB=1.0,lengthB=0.2,angleB=None
-	widthA=1.0,widthB=1.0
- >	head_length=0.4,head_width=0.2
<-	head_length=0.4,head_width=0.2
<->	head_length=0.4,head_width=0.2
< -	head_length=0.4,head_width=0.2
< - >	head_length=0.4,head_width=0.2
fancy	head_length=0.4,head_width=0.4,tail_width=0.4
simple	head_length=0.5,head_width=0.5,tail_width=0.2
wedge	tail_width=0.3,shrink_factor=0.5

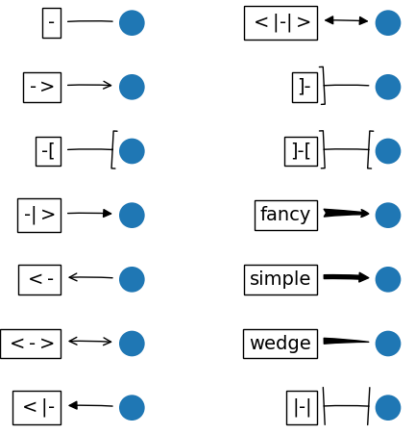


Fig. 30: Fancyarrow Demo

Some arrowstyles only work with connection styles that generate a quadratic-spline segment. They are fancy, simple, and wedge. For these arrow styles, you must use the "angle3" or "arc3" connection style.

If the annotation string is given, the patchA is set to the bbox patch of the text by default.

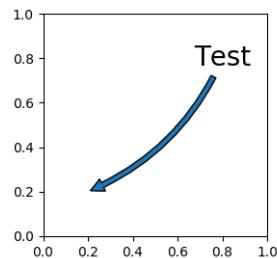


Fig. 31: Annotate Simple02

As in the text command, a box around the text can be drawn using the bbox argument.

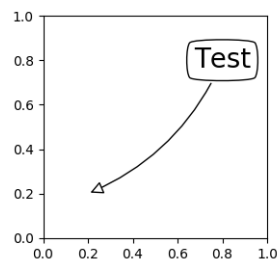


Fig. 32: Annotate Simple03

By default, the starting point is set to the center of the text extent. This can be adjusted with relpos key value. The values are normalized to the extent of the text. For example, (0,0) means lower-left corner and (1,1) means top-right.

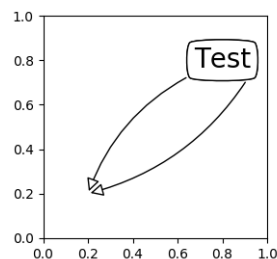


Fig. 33: Annotate Simple04

Placing Artist at the anchored location of the Axes

There are classes of artists that can be placed at an anchored location in the Axes. A common example is the legend. This type of artist can be created by using the OffsetBox class. A few

predefined classes are available in `mpl_toolkits.axes_grid1.anchored_artists` others in `matplotlib.offsetbox`

```
from matplotlib.offsetbox import AnchoredText
at = AnchoredText("Figure 1a",
                  prop=dict(size=8), frameon=True,
                  loc=2,
                  )
at.patch.set_boxstyle("round,pad=0.,rounding_size=0.2")
ax.add_artist(at)
```

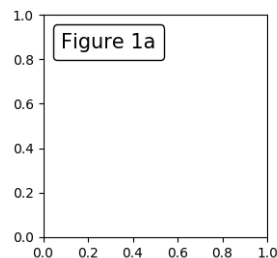


Fig. 34: Anchored Box01

The `loc` keyword has same meaning as in the `legend` command.

A simple application is when the size of the artist (or collection of artists) is known in pixel size during the time of creation. For example, If you want to draw a circle with fixed size of 20 pixel x 20 pixel (radius = 10 pixel), you can utilize `AnchoredDrawingArea`. The instance is created with a size of the drawing area (in pixels), and arbitrary artists can be added to the drawing area. Note that the extents of the artists that are added to the drawing area are not related to the placement of the drawing area itself. Only the initial size matters.

```
from mpl_toolkits.axes_grid1.anchored_artists import AnchoredDrawingArea

ada = AnchoredDrawingArea(20, 20, 0, 0,
                          loc=1, pad=0., frameon=False)
p1 = Circle((10, 10), 10)
ada.drawing_area.add_artist(p1)
p2 = Circle((30, 10), 5, fc="r")
ada.drawing_area.add_artist(p2)
```

The artists that are added to the drawing area should not have a transform set (it will be overridden) and the dimensions of those artists are interpreted as a pixel coordinate, i.e., the radius of the circles in above example are 10 pixels and 5 pixels, respectively.

Sometimes, you want your artists to scale with the data coordinate (or coordinates other than canvas pixels). You can use `AnchoredAuxTransformBox` class. This is similar to `AnchoredDrawingArea` except that the extent of the artist is determined during the drawing time respecting the specified transform.

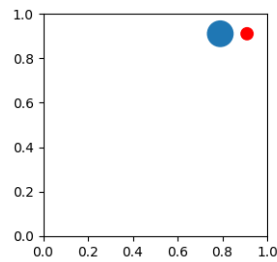


Fig. 35: Anchored Box02

```
from mpl_toolkits.axes_grid1.anchored_artists import AnchoredAuxTransformBox

box = AnchoredAuxTransformBox(ax.transData, loc=2)
el = Ellipse((0,0), width=0.1, height=0.4, angle=30) # in data coordinates!
box.drawing_area.add_artist(el)
```

The ellipse in the above example will have width and height corresponding to 0.1 and 0.4 in data coordinateing and will be automatically scaled when the view limits of the axes change.

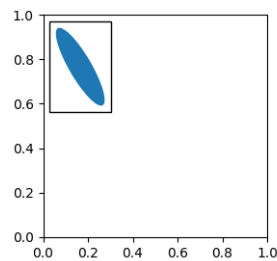


Fig. 36: Anchored Box03

As in the legend, the `bbox_to_anchor` argument can be set. Using the `HPacker` and `VPacker`, you can have an arrangement(?) of artist as in the legend (as a matter of fact, this is how the legend is created).

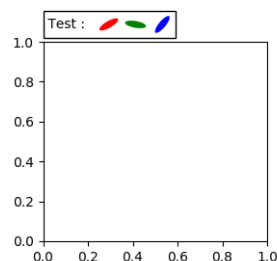


Fig. 37: Anchored Box04

Note that unlike the legend, the `bbox_transform` is set to `IdentityTransform` by default.

Using Complex Coordinates with Annotations

The Annotation in matplotlib supports several types of coordinates as described in [Basic annotation](#). For an advanced user who wants more control, it supports a few other options.

1. Transform instance. For example,

```
ax.annotate("Test", xy=(0.5, 0.5), xycoords=ax.transAxes)
```

is identical to

```
ax.annotate("Test", xy=(0.5, 0.5), xycoords="axes fraction")
```

With this, you can annotate a point in other axes.

```
ax1, ax2 = subplot(121), subplot(122)
ax2.annotate("Test", xy=(0.5, 0.5), xycoords=ax1.transData,
            xytext=(0.5, 0.5), textcoords=ax2.transData,
            arrowprops=dict(arrowstyle="->"))
```

2. Artist instance. The xy value (or xytext) is interpreted as a fractional coordinate of the bbox (return value of `get_window_extent`) of the artist.

```
an1 = ax.annotate("Test 1", xy=(0.5, 0.5), xycoords="data",
                 va="center", ha="center",
                 bbox=dict(boxstyle="round", fc="w"))
an2 = ax.annotate("Test 2", xy=(1, 0.5), xycoords=an1, # (1,0.5) of the an1's bbox
                 xytext=(30,0), textcoords="offset points",
                 va="center", ha="left",
                 bbox=dict(boxstyle="round", fc="w"),
                 arrowprops=dict(arrowstyle="->"))
```

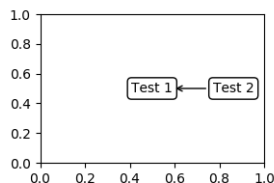


Fig. 38: Annotation with Simple Coordinates

Note that it is your responsibility that the extent of the coordinate artist (an1 in above example) is determined before an2 gets drawn. In most cases, it means that an2 needs to be drawn later than an1.

3. A callable object that returns an instance of either `BboxBase` or `Transform`. If a transform is returned, it is the same as 1 and if a bbox is returned, it is the same as 2. The callable object should take a single argument of the renderer instance. For example, the following two commands give identical results

```

an2 = ax.annotate("Test 2", xy=(1, 0.5), xycoords=an1,
                  xytext=(30,0), textcoords="offset points")
an2 = ax.annotate("Test 2", xy=(1, 0.5), xycoords=an1.get_window_extent,
                  xytext=(30,0), textcoords="offset points")

```

4. A tuple of two coordinate specifications. The first item is for the x-coordinate and the second is for the y-coordinate. For example,

```

annotate("Test", xy=(0.5, 1), xycoords=("data", "axes fraction"))

```

0.5 is in data coordinates, and 1 is in normalized axes coordinates. You may use an artist or transform as with a tuple. For example,

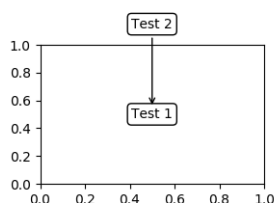


Fig. 39: Annotation with Simple Coordinates 2

5. Sometimes, you want your annotation with some "offset points", not from the annotated point but from some other point. OffsetFrom is a helper class for such cases.

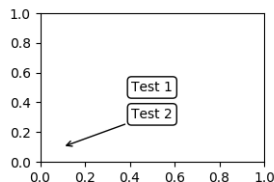


Fig. 40: Annotation with Simple Coordinates 3

You may take a look at this example /gallery/text_labels_and_annotations/annotation_demo.

Using ConnectionPatch

The ConnectionPatch is like an annotation without text. While the annotate function is recommended in most situations, the ConnectionPatch is useful when you want to connect points in different axes.

```

from matplotlib.patches import ConnectionPatch
xy = (0.2, 0.2)
con = ConnectionPatch(xyA=xy, xyB=xy, coordsA="data", coordsB="data",
                     axesA=ax1, axesB=ax2)
ax2.add_artist(con)

```

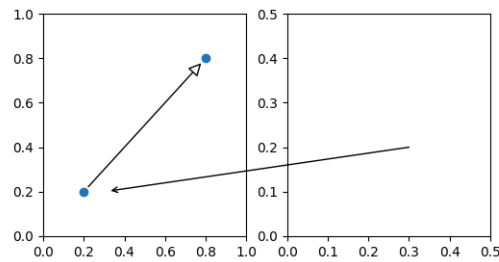



Fig. 41: Connect Simple01

The above code connects point `xy` in the data coordinates of `ax1` to point `xy` in the data coordinates of `ax2`. Here is a simple example.

While the `ConnectionPatch` instance can be added to any axes, you may want to add it to the axes that is latest in drawing order to prevent overlap by other axes.

Advanced Topics

Zoom effect between Axes

`mpl_toolkits.axes_grid1.inset_locator` defines some patch classes useful for interconnecting two axes. Understanding the code requires some knowledge of how `mpl`'s transform works. But, utilizing it will be straight forward.

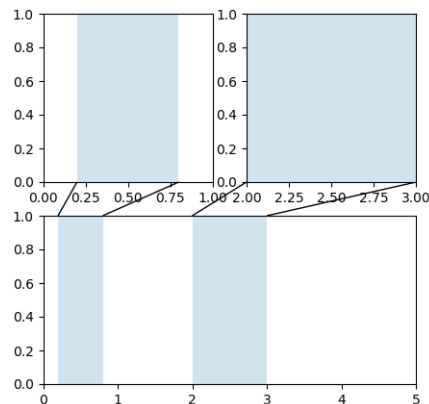


Fig. 42: Axes Zoom Effect

Define Custom BoxStyle

You can use a custom box style. The value for the boxstyle can be a callable object in the following forms.:

```
def __call__(self, x0, y0, width, height, mutation_size,
             aspect_ratio=1.):
    """
    Given the location and size of the box, return the path of
    the box around it.

    - *x0*, *y0*, *width*, *height* : location and size of the box
    - *mutation_size* : a reference scale for the mutation.
    - *aspect_ratio* : aspect-ratio for the mutation.
    """
    path = ...
    return path
```

Here is a complete example.

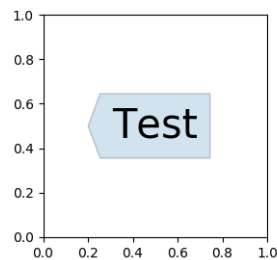


Fig. 43: Custom Boxstyle01

However, it is recommended that you derive from the `matplotlib.patches.BoxStyle._Base` as demonstrated below.

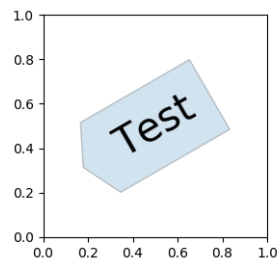


Fig. 44: Custom Boxstyle02

Similarly, you can define a custom `ConnectionStyle` and a custom `ArrowStyle`. See the source code of `lib/matplotlib/patches.py` and check how each style class is defined.

Note: [Click here to download the full example code](#)

3.5.4 Text rendering With LaTeX

Rendering text with LaTeX in Matplotlib.

Matplotlib has the option to use LaTeX to manage all text layout. This option is available with the following backends:

- Agg
- PS
- PDF

The LaTeX option is activated by setting `text.usetex : True` in your rc settings. Text handling with matplotlib's LaTeX support is slower than matplotlib's very capable `mathtext`, but is more flexible, since different LaTeX packages (font packages, math packages, etc.) can be used. The results can be striking, especially when you take care to use the same fonts in your figures as in the main document.

Matplotlib's LaTeX support requires a working LaTeX installation, `dvipng` (which may be included with your LaTeX installation), and `Ghostscript` (GPL Ghostscript 8.60 or later is recommended). The executables for these external dependencies must all be located on your PATH.

There are a couple of options to mention, which can be changed using rc settings. Here is an example matplotlibrc file:

```
font.family      : serif
font.serif       : Times, Palatino, New Century Schoolbook, Bookman, Computer Modern,
↳Roman
font.sans-serif   : Helvetica, Avant Garde, Computer Modern Sans serif
font.cursive      : Zapf Chancery
font.monospace    : Courier, Computer Modern Typewriter

text.usetex      : true
```

The first valid font in each family is the one that will be loaded. If the fonts are not specified, the Computer Modern fonts are used by default. All of the other fonts are Adobe fonts. Times and Palatino each have their own accompanying math fonts, while the other Adobe serif fonts make use of the Computer Modern math fonts. See the `PSNFSS` documentation for more details.

To use LaTeX and select Helvetica as the default font, without editing matplotlibrc use:

```
from matplotlib import rc
rc('font',**{'family':'sans-serif','sans-serif':['Helvetica']})
## for Palatino and other serif fonts use:
#rc('font',**{'family':'serif','serif':['Palatino']})
rc('text', usetex=True)
```

Here is the standard example, `tex_demo.py`:

Note that display math mode ($e=mc^2$) is not supported, but adding the command `\displaystyle`, as in `tex_demo.py`, will produce the same results.

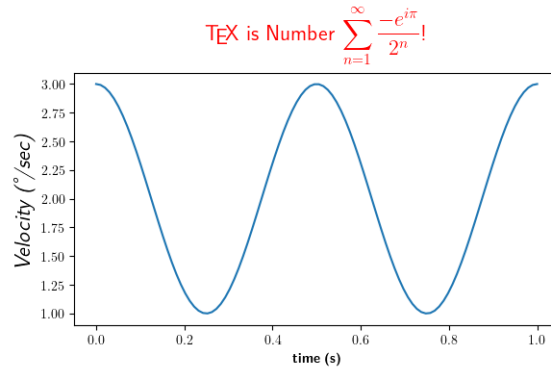


Fig. 45: TeX Demo

Note: Certain characters require special escaping in TeX, such as:

\$ % & ~ _ ^ \ { } \ (\) \ [\]

Therefore, these characters will behave differently depending on the rcParam text.usetex flag.

usetex with unicode

It is also possible to use unicode strings with the LaTeX text manager, here is an example taken from `tex_demo.py`. The axis labels include Unicode text:

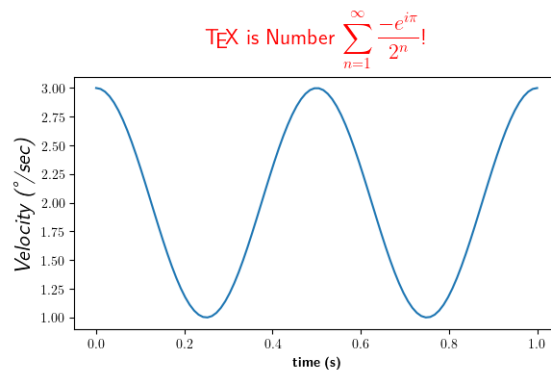


Fig. 46: TeX Unicode Demo

Postscript options

In order to produce encapsulated postscript files that can be embedded in a new LaTeX document, the default behavior of matplotlib is to distill the output, which removes some postscript operators used by LaTeX that are illegal in an eps file. This step produces results which may be unacceptable to some users, because the text is coarsely rasterized and converted to bitmaps, which are not

scalable like standard postscript, and the text is not searchable. One workaround is to set `ps.distiller.res` to a higher value (perhaps 6000) in your rc settings, which will produce larger files but may look better and scale reasonably. A better workaround, which requires [Poppler](#) or [Xpdf](#), can be activated by changing the `ps.usedistiller` rc setting to `xpdf`. This alternative produces postscript without rasterizing text, so it scales properly, can be edited in Adobe Illustrator, and searched text in pdf documents.

Possible hangups

- On Windows, the `PATH` environment variable may need to be modified to include the directories containing the `latex`, `dvipng` and `ghostscript` executables. See [Environment Variables and Setting environment variables in windows](#) for details.
- Using MiKTeX with Computer Modern fonts, if you get odd `*Agg` and PNG results, go to [MiKTeX/Options](#) and update your format files
- On Ubuntu and Gentoo, the base texlive install does not ship with the `typelcm` package. You may need to install some of the extra packages to get all the goodies that come bundled with other latex distributions.
- Some progress has been made so matplotlib uses the dvi files directly for text layout. This allows latex to be used for text layout with the pdf and svg backends, as well as the `*Agg` and PS backends. In the future, a latex installation may be the only external dependency.

Troubleshooting

- Try deleting your `.matplotlib/tex.cache` directory. If you don't know where to find `.matplotlib`, see [matplotlib configuration and cache directory locations](#).
- Make sure LaTeX, `dvipng` and `ghostscript` are each working and on your `PATH`.
- Make sure what you are trying to do is possible in a LaTeX document, that your LaTeX syntax is valid and that you are using raw strings if necessary to avoid unintended escape sequences.
- Most problems reported on the mailing list have been cleared up by upgrading [Ghostscript](#). If possible, please try upgrading to the latest release before reporting problems to the list.
- The `text.latex.preamble` rc setting is not officially supported. This option provides lots of flexibility, and lots of ways to cause problems. Please disable this option before reporting problems to the mailing list.
- If you still need help, please see [Getting help](#)

Note: [Click here to download the full example code](#)

3.5.5 Typesetting With XeLaTeX/LuaLaTeX

How to typeset text with the pgf backend in Matplotlib.

Using the pgf backend, matplotlib can export figures as pgf drawing commands that can be processed with pdflatex, xelatex or lualatex. XeLaTeX and LuaLaTeX have full unicode support and can use any font that is installed in the operating system, making use of advanced typographic features of OpenType, AAT and Graphite. Pgf pictures created by `plt.savefig('figure.pgf')` can be embedded as raw commands in LaTeX documents. Figures can also be directly compiled and saved to PDF with `plt.savefig('figure.pdf')` by either switching to the backend

```
matplotlib.use('pgf')
```

or registering it for handling pdf output

```
from matplotlib.backends.backend_pgf import FigureCanvasPgf
matplotlib.backend_bases.register_backend('pdf', FigureCanvasPgf)
```

The second method allows you to keep using regular interactive backends and to save xelatex, lualatex or pdflatex compiled PDF files from the graphical user interface.

Matplotlib's pgf support requires a recent [LaTeX](#) installation that includes the TikZ/PGF packages (such as [TeXLive](#)), preferably with XeLaTeX or LuaLaTeX installed. If either pdftocairo or ghostscript is present on your system, figures can optionally be saved to PNG images as well. The executables for all applications must be located on your PATH.

Rc parameters that control the behavior of the pgf backend:

Parameter	Documentation
<code>pgf.preamble</code>	Lines to be included in the LaTeX preamble
<code>pgf.rcfonts</code>	Setup fonts from rc params using the fontspec package
<code>pgf.texsystem</code>	Either "xelatex" (default), "lualatex" or "pdflatex"

Note: TeX defines a set of special characters, such as:

```
# $ % & ~ _ ^ \ { }
```

Generally, these characters must be escaped correctly. For convenience, some characters (`_`, `^`, `%`) are automatically escaped outside of math environments.

Font specification

The fonts used for obtaining the size of text elements or when compiling figures to PDF are usually defined in the matplotlib rc parameters. You can also use the LaTeX default Computer Modern fonts by clearing the lists for `font.serif`, `font.sans-serif` or `font.monospace`. Please note that the glyph coverage of these fonts is very limited. If you want to keep the Computer Modern font face but

require extended unicode support, consider installing the [Computer Modern Unicode](#) fonts CMU Serif, CMU Sans Serif, etc.

When saving to .pgf, the font configuration matplotlib used for the layout of the figure is included in the header of the text file.

```
"""
=====
Pgf Fonts
=====

"""
# -*- coding: utf-8 -*-

import matplotlib as mpl
mpl.use("pgf")
pgf_with_rc_fonts = {
    "font.family": "serif",
    "font.serif": [], # use latex default serif font
    "font.sans-serif": ["DejaVu Sans"], # use a specific sans-serif font
}
mpl.rcParams.update(pgf_with_rc_fonts)

import matplotlib.pyplot as plt
plt.figure(figsize=(4.5, 2.5))
plt.plot(range(5))
plt.text(0.5, 3., "serif")
plt.text(0.5, 2., "monospace", family="monospace")
plt.text(2.5, 2., "sans-serif", family="sans-serif")
plt.text(2.5, 1., "comic sans", family="Comic Sans MS")
plt.xlabel(u"μ is not  $\mu$ ")
plt.tight_layout(.5)
```

Custom preamble

Full customization is possible by adding your own commands to the preamble. Use the `pgf.preamble` parameter if you want to configure the math fonts, using `unicode-math` for example, or for loading additional packages. Also, if you want to do the font configuration yourself instead of using the fonts specified in the rc parameters, make sure to disable `pgf.rcfonts`.

```
"""
=====
Pgf Preamble
=====

"""
```

(continues on next page)

(continued from previous page)

```
# -*- coding: utf-8 -*-
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)

import six

import matplotlib as mpl
mpl.use("pgf")
pgf_with_custom_preamble = {
    "font.family": "serif", # use serif/main font for text elements
    "text.usetex": True,    # use inline math for ticks
    "pgf.rcfonts": False,   # don't setup fonts from rc parameters
    "pgf.preamble": [
        "\\usepackage{units}",      # load additional packages
        "\\usepackage{metalogo}",
        "\\usepackage{unicode-math}", # unicode math setup
        r"\setmathfont{xits-math.otf}",
        r"\setmainfont{DejaVu Serif}", # serif font via preamble
    ]
}
mpl.rcParams.update(pgf_with_custom_preamble)
```

Choosing the TeX system

The TeX system to be used by matplotlib is chosen by the `pgf.texsystem` parameter. Possible values are 'xelatex' (default), 'lualatex' and 'pdflatex'. Please note that when selecting `pdflatex` the fonts and unicode handling must be configured in the preamble.

```
"""
=====
Pgf Texsystem
=====

"""
# -*- coding: utf-8 -*-

import matplotlib as mpl
mpl.use("pgf")
pgf_with_pdflatex = {
    "pgf.texsystem": "pdflatex",
    "pgf.preamble": [
        r"\usepackage[utf8x]{inputenc}",
        r"\usepackage[T1]{fontenc}",
        r"\usepackage{cmbright}",
    ]
}
```

(continues on next page)

(continued from previous page)

```

    ]
}
mpl.rcParams.update(pgf_with_pdflatex)

import matplotlib.pyplot as plt
plt.figure(figsize=(4.5, 2.5))
plt.plot(range(5))
plt.text(0.5, 3., "serif", family="serif")
plt.text(0.5, 2., "monospace", family="monospace")
plt.text(2.5, 2., "sans-serif", family="sans-serif")
plt.xlabel(u" $\mu$  is not  $\mu$ ")
plt.tight_layout(.5)

```

Troubleshooting

- Please note that the TeX packages found in some Linux distributions and MiKTeX installations are dramatically outdated. Make sure to update your package catalog and upgrade or install a recent TeX distribution.
- On Windows, the PATH environment variable may need to be modified to include the directories containing the latex, dvipng and ghostscript executables. See Environment Variables and Setting environment variables in windows for details.
- A limitation on Windows causes the backend to keep file handles that have been opened by your application open. As a result, it may not be possible to delete the corresponding files until the application closes (see [#1324](#)).
- Sometimes the font rendering in figures that are saved to png images is very bad. This happens when the pdftocairo tool is not available and ghostscript is used for the pdf to png conversion.
- Make sure what you are trying to do is possible in a LaTeX document, that your LaTeX syntax is valid and that you are using raw strings if necessary to avoid unintended escape sequences.
- The pgf.preamble rc setting provides lots of flexibility, and lots of ways to cause problems. When experiencing problems, try to minimize or disable the custom preamble.
- Configuring an unicode-math environment can be a bit tricky. The TeXLive distribution for example provides a set of math fonts which are usually not installed system-wide. XeTeX, unlike LuaLatex, cannot find these fonts by their name, which is why you might have to specify `\setmathfont{xits-math.otf}` instead of `\setmathfont{XITS Math}` or alternatively make the fonts available to your OS. See this [tex.stackexchange.com](https://tex.stackexchange.com/questions/1324) question for more details.
- If the font configuration used by matplotlib differs from the font setting in your LaTeX document, the alignment of text elements in imported figures may be off. Check the header of your .pgf file if you are unsure about the fonts matplotlib used for the layout.

- Vector images and hence .pgf files can become bloated if there are a lot of objects in the graph. This can be the case for image processing or very big scatter graphs. In an extreme case this can cause TeX to run out of memory: "TeX capacity exceeded, sorry" You can configure latex to increase the amount of memory available to generate the .pdf image as discussed on tex.stackexchange.com. Another way would be to "rasterize" parts of the graph causing problems using either the `rasterized=True` keyword, or `.set_rasterized(True)` as per [this example](#).
- If you still need help, please see [Getting help](#)

Note: [Click here to download the full example code](#)

3.5.6 Writing mathematical expressions

An introduction to writing mathematical expressions in Matplotlib.

You can use a subset TeX markup in any matplotlib text string by placing it inside a pair of dollar signs (\$).

Note that you do not need to have TeX installed, since Matplotlib ships its own TeX expression parser, layout engine, and fonts. The layout engine is a fairly direct adaptation of the layout algorithms in Donald Knuth's TeX, so the quality is quite good (matplotlib also provides a `usetex` option for those who do want to call out to TeX to generate their text (see [Text rendering With LaTeX](#)).

Any text element can use math text. You should use raw strings (precede the quotes with an 'r'), and surround the math text with dollar signs (\$), as in TeX. Regular text and `mathtext` can be interleaved within the same string. `Mathtext` can use DejaVu Sans (default), DejaVu Serif, the Computer Modern fonts (from (La)TeX), **STIX** fonts (with are designed to blend well with Times), or a Unicode font that you provide. The `mathtext` font can be selected with the customization variable `mathtext.fontset` (see [Customizing Matplotlib with style sheets and rcParams](#))

Here is a simple example:

```
# plain text
plt.title('alpha > beta')
```

produces "alpha > beta".

Whereas this:

```
# math text
plt.title(r'$\alpha > \beta$')
```

produces " $\alpha > \beta$ ".

Note: `Mathtext` should be placed between a pair of dollar signs (\$). To make it easy to display monetary values, e.g., "\$100.00", if a single dollar sign is present in the entire string, it will be

displayed verbatim as a dollar sign. This is a small change from regular TeX, where the dollar sign in non-math text would have to be escaped (`\$`).

Note: While the syntax inside the pair of dollar signs (\$) aims to be TeX-like, the text outside does not. In particular, characters such as:

```
# $ % & ~ _ ^ \ { } \ ( \ ) \ [ \ ]
```

have special meaning outside of math mode in TeX. Therefore, these characters will behave differently depending on the `rcParam text.usetex` flag. See the [usetex tutorial](#) for more information.

Subscripts and superscripts

To make subscripts and superscripts, use the `'_'` and `'^'` symbols:

```
r'$\alpha_i > \beta_i$'
```

$$\alpha_i > \beta_i$$

Some symbols automatically put their sub/superscripts under and over the operator. For example, to write the sum of x_i from 0 to ∞ , you could do:

```
r'$\sum_{i=0}^{\infty} x_i$'
```

$$\sum_{i=0}^{\infty} x_i$$

Fractions, binomials, and stacked numbers

Fractions, binomials, and stacked numbers can be created with the `\frac{...}{...}`, `\binom{...}{...}` and `\stackrel{...}{...}` commands, respectively:

```
r'$\frac{3}{4} \binom{3}{4} \stackrel{3}{4}$'
```

produces

$$\frac{3}{4} \binom{3}{4}^3_4$$

Fractions can be arbitrarily nested:

```
r'$\frac{5 - \frac{1}{x}}{4}$'
```

produces

$$\frac{5 - \frac{1}{x}}{4}$$

Note that special care needs to be taken to place parentheses and brackets around fractions. Doing things the obvious way produces brackets that are too small:

```
r'$(\frac{5 - \frac{1}{x}}{4})$'
```

$$\left(\frac{5 - \frac{1}{x}}{4}\right)$$

The solution is to precede the bracket with `\left` and `\right` to inform the parser that those brackets encompass the entire object.:

```
r'$\left(\frac{5 - \frac{1}{x}}{4}\right)$'
```

$$\left(\frac{5 - \frac{1}{x}}{4}\right)$$

Radicals

Radicals can be produced with the `\sqrt[]{}{}` command. For example:

```
r'$\sqrt{2}$'
```

$$\sqrt{2}$$

Any base can (optionally) be provided inside square brackets. Note that the base must be a simple expression, and can not contain layout commands such as fractions or sub/superscripts:

```
r'$\sqrt[3]{x}$'
```

$$\sqrt[3]{x}$$

Fonts

The default font is italics for mathematical symbols.

Note: This default can be changed using the `mathtext.default rcParam`. This is useful, for example, to use the same font as regular non-math text for math text, by setting it to regular.

To change fonts, e.g., to write "sin" in a Roman font, enclose the text in a font command:

```
r'$s(t) = \mathcal{A}\mathrm{sin}(2 \omega t)$'
```

$$s(t) = \mathcal{A}\sin(2\omega t)$$

More conveniently, many commonly used function names that are typeset in a Roman font have shortcuts. So the expression above could be written as follows:

```
r'$s(t) = \mathcal{A}\sin(2 \omega t)$'
```

$$s(t) = \mathcal{A} \sin(2\omega t)$$

Here "s" and "t" are variable in italics font (default), "sin" is in Roman font, and the amplitude "A" is in calligraphy font. Note in the example above the calligraphy A is squished into the sin. You can use a spacing command to add a little whitespace between them:

```
r's(t) = \mathcal{A}\ /\ \sin(2 \omega t)'
```

$$s(t) = \mathcal{A} \sin(2\omega t)$$

The choices available with all fonts are:

Command	Result
<code>\mathrm{Roman}</code>	Roman
<code>\mathit{Italic}</code>	<i>Italic</i>
<code>\mathtt{Typewriter}</code>	Typewriter
<code>\mathcal{CALLIGRAPHY}</code>	<i>CALLIGRAPHY</i>

When using the [STIX](#) fonts, you also have the choice of:

Command	Result
<code>\mathbb{blackboard}</code>	\mathbb{A}
<code>\mathrm{\mathbb{blackboard}}</code>	\mathbb{A}
<code>\mathfrak{Fraktur}</code>	\mathfrak{A}
<code>\mathsf{sansserif}</code>	sansserif
<code>\mathrm{\mathsf{sansserif}}</code>	sansserif

There are also three global "font sets" to choose from, which are selected using the `mathtext.fontset` parameter in `matplotlibrc`.

cm: Computer Modern (TeX)

$$\mathcal{R} \prod_{i=\alpha_{i+1}}^{\infty} a_i \sin(2\pi f x_i)$$

stix: STIX (designed to blend well with Times)

$$\mathcal{R} \prod_{i=\alpha_{i+1}}^{\infty} a_i \sin(2\pi f x_i)$$

stixsans: STIX sans-serif

$$\mathcal{R} \prod_{i=\alpha_{i+1}}^{\infty} a_i \sin(2\pi f x_i)$$

Additionally, you can use `\mathdefault{...}` or its alias `\mathregular{...}` to use the font used for regular text outside of `mathtext`. There are a number of limitations to this approach, most notably that far fewer symbols will be available, but it can be useful to make math expressions blend well with other text in the plot.

Custom fonts

`mathtext` also provides a way to use custom fonts for math. This method is fairly tricky to use, and should be considered an experimental feature for patient users only. By setting the rcParam `mathtext.fontset` to `custom`, you can then set the following parameters, which control which font file to use for a particular set of math characters.

Parameter	Corresponds to
<code>mathtext.it</code>	<code>\mathit{}</code> or default italic
<code>mathtext.rm</code>	<code>\mathrm{}</code> Roman (upright)
<code>mathtext.tt</code>	<code>\mathtt{}</code> Typewriter (monospace)
<code>mathtext.bf</code>	<code>\mathbf{}</code> bold italic
<code>mathtext.cal</code>	<code>\mathcal{}</code> calligraphic
<code>mathtext.sf</code>	<code>\mathsf{}</code> sans-serif

Each parameter should be set to a `fontconfig` font descriptor (as defined in the yet-to-be-written font chapter).

The fonts used should have a Unicode mapping in order to find any non-Latin characters, such as Greek. If you want to use a math symbol that is not contained in your custom fonts, you can set the rcParam `mathtext.fallback_to_cm` to `True` which will cause the `mathtext` system to use characters from the default Computer Modern fonts whenever a particular character can not be found in the custom font.

Note that the math glyphs specified in Unicode have evolved over time, and many fonts may not have glyphs in the correct place for `mathtext`.

Accents

An accent command may precede any symbol to add an accent above it. There are long and short forms for some of them.

Command	Result
<code>\acute a</code> or <code>\'a</code>	\acute{a}
<code>\bar a</code>	\bar{a}
<code>\breve a</code>	\breve{a}
<code>\ddot a</code> or <code>\''a</code>	\ddot{a}
<code>\dot a</code> or <code>\.a</code>	\dot{a}
<code>\grave a</code> or <code>\`a</code>	\grave{a}
<code>\hat a</code> or <code>\^a</code>	\hat{a}
<code>\tilde a</code> or <code>\~a</code>	\tilde{a}
<code>\vec a</code>	\vec{a}
<code>\overline{abc}</code>	\overline{abc}

In addition, there are two special accents that automatically adjust to the width of the symbols below:

Command	Result
<code>\widehat{xyz}</code>	\widehat{xyz}
<code>\widetilde{xyz}</code>	\widetilde{xyz}

Care should be taken when putting accents on lower-case i's and j's. Note that in the following `\imath` is used to avoid the extra dot over the i:

r"\$\hat i\$ \ \hat \imath\$"

\hat{i} $\hat{\imath}$

Symbols

You can also use a large number of the TeX symbols, as in `\infty`, `\leftarrow`, `\sum`, `\int`.

Lower-case Greek

α <code>\alpha</code>	β <code>\beta</code>	χ <code>\chi</code>	δ <code>\delta</code>	F <code>\digamma</code>	ϵ <code>\epsilon</code>
η <code>\eta</code>	γ <code>\gamma</code>	ι <code>\iota</code>	κ <code>\kappa</code>	λ <code>\lambda</code>	μ <code>\mu</code>
ν <code>\nu</code>	ω <code>\omega</code>	ϕ <code>\phi</code>	π <code>\pi</code>	ψ <code>\psi</code>	ρ <code>\rho</code>
σ <code>\sigma</code>	τ <code>\tau</code>	θ <code>\theta</code>	υ <code>\upsilon</code>	ε <code>\varepsilon</code>	\varkappa <code>\varkappa</code>
φ <code>\varphi</code>	ϖ <code>\varpi</code>	ϱ <code>\varrho</code>	ς <code>\varsigma</code>	ϑ <code>\vartheta</code>	ξ <code>\xi</code>
ζ <code>\zeta</code>					

Upper-case Greek

Δ \Delta	Γ \Gamma \Gamma	Λ \Lambda \Lambda	Ω \Omega \Omega	Φ \Phi \Phi	Π \Pi \Pi	Ψ \Psi \Psi	Σ \Sigma \Sigma
Θ \Theta \Theta	Υ \Upsilon \Upsilon	Ξ \Xi \Xi	Υ \Upsilon \Upsilon	∇ \nabla \nabla			

Hebrew

\aleph \aleph	\beth \beth	\daleth \daleth	\gimel \gimel		
-----------------	---------------	-------------------	-----------------	--	--

Delimiters

$//$	$[$ \lbrack	\Downarrow \Downarrow \Downarrow	\Uparrow \Uparrow \Uparrow	\Vdash \Vdash	\backslash \backslash \backslash
\downarrow \downarrow	\langle \langle	\lceil \lceil	\lfloor \lfloor	\llcorner \llcorner	\lrcorner \lrcorner
\rangle \rangle	\rceil \rceil	\rfloor \rfloor	\ulcorner \ulcorner	\uparrow \uparrow	\urcorner \urcorner
$ $ \vert	$\{$ \{	$\ $ \	$\}$ \}	$]$ \]	$ $

Big symbols

\bigcap \bigcap \bigcap	\bigcup \bigcup \bigcup	\bigodot \bigodot \bigodot	\bigoplus \bigoplus \bigoplus	\bigotimes \bigotimes \bigotimes	\biguplus \biguplus \biguplus
\bigvee \bigvee	\bigwedge \bigwedge \bigwedge	\coprod \coprod	\int \int	\oint \oint	\prod \prod
\sum \sum					

Standard function names

\Pr \Pr	\arccos \arccos	\arcsin \arcsin	\arctan \arctan	\arg \arg	\cos \cos
\cosh \cosh	\cot \cot	\coth \coth	\csc \csc	\deg \deg	\det \det
\dim \dim	\exp \exp	\gcd \gcd	\hom \hom	\inf \inf	\ker \ker
\lg \lg	\lim \lim	\liminf \liminf	\limsup \limsup	\ln \ln	\log \log
\max \max	\min \min	\sec \sec	\sin \sin	\sinh \sinh	\sup \sup
\tan \tan	\tanh \tanh				

Binary operation and relation symbols

\bumpeq \Bumpeq	\cap \Cap	\cup \Cup	\doteq \Doteq
\Join \Join	\subseteq \Subset	\supseteq \Supset	\Vdash \Vdash
\Vdash \Vdash	\approx \approx	\approx \approx	$*$ \ast
\asymp \asymp	\backsim \backsim	\backsim \backsim	\backsimeq \backsimeq

Continued on next page

Table 2 – continued from previous page

$\bar{\wedge}$ \barwedge	\because \because	\emptyset \between	\bigcirc \bigcirc
\bigtriangledown \bigtriangledown	\bigtriangleup \bigtriangleup	\blacktriangleleft \blacktriangleleft	\blacktriangleright \blacktriangleright
\bot \bot	\bowtie \bowtie	\boxdot \boxdot	\boxminus \boxminus
\boxplus \boxplus	\boxtimes \boxtimes	\bullet \bullet	\bumpeq \bumpeq
\cap \cap	\cdot \cdot	\circ \circ	\circeq \circeq
\coloneq \coloneq	\cong \cong	\cup \cup	\curlyeqprec \curlyeqprec
\curlyeqsucc \curlyeqsucc	\curlyvee \curlyvee	\curlywedge \curlywedge	\dagger \dag
\dashv \dashv	\ddag \ddag	\diamond \diamond	\div \div
\divideontimes \divideontimes	\doteq \doteq	\doteqdot \doteqdot	\dotplus \dotplus
\doublebarwedge \doublebarwedge	\eqcirc \eqcirc	\eqcolon \eqcolon	\eqsim \eqsim
\eqslantgtr \eqslantgtr	\eqslantless \eqslantless	\equiv \equiv	\fallingdotseq \fallingdotseq
\frown \frown	\geq \geq	\geqeq \geqeq	\geqslant \geqslant
\gg \gg	\ggg \ggg	\gtrapprox \gtrapprox	\gtrless \gtrless
\gnsim \gnsim	\gtrapprox \gtrapprox	\gtrdot \gtrdot	\gtreqless \gtreqless
\gtreqless \gtreqless	\gtrless \gtrless	\gtrsim \gtrsim	\in \in
\intercal \intercal	\leftthreetimes \leftthreetimes	\leq \leq	\leqeq \leqeq
\leslant \leslant	\lessapprox \lessapprox	\lessdot \lessdot	\lesseqgtr \lesseqgtr
\lesseqqgtr \lesseqqgtr	\lessgtr \lessgtr	\lesssim \lesssim	\ll \ll
\lll \lll	\lnapprox \lnapprox	\lneqq \lneqq	\lnsim \lnsim
\ltimes \ltimes	\mid \mid	\models \models	\mp \mp
\nVDash \nVDash	\nVdash \nVdash	\napprox \napprox	\ncong \ncong
\neq \neq	\neq \neq	\neq \neq	\nequiv \nequiv
\ngeq \ngeq	\ngtr \ngtr	\ni \ni	\nleq \nleq
\nless \nless	\nmid \nmid	\notin \notin	\nparallel \nparallel
\nprec \nprec	\nsim \nsim	\nsubset \nsubset	\nsubseteq \nsubseteq
\nsucc \nsucc	\nsupset \nsupset	\nsupseteq \nsupseteq	\ntriangleleft \ntriangleleft
\ntrianglelefteq \ntrianglelefteq	\ntriangleright \ntriangleright	\ntrianglerighteq \ntrianglerighteq	\nvDash \nvDash
\nvdash \nvdash	\odot \odot	\ominus \ominus	\oplus \oplus
\oslash \oslash	\otimes \otimes	\parallel \parallel	\perp \perp
\pitchfork \pitchfork	\pm \pm	\prec \prec	\preccurlyeq \preccurlyeq
\preccurlyeq \preccurlyeq	\preceq \preceq	\preccurlyeq \preccurlyeq	\precsim \precsim
\precsim \precsim	\propto \propto	\rightthreetimes \rightthreetimes	\risingdotseq \risingdotseq
\rtimes \rtimes	\sim \sim	\simeq \simeq	$/$ \slash
\smile \smile	\sqcap \sqcap	\sqcup \sqcup	\sqsubset \sqsubset
\sqsubset \sqsubset	\sqsubseteq \sqsubseteq	\sqsupset \sqsupset	\sqsupseteq \sqsupseteq
\sqsupseteq \sqsupseteq	\star \star	\subset \subset	\subseteq \subseteq
\subseteq \subseteq	\subsetneq \subsetneq	\subsetneqq \subsetneqq	$>$ \succ
\succapprox \succapprox	\succcurlyeq \succcurlyeq	\succeq \succeq	\succapprox \succapprox
\succsim \succsim	\succsim \succsim	\supset \supset	\supseteq \supseteq
\supseteq \supseteq	\supsetneq \supsetneq	\supsetneqq \supsetneqq	\therefore \therefore
\times \times	\top \top	\triangleleft \triangleleft	\trianglelefteq \trianglelefteq
\triangleq \triangleq	\triangleright \triangleright	\trianglerighteq \trianglerighteq	\uplus \uplus
\vDash \vDash	\varpropto \varpropto	\vartriangleleft \vartriangleleft	\vartriangleright \vartriangleright
\vdash \vdash	\vee \vee	\veebar \veebar	\wedge \wedge

Continued on next page

Table 2 – continued from previous page

\wr <code>\wr</code>			
------------------------	--	--	--

Arrow symbols

\Downarrow <code>\Downarrow</code>	\Leftarrow <code>\Leftarrow</code>	\Leftrightarrow <code>\Leftrightarrow</code>	\Lleftarrow <code>\Lleftarrow</code>
\Longleftarrow <code>\Longleftarrow</code>	\Longleftrightarrow <code>\Longleftrightarrow</code>	\Longrightarrow <code>\Longrightarrow</code>	\Lsh <code>\Lsh</code>
\nearrow <code>\nearrow</code>	\nrightarrow <code>\nrightarrow</code>	\Rightarrow <code>\Rightarrow</code>	\Rightarrow <code>\Rightarrow</code>
\Rsh <code>\Rsh</code>	\searrow <code>\searrow</code>	\swarrow <code>\swarrow</code>	\Uparrow <code>\Uparrow</code>
\Updownarrow <code>\Updownarrow</code>	\circlearrowleft <code>\circlearrowleft</code>	\circlearrowright <code>\circlearrowright</code>	\curvearrowleft <code>\curvearrowleft</code>
\curvearrowright <code>\curvearrowright</code>	\dashleftarrow <code>\dashleftarrow</code>	\dashrightarrow <code>\dashrightarrow</code>	\downarrow <code>\downarrow</code>
\Downarrow <code>\Downarrow</code>	\downharpoonleft <code>\downharpoonleft</code>	\downharpoonright <code>\downharpoonright</code>	\hookleftarrow <code>\hookleftarrow</code>
\hookrightarrow <code>\hookrightarrow</code>	\leadsto <code>\leadsto</code>	\leftarrow <code>\leftarrow</code>	\leftarrowtail <code>\leftarrowtail</code>
\leftharpoonup <code>\leftharpoonup</code>	\leftharpoonup <code>\leftharpoonup</code>	\leftleftarrows <code>\leftleftarrows</code>	\leftrightarrows <code>\leftrightarrows</code>
\leftrightsquigarrow <code>\leftrightsquigarrow</code>	\leftrightsquigarrow <code>\leftrightsquigarrow</code>	\leftrightsquigarrow <code>\leftrightsquigarrow</code>	\leftrightsquigarrow <code>\leftrightsquigarrow</code>
\longleftarrow <code>\longleftarrow</code>	\longleftrightarrow <code>\longleftrightarrow</code>	\longmapsto <code>\longmapsto</code>	\longrightarrow <code>\longrightarrow</code>
\looparrowleft <code>\looparrowleft</code>	\looparrowright <code>\looparrowright</code>	\mapsto <code>\mapsto</code>	\multimap <code>\multimap</code>
\nLeftarrow <code>\nLeftarrow</code>	\nLeftrightarrow <code>\nLeftrightarrow</code>	\nRightarrow <code>\nRightarrow</code>	\nearrow <code>\nearrow</code>
\nleftarrow <code>\nleftarrow</code>	\nleftrightarrow <code>\nleftrightarrow</code>	\rightarrow <code>\rightarrow</code>	\nrightarrow <code>\nrightarrow</code>
\rightarrow <code>\rightarrow</code>	\rightarrowtail <code>\rightarrowtail</code>	\rightharpoonup <code>\rightharpoonup</code>	\rightharpoonup <code>\rightharpoonup</code>
\rightleftarrows <code>\rightleftarrows</code>	\rightleftarrows <code>\rightleftarrows</code>	\rightleftarrows <code>\rightleftarrows</code>	\rightleftarrows <code>\rightleftarrows</code>
\rightrightarrows <code>\rightrightarrows</code>	\rightrightarrows <code>\rightrightarrows</code>	\rightsquigarrow <code>\rightsquigarrow</code>	\searrow <code>\searrow</code>
\swarrow <code>\swarrow</code>	\rightarrow <code>\rightarrow</code>	\twoheadleftarrow <code>\twoheadleftarrow</code>	\twoheadrightarrow <code>\twoheadrightarrow</code>
\uparrow <code>\uparrow</code>	\updownarrow <code>\updownarrow</code>	\updownarrow <code>\updownarrow</code>	\upharpoonleft <code>\upharpoonleft</code>
\upharpoonright <code>\upharpoonright</code>	\upuparrows <code>\upuparrows</code>		

Miscellaneous symbols

$\$ \backslash \$$	$\backslash AA$	$\backslash Finv$	$\backslash Game$
$\Im \backslash Im$	$\P \backslash P$	$\Re \backslash Re$	$\S \backslash S$
$\angle \backslash angle$	$\backprime \backslash backprime$	$\bigstar \backslash bigstar$	$\blacksquare \backslash blacksquare$
$\blacktriangle \backslash blacktriangle$	$\blacktriangledown \backslash blacktriangledown$	$\cdots \backslash cdots$	$\checkmark \backslash checkmark$
$\textcircled{R} \backslash circledR$	$\textcircled{S} \backslash circledS$	$\clubsuit \backslash clubsuit$	$\complement \backslash complement$
$\copyright \backslash copyright$	$\ddots \backslash ddots$	$\diamondsuit \backslash diamondsuit$	$\ell \backslash ell$
$\emptyset \backslash emptyset$	$\eth \backslash eth$	$\exists \backslash exists$	$\flat \backslash flat$
$\forall \backslash forall$	$\hbar \backslash hbar$	$\heartsuit \backslash heartsuit$	$\hslash \backslash hslash$
$\iiint \backslash iint$	$\iint \backslash iint$	$\iint \backslash iint$	$\imath \backslash imath$
$\infty \backslash infty$	$\jmath \backslash jmath$	$\ldots \backslash ldots$	$\measuredangle \backslash measuredangle$
$\natural \backslash natural$	$\neg \backslash neg$	$\nexists \backslash nexists$	$\oiint \backslash oiiint$
$\partial \backslash partial$	$\prime \backslash prime$	$\sharp \backslash sharp$	$\spadesuit \backslash spadesuit$
$\sphericalangle \backslash sphericalangle$	$\ss \backslash ss$	$\triangledown \backslash triangledown$	$\varnothing \backslash varnothing$
$\vartriangle \backslash vartriangle$	$\vdots \backslash vdots$	$\wp \backslash wp$	$\yen \backslash yen$

If a particular symbol does not have a name (as is true of many of the more obscure symbols in the STIX fonts), Unicode characters can also be used:

```
ur'$\u23ce$'
```

Example

Here is an example illustrating many of these features in context.

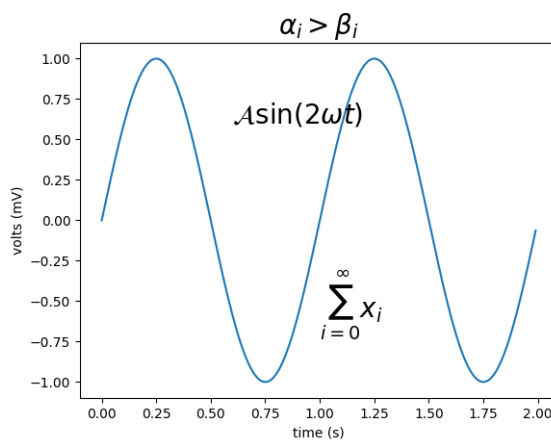


Fig. 47: Pyplot Mathtext

Note: [Click here to download the full example code](#)

3.5.7 Text properties and layout

Controlling properties of text and its layout with Matplotlib.

The `matplotlib.text.Text` instances have a variety of properties which can be configured via keyword arguments to the text commands (e.g., `title()`, `xlabel()` and `text()`).

Property	Value Type
<code>alpha</code>	<code>float</code>
<code>backgroundcolor</code>	any matplotlib <code>color</code>
<code>bbox</code>	Rectangle prop dict plus key 'pad' which is a pad in points
<code>clip_box</code>	a matplotlib.transform.Bbox instance
<code>clip_on</code>	<code>bool</code>
<code>clip_path</code>	a Path instance and a Transform instance, a Patch
<code>color</code>	any matplotlib <code>color</code>
<code>family</code>	['serif' 'sans-serif' 'cursive' 'fantasy' 'monospace']
<code>fontproperties</code>	a FontProperties instance
<code>horizontalalignment</code> or <code>ha</code>	['center' 'right' 'left']
<code>label</code>	any string
<code>linespacing</code>	<code>float</code>
<code>multialignment</code>	['left' 'right' 'center']
<code>name</code> or <code>fontname</code>	string e.g., ['Sans' 'Courier' 'Helvetica' ...]
<code>picker</code>	[None float boolean callable]
<code>position</code>	(x, y)
<code>rotation</code>	[angle in degrees 'vertical' 'horizontal']
<code>size</code> or <code>fontsize</code>	[size in points relative size, e.g., 'smaller', 'x-large']
<code>style</code> or <code>fontstyle</code>	['normal' 'italic' 'oblique']
<code>text</code>	string or anything printable with '%s' conversion
<code>transform</code>	a Transform instance
<code>variant</code>	['normal' 'small-caps']
<code>verticalalignment</code> or <code>va</code>	['center' 'top' 'bottom' 'baseline']
<code>visible</code>	<code>bool</code>
<code>weight</code> or <code>fontweight</code>	['normal' 'bold' 'heavy' 'light' 'ultrabold' 'ultralight']
<code>x</code>	<code>float</code>
<code>y</code>	<code>float</code>
<code>zorder</code>	any number

You can lay out text with the alignment arguments `horizontalalignment`, `verticalalignment`, and `multialignment`. `horizontalalignment` controls whether the x positional argument for the text indicates the left, center or right side of the text bounding box. `verticalalignment` controls whether the y positional argument for the text indicates the bottom, center or top side of the text bounding box. `multialignment`, for newline separated strings only, controls whether the different lines are left, center or right justified. Here is an example which uses the `text()` command to show the various alignment possibilities. The use of `transform=ax.transAxes` throughout the code indicates that the coordinates are given relative to the axes bounding box, with 0,0 being the lower left of the axes and 1,1 the upper right.

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# build a rectangle in axes coords
left, width = .25, .5
bottom, height = .25, .5
right = left + width
top = bottom + height

fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])

# axes coordinates are 0,0 is bottom left and 1,1 is upper right
p = patches.Rectangle(
    (left, bottom), width, height,
    fill=False, transform=ax.transAxes, clip_on=False
)

ax.add_patch(p)

ax.text(left, bottom, 'left top',
        horizontalalignment='left',
        verticalalignment='top',
        transform=ax.transAxes)

ax.text(left, bottom, 'left bottom',
        horizontalalignment='left',
        verticalalignment='bottom',
        transform=ax.transAxes)

ax.text(right, top, 'right bottom',
        horizontalalignment='right',
        verticalalignment='bottom',
        transform=ax.transAxes)

ax.text(right, top, 'right top',
        horizontalalignment='right',
        verticalalignment='top',
        transform=ax.transAxes)

ax.text(right, bottom, 'center top',
        horizontalalignment='center',
        verticalalignment='top',
        transform=ax.transAxes)

ax.text(left, 0.5*(bottom+top), 'right center',
```

(continues on next page)

(continued from previous page)

```
horizontalalignment='right',
verticalalignment='center',
rotation='vertical',
transform=ax.transAxes)

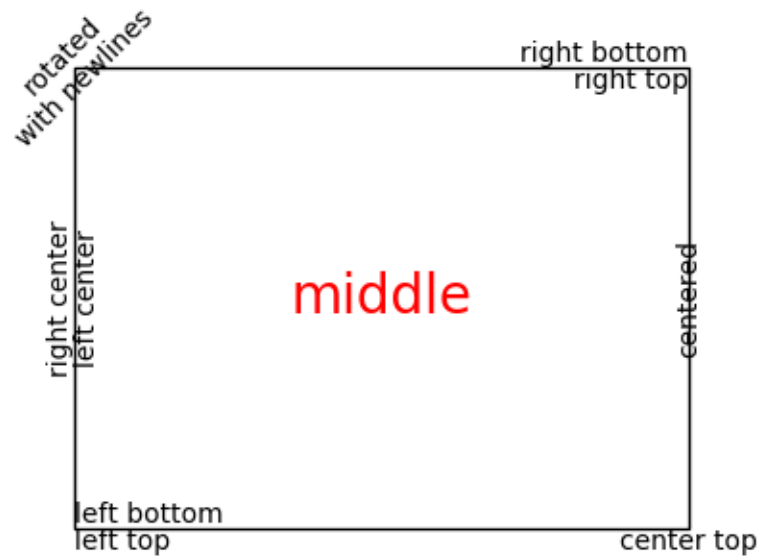
ax.text(left, 0.5*(bottom+top), 'left center',
        horizontalalignment='left',
        verticalalignment='center',
        rotation='vertical',
        transform=ax.transAxes)

ax.text(0.5*(left+right), 0.5*(bottom+top), 'middle',
        horizontalalignment='center',
        verticalalignment='center',
        fontsize=20, color='red',
        transform=ax.transAxes)

ax.text(right, 0.5*(bottom+top), 'centered',
        horizontalalignment='center',
        verticalalignment='center',
        rotation='vertical',
        transform=ax.transAxes)

ax.text(left, top, 'rotated\nwith newlines',
        horizontalalignment='center',
        verticalalignment='center',
        rotation=45,
        transform=ax.transAxes)

ax.set_axis_off()
plt.show()
```



3.5.8 Default Font

The base default font is controlled by a set of rcParams. To set the font for mathematical expressions, use the rcParams beginning with `mathtext` (see [mathtext](#)).

rcParam	usage
'font.family'	List of either names of font or {'cursive', 'fantasy', 'monospace', 'sans', 'sans serif', 'sans-serif', 'serif'}.
'font.style'	The default style, ex 'normal', 'italic'.
'font.variant'	Default variant, ex 'normal', 'small-caps' (untested)
'font.stretch'	Default stretch, ex 'normal', 'condensed' (incomplete)
'font.weight'	Default weight. Either string or integer
'font.size'	Default font size in points. Relative font sizes ('large', 'x-small') are computed against this size.

The mapping between the family aliases ({'cursive', 'fantasy', 'monospace', 'sans', 'sans serif', 'sans-

'serif', 'serif'}) and actual font names is controlled by the following rcParams:

family alias	rcParam with mappings
'serif'	'font.serif'
'monospace'	'font.monospace'
'fantasy'	'font.fantasy'
'cursive'	'font.cursive'
{'sans', 'sans serif', 'sans-serif'}	'font.sans-serif'

which are lists of font names.

Text with non-latin glyphs

As of v2.0 the default font contains glyphs for many western alphabets, but still does not cover all of the glyphs that may be required by mpl users. For example, DejaVu has no coverage of Chinese, Korean, or Japanese.

To set the default font to be one that supports the code points you need, prepend the font name to 'font.family' or the desired alias lists

```
matplotlib.rcParams['font.sans-serif'] = ['Source Han Sans TW', 'sans-serif']
```

or set it in your .matplotlibrc file:

```
font.sans-serif: Source Han Sans TW, Arial, sans-serif
```

To control the font used on per-artist basis use the 'name', 'fontname' or 'fontproperties' kwargs documented [above](#).

On linux, [fc-list](#) can be a useful tool to discover the font name; for example

```
$ fc-list :lang=zh family
Noto to Sans Mono CJK TC,Noto Sans Mono CJK TC Bold
Noto Sans CJK TC,Noto Sans CJK TC Medium
Noto Sans CJK TC,Noto Sans CJK TC DemiLight
Noto Sans CJK KR,Noto Sans CJK KR Black
Noto Sans CJK TC,Noto Sans CJK TC Black
Noto Sans Mono CJK TC,Noto Sans Mono CJK TC Regular
Noto Sans CJK SC,Noto Sans CJK SC Light
```

lists all of the fonts that support Chinese.

Note: [Click here to download the full example code](#)

3.5.9 Text in Matplotlib Plots

Introduction to plotting and working with text in Matplotlib.

Matplotlib has extensive text support, including support for mathematical expressions, truetype support for raster and vector outputs, newline separated text with arbitrary rotations, and unicode support.

Because it embeds fonts directly in output documents, e.g., for postscript or PDF, what you see on the screen is what you get in the hardcopy. [FreeType](#) support produces very nice, antialiased fonts, that look good even at small raster sizes. matplotlib includes its own `matplotlib.font_manager` (thanks to Paul Barrett), which implements a cross platform, W3C compliant font finding algorithm.

The user has a great deal of control over text properties (font size, font weight, text location and color, etc.) with sensible defaults set in the rc file. And significantly, for those interested in mathematical or scientific figures, matplotlib implements a large number of TeX math symbols and commands, supporting [mathematical expressions](#) anywhere in your figure.

Basic text commands

The following commands are used to create text in the pyplot interface and the object-oriented API:

pyplot API	OO API	description
<code>text</code>	<code>text</code>	Add text at an arbitrary location of the Axes.
<code>annotate</code>	<code>annotate</code>	Add an annotation, with an optional arrow, at an arbitrary location of the Axes.
<code>xlabel</code>	<code>set_xlabel</code>	Add a label to the Axes's x-axis.
<code>ylabel</code>	<code>set_ylabel</code>	Add a label to the Axes's y-axis.
<code>title</code>	<code>set_title</code>	Add a title to the Axes.
<code>figtext</code>	<code>text</code>	Add text at an arbitrary location of the Figure.
<code>suptitle</code>	<code>suptitle</code>	Add a title to the Figure.

All of these functions create and return a `Text` instance, which can be configured with a variety of font and other properties. The example below shows all of these commands in action, and more detail is provided in the sections that follow.

```
import matplotlib
import matplotlib.pyplot as plt

fig = plt.figure()
fig.suptitle('bold figure supitle', fontsize=14, fontweight='bold')

ax = fig.add_subplot(111)
fig.subplots_adjust(top=0.85)
```

(continues on next page)

(continued from previous page)

```
ax.set__title('axes title')

ax.set__xlabel('xlabel')
ax.set__ylabel('ylabel')

ax.text(3, 8, 'boxed italics text in data coords', style='italic',
        bbox={'facecolor': 'red', 'alpha': 0.5, 'pad': 10})

ax.text(2, 6, r'an equation:  $E=mc^2$ ', fontsize=15)

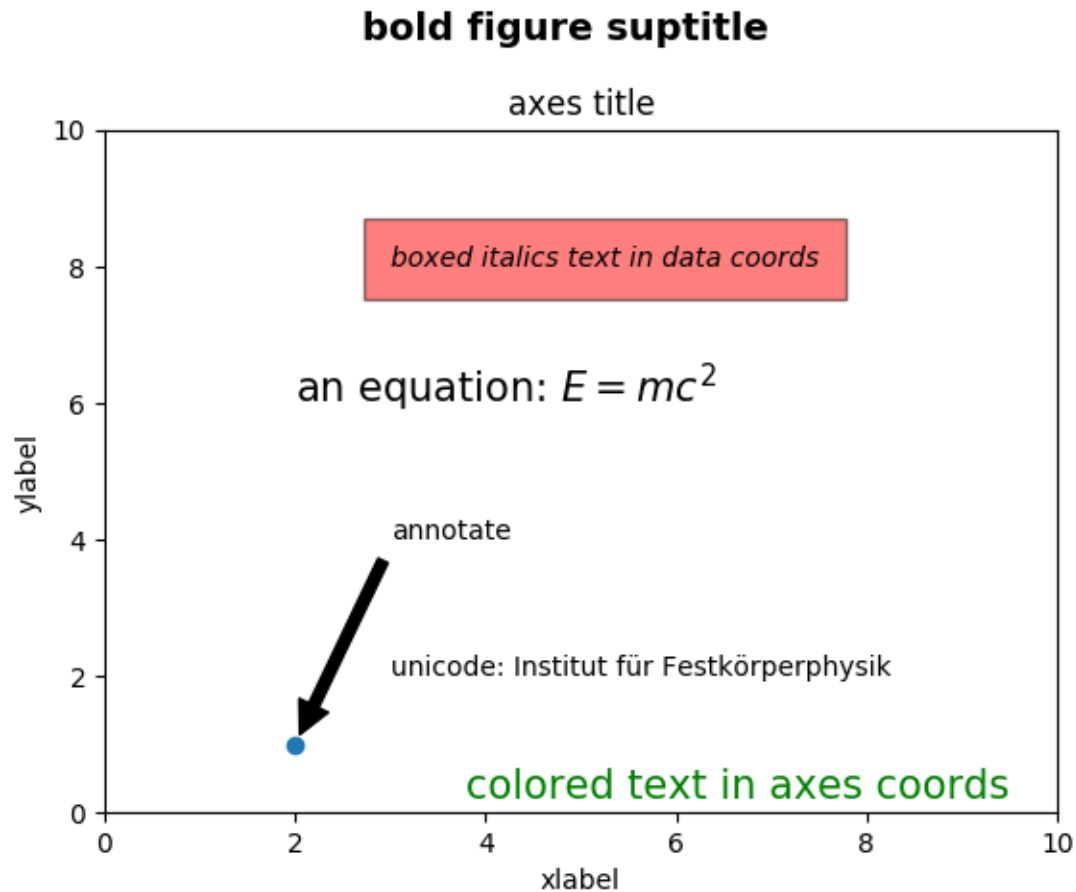
ax.text(3, 2, u'unicode: Institut für Festkörperphysik')

ax.text(0.95, 0.01, 'colored text in axes coords',
        verticalalignment='bottom', horizontalalignment='right',
        transform=ax.transAxes,
        color='green', fontsize=15)

ax.plot([2], [1], 'o')
ax.annotate('annotate', xy=(2, 1), xytext=(3, 4),
            arrowprops=dict(facecolor='black', shrink=0.05))

ax.axis([0, 10, 0, 10])

plt.show()
```



Labels for x- and y-axis

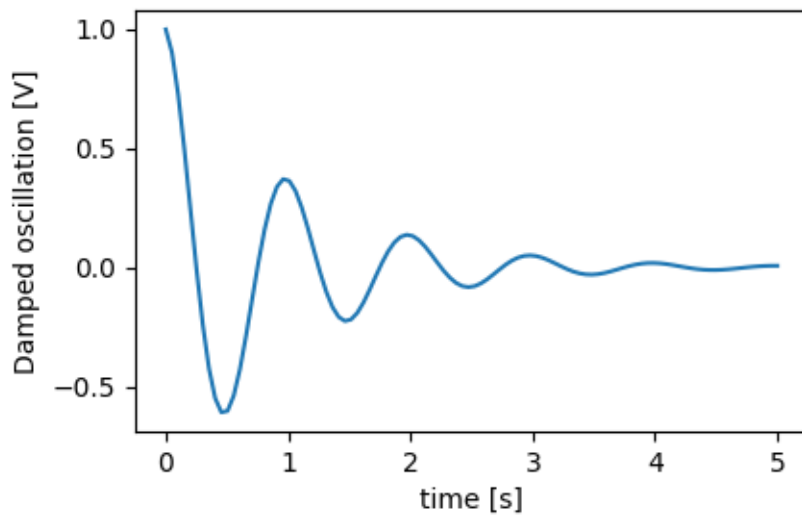
Specifying the labels for the x- and y-axis is straightforward, via the `set_xlabel` and `set_ylabel` methods.

```
import matplotlib.pyplot as plt
import numpy as np

x1 = np.linspace(0.0, 5.0, 100)
y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)

fig, ax = plt.subplots(figsize=(5, 3))
fig.subplots_adjust(bottom=0.15, left=0.2)
ax.plot(x1, y1)
ax.set_xlabel('time [s]')
ax.set_ylabel('Damped oscillation [V]')

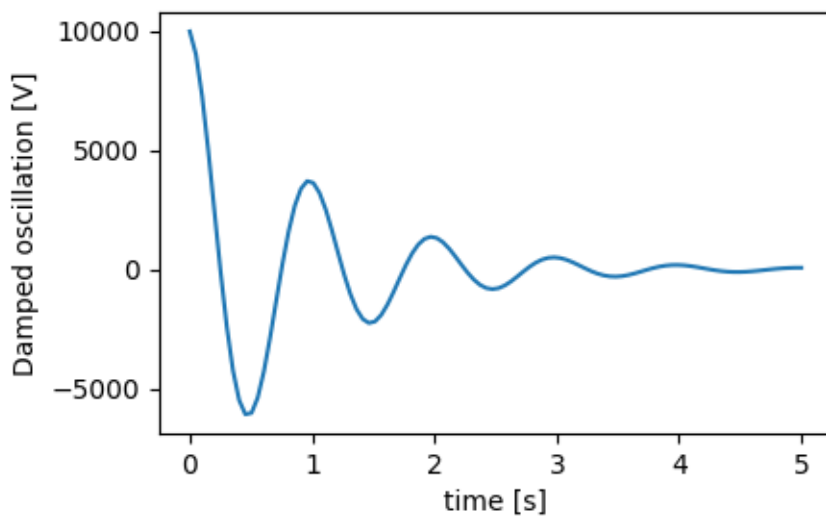
plt.show()
```



The x- and y-labels are automatically placed so that they clear the x- and y-ticklabels. Compare the plot below with that above, and note the y-label is to the left of the one above.

```
fig, ax = plt.subplots(figsize=(5, 3))
fig.subplots_adjust(bottom=0.15, left=0.2)
ax.plot(x1, y1*10000)
ax.set_xlabel('time [s]')
ax.set_ylabel('Damped oscillation [V]')

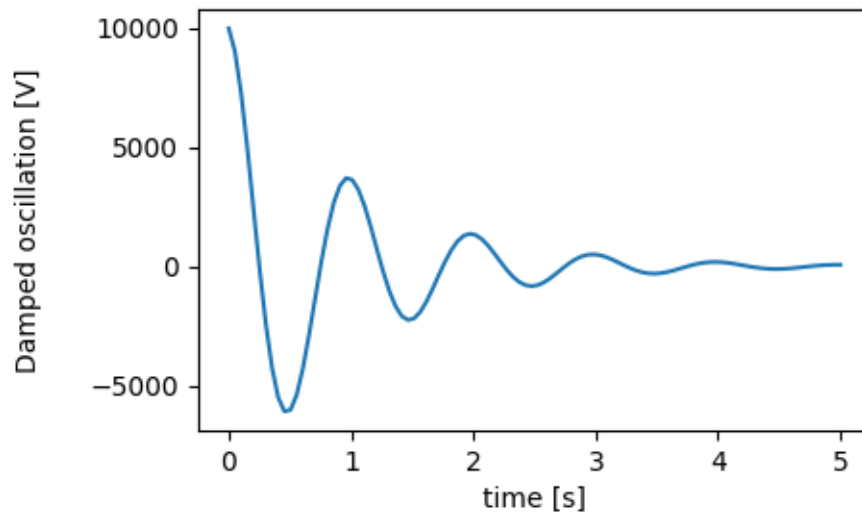
plt.show()
```



If you want to move the labels, you can specify the `labelpad` keyword argument, where the value is points ($1/72''$, the same unit used to specify fontsizes).

```
fig, ax = plt.subplots(figsize=(5, 3))
fig.subplots_adjust(bottom=0.15, left=0.2)
ax.plot(x1, y1*10000)
ax.set_xlabel('time [s]')
ax.set_ylabel('Damped oscillation [V]', labelpad=18)

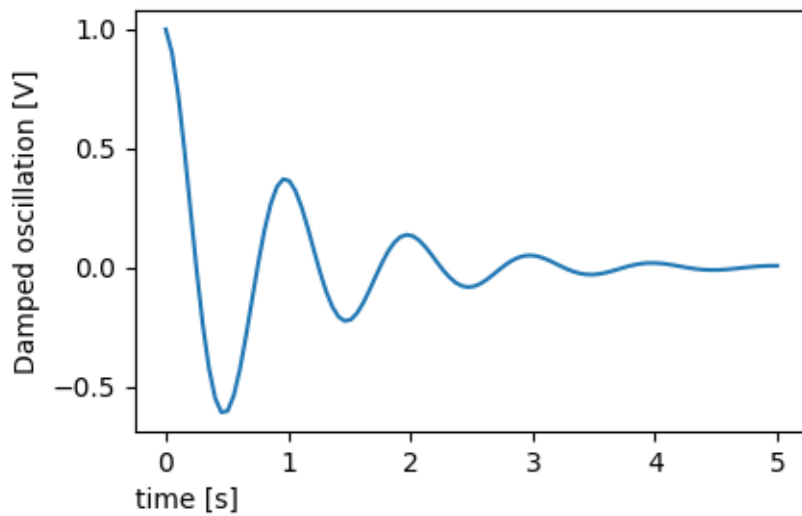
plt.show()
```



Or, the labels accept all the Text keyword arguments, including position, via which we can manually specify the label positions. Here we put the xlabel to the far left of the axis. Note, that the y-coordinate of this position has no effect - to adjust the y-position we need to use the `labelpad` kwarg.

```
fig, ax = plt.subplots(figsize=(5, 3))
fig.subplots_adjust(bottom=0.15, left=0.2)
ax.plot(x1, y1)
ax.set_xlabel('time [s]', position=(0., 1e6),
              horizontalalignment='left')
ax.set_ylabel('Damped oscillation [V]')

plt.show()
```



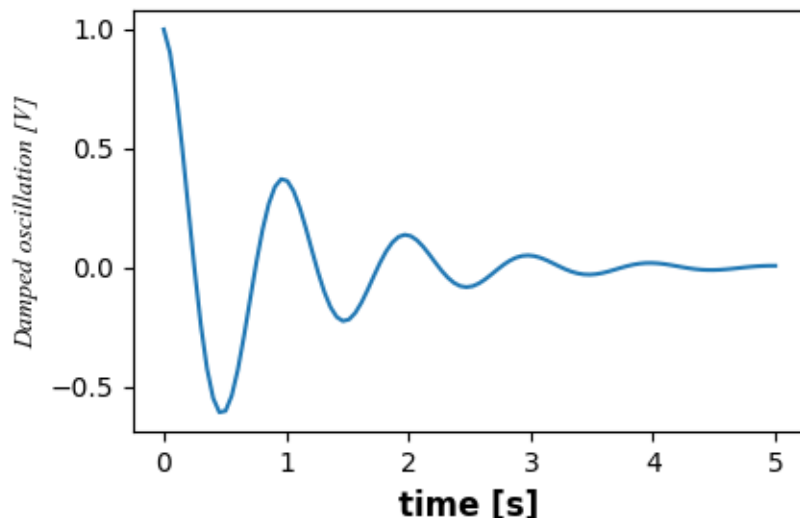
All the labelling in this tutorial can be changed by manipulating the `matplotlib.font_manager.FontProperties` method, or by named kwargs to `set_xlabel`

```
from matplotlib.font_manager import FontProperties

font = FontProperties()
font.set_family('serif')
font.set_name('Times New Roman')
font.set_style('italic')

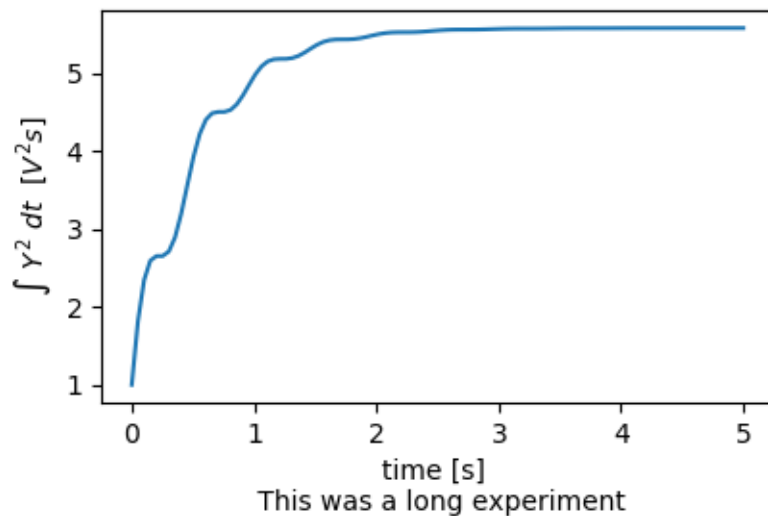
fig, ax = plt.subplots(figsize=(5, 3))
fig.subplots_adjust(bottom=0.15, left=0.2)
ax.plot(x1, y1)
ax.set_xlabel('time [s]', fontsize='large', fontweight='bold')
ax.set_ylabel('Damped oscillation [V]', fontproperties=font)

plt.show()
```



Finally, we can use native TeX rendering in all text objects and have multiple lines:

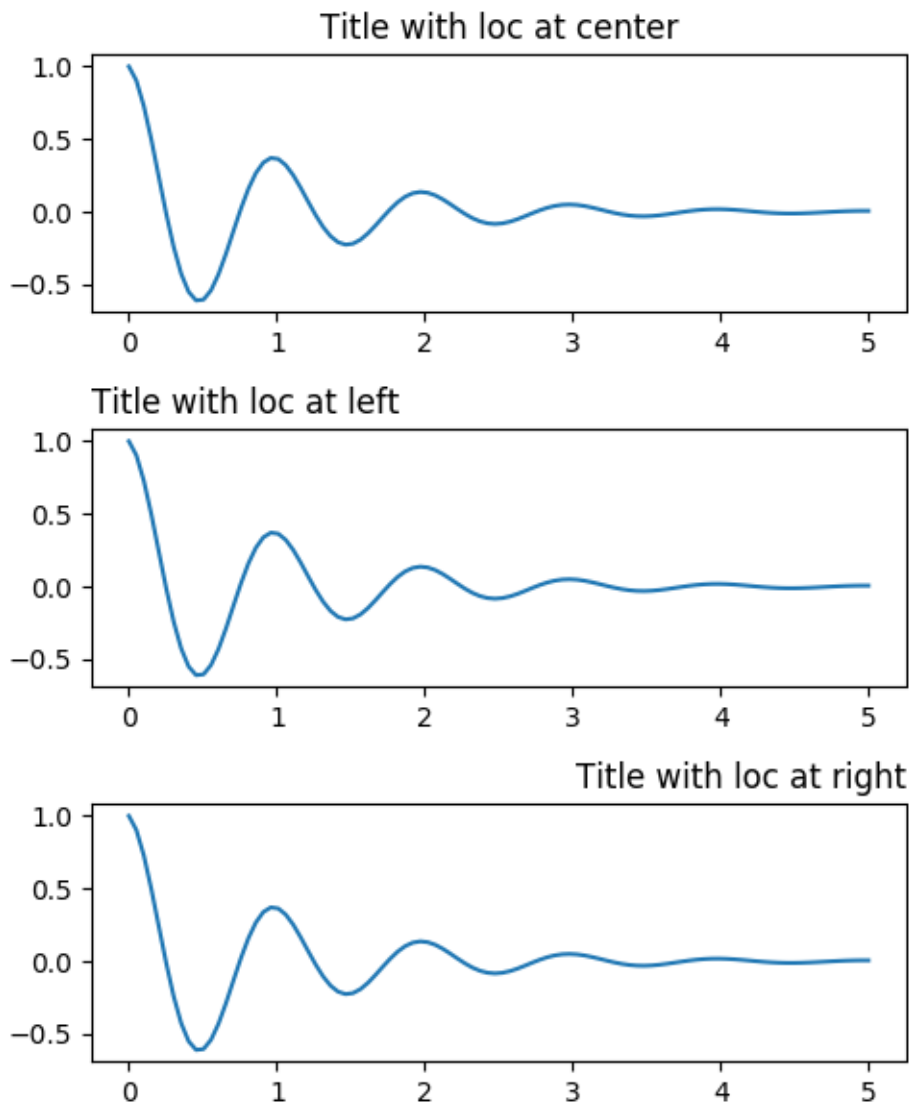
```
fig, ax = plt.subplots(figsize=(5, 3))
fig.subplots_adjust(bottom=0.2, left=0.2)
ax.plot(x1, np.cumsum(y1**2))
ax.set_xlabel('time [s] \n This was a long experiment')
ax.set_ylabel(r'$\int Y^2 dt \ [V^2 s]$')
plt.show()
```



Titles

Subplot titles are set in much the same way as labels, but there is the `loc` keyword arguments that can change the position and justification from the default value of `loc=center`.

```
fig, axs = plt.subplots(3, 1, figsize=(5, 6), tight_layout=True)
locs = ['center', 'left', 'right']
for ax, loc in zip(axs, locs):
    ax.plot(x1, y1)
    ax.set_title('Title with loc at '+loc, loc=loc)
plt.show()
```



Vertical spacing for titles is controlled via `rcParams[axes.titlepad]`, which defaults to 5 points. Setting to a different value moves the title.

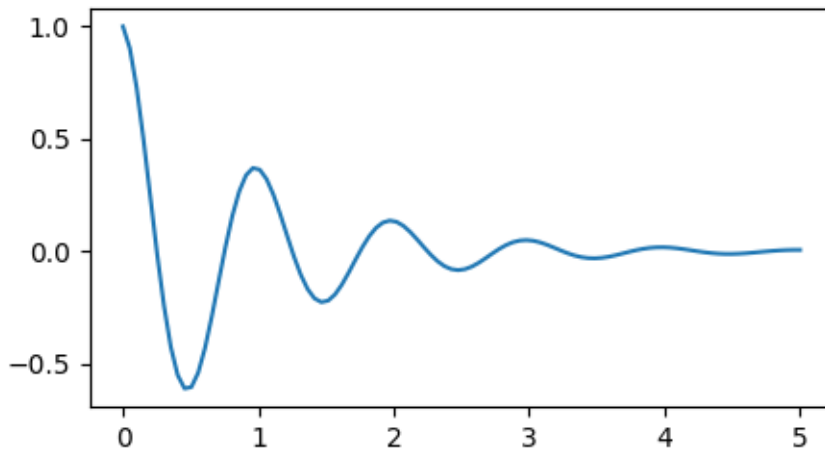
```
fig, ax = plt.subplots(figsize=(5, 3))
fig.subplots_adjust(top=0.8)
ax.plot(x1, y1)
ax.set_title('Vertically offset title', pad=30)
```

(continues on next page)

(continued from previous page)

```
plt.show()
```

Vertically offset title



Ticks and ticklabels

Placing ticks and ticklabels is a very tricky aspect of making a figure. Matplotlib does the best it can automatically, but it also offers a very flexible framework for determining the choices for tick locations, and how they are labelled.

Terminology

Axes have an `matplotlib.axis` object for the `ax.xaxis` and `ax.yaxis` that contain the information about how the labels in the axis are laid out.

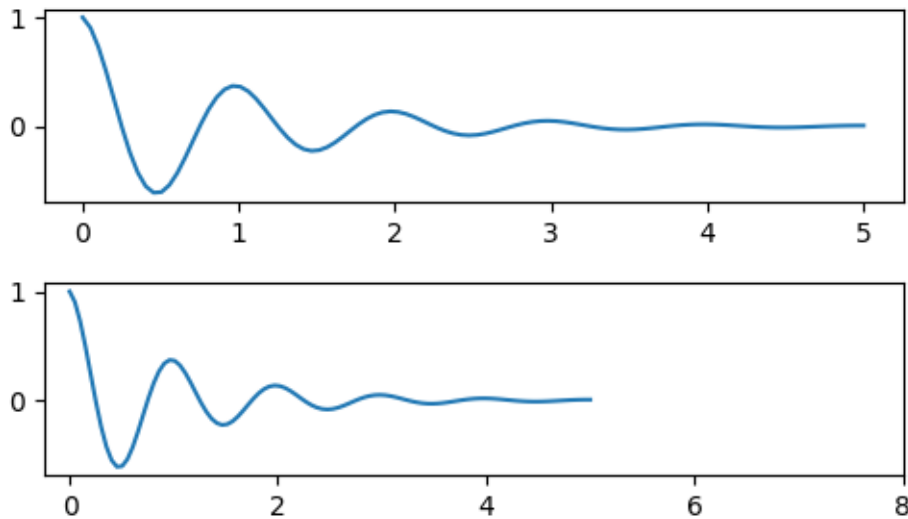
The axis API is explained in detail in the documentation to `axis`.

An `Axis` object has major and minor ticks. The `Axis` has a `matplotlib.xaxis.set_major_locator` and `matplotlib.xaxis.set_minor_locator` methods that use the data being plotted to determine the location of major and minor ticks. There are also `matplotlib.xaxis.set_major_formatter` and `matplotlib.xaxis.set_minor_formatter` methods that format the tick labels.

Simple ticks

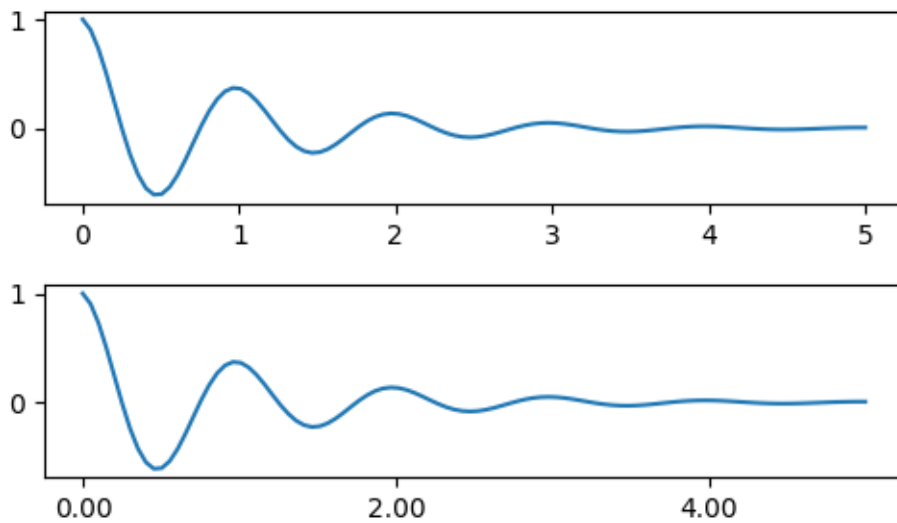
It often is convenient to simply define the tick values, and sometimes the tick labels, overriding the default locators and formatters. This is discouraged because it breaks interactive navigation of the plot. It also can reset the axis limits: note that the second plot has the ticks we asked for, including ones that are well outside the automatic view limits.

```
fig, axs = plt.subplots(2, 1, figsize=(5, 3), tight_layout=True)
axs[0].plot(x1, y1)
axs[1].plot(x1, y1)
axs[1].xaxis.set_ticks(np.arange(0., 8.1, 2.))
plt.show()
```



We can of course fix this after the fact, but it does highlight a weakness of hard-coding the ticks. This example also changes the format of the ticks:

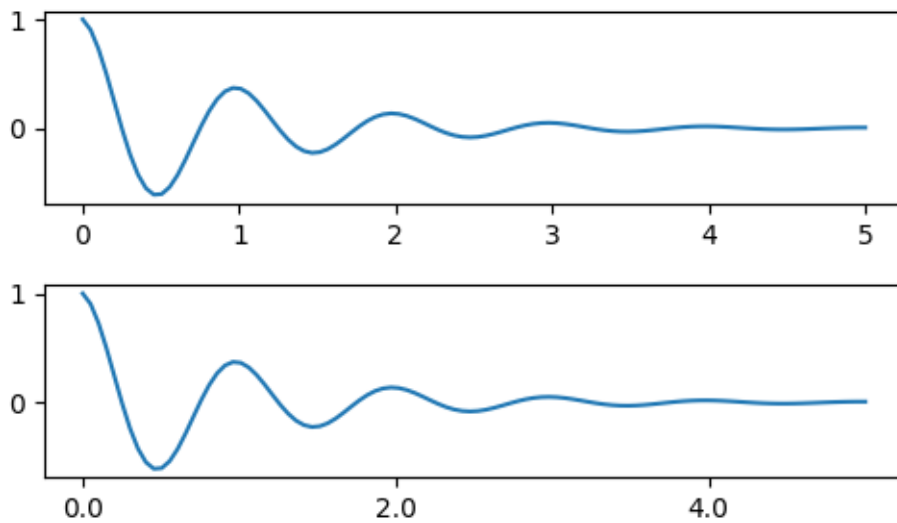
```
fig, axs = plt.subplots(2, 1, figsize=(5, 3), tight_layout=True)
axs[0].plot(x1, y1)
axs[1].plot(x1, y1)
ticks = np.arange(0., 8.1, 2.)
# list comprehension to get all tick labels...
tickla = ['%1.2f' % tick for tick in ticks]
axs[1].xaxis.set_ticks(ticks)
axs[1].xaxis.set_ticklabels(tickla)
axs[1].set_xlim(axs[0].get_xlim())
plt.show()
```



Tick Locators and Formatters

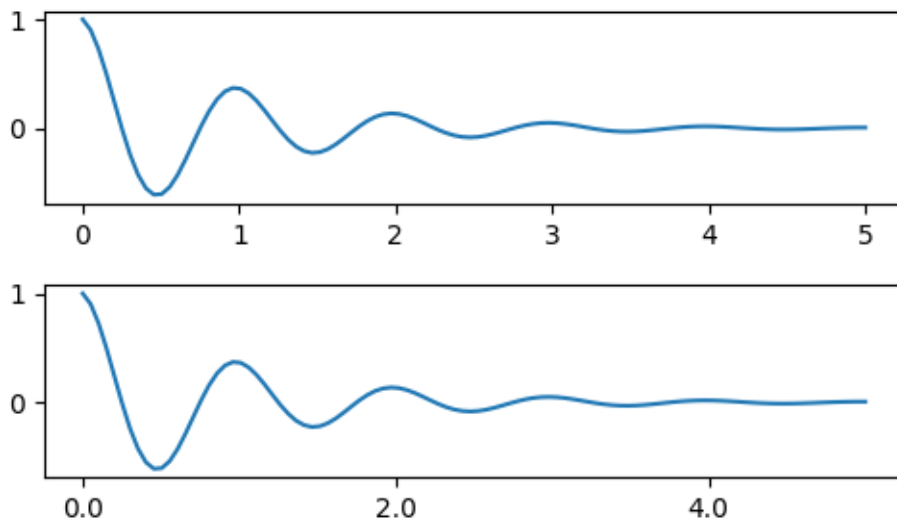
Instead of making a list of all the tick labels, we could have used a `matplotlib.ticker.FormatStrFormatter` and passed it to the `ax.xaxis`

```
fig, axs = plt.subplots(2, 1, figsize=(5, 3), tight_layout=True)
axs[0].plot(x1, y1)
axs[1].plot(x1, y1)
ticks = np.arange(0., 8.1, 2.)
# list comprehension to get all tick labels...
formatter = matplotlib.ticker.StrMethodFormatter('{x:1.1f}')
axs[1].xaxis.set_ticks(ticks)
axs[1].xaxis.set_major_formatter(formatter)
axs[1].set_xlim(axs[0].get_xlim())
plt.show()
```



And of course we could have used a non-default locator to set the tick locations. Note we still pass in the tick values, but the x-limit fix used above is not needed.

```
fig, axs = plt.subplots(2, 1, figsize=(5, 3), tight_layout=True)
axs[0].plot(x1, y1)
axs[1].plot(x1, y1)
formatter = matplotlib.ticker.FormatStrFormatter('%1.1f')
locator = matplotlib.ticker.FixedLocator(ticks)
axs[1].xaxis.set_major_locator(locator)
axs[1].xaxis.set_major_formatter(formatter)
plt.show()
```



The default formatter is the `matplotlib.ticker.MaxNLocator` called as `ticker.MaxNLocator(self, nbins='auto', steps=[1, 2, 2.5, 5, 10])`. The `steps` keyword contains a list of multiples that can be used for tick values. i.e. in this case, 2, 4, 6 would be acceptable ticks, as would 20, 40, 60 or

0.2, 0.4, 0.6. However, 3, 6, 9 would not be acceptable because 3 doesn't appear in the list of steps.

`nbins=auto` uses an algorithm to determine how many ticks will be acceptable based on how long the axis is. The fontsize of the ticklabel is taken into account, but the length of the tick string is not (because its not yet known.) In the bottom row, the ticklabels are quite large, so we set `nbins=4` to make the labels fit in the right-hand plot.

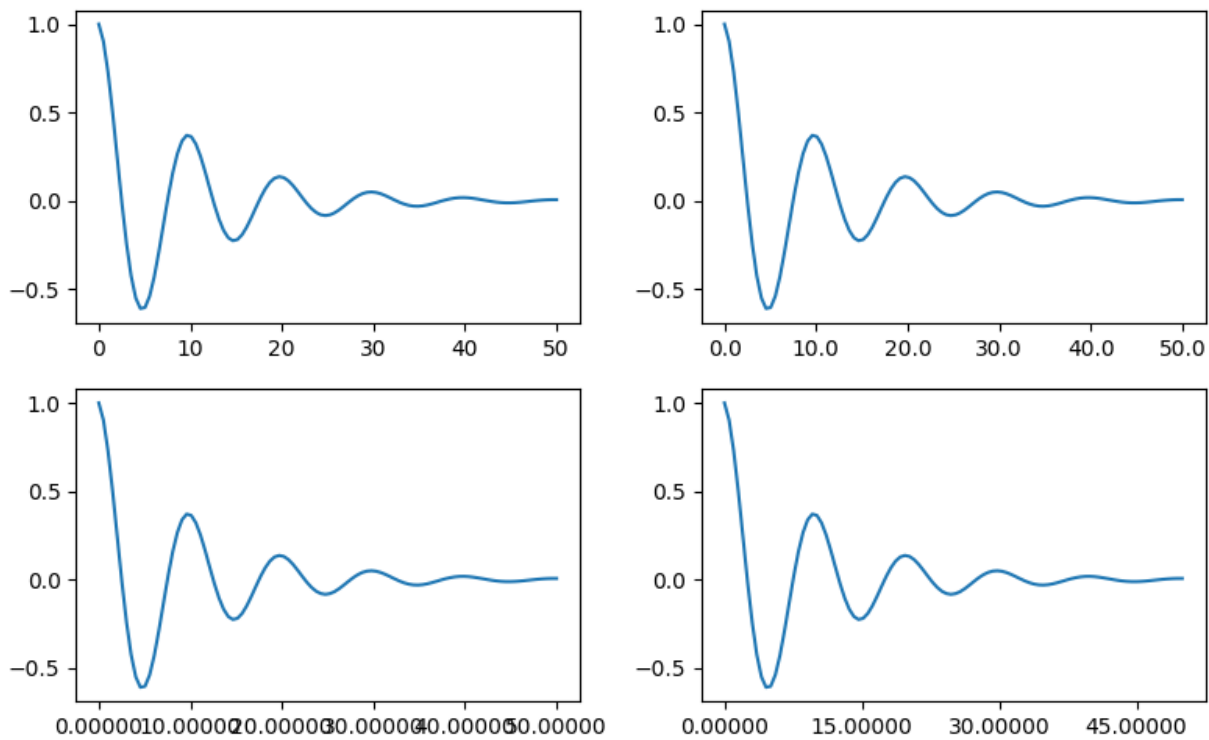
```
fig, axs = plt.subplots(2, 2, figsize=(8, 5), tight_layout=True)
axs = axs.flatten()
for n, ax in enumerate(axs):
    ax.plot(x1*10., y1)

formatter = matplotlib.ticker.FormatStrFormatter('%1.1f')
locator = matplotlib.ticker.MaxNLocator(nbins='auto', steps=[1, 4, 10])
axs[1].xaxis.set_major_locator(locator)
axs[1].xaxis.set_major_formatter(formatter)

formatter = matplotlib.ticker.FormatStrFormatter('%1.5f')
locator = matplotlib.ticker.AutoLocator()
axs[2].xaxis.set_major_formatter(formatter)
axs[2].xaxis.set_major_locator(locator)

formatter = matplotlib.ticker.FormatStrFormatter('%1.5f')
locator = matplotlib.ticker.MaxNLocator(nbins=4)
axs[3].xaxis.set_major_formatter(formatter)
axs[3].xaxis.set_major_locator(locator)

plt.show()
```

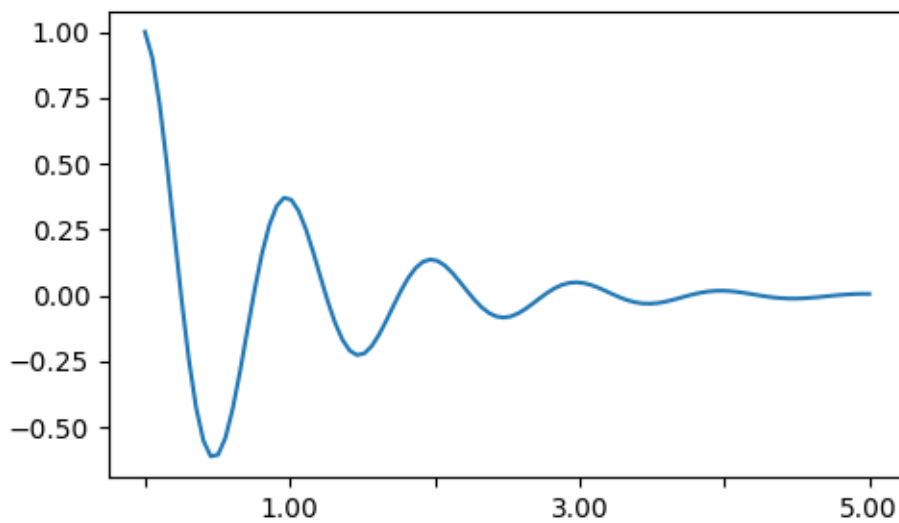


Finally, we can specify functions for the formatter using `matplotlib.ticker.FuncFormatter`.

```
def formatoddticks(x, pos):
    """Format odd tick positions
    """
    if x % 2:
        return '%1.2f' % x
    else:
        return ''
```

```
fig, ax = plt.subplots(figsize=(5, 3), tight_layout=True)
ax.plot(x1, y1)
formatter = matplotlib.ticker.FuncFormatter(formatoddticks)
locator = matplotlib.ticker.MaxNLocator(nbins=6)
ax.xaxis.set_major_formatter(formatter)
ax.xaxis.set_major_locator(locator)

plt.show()
```



Dateticks

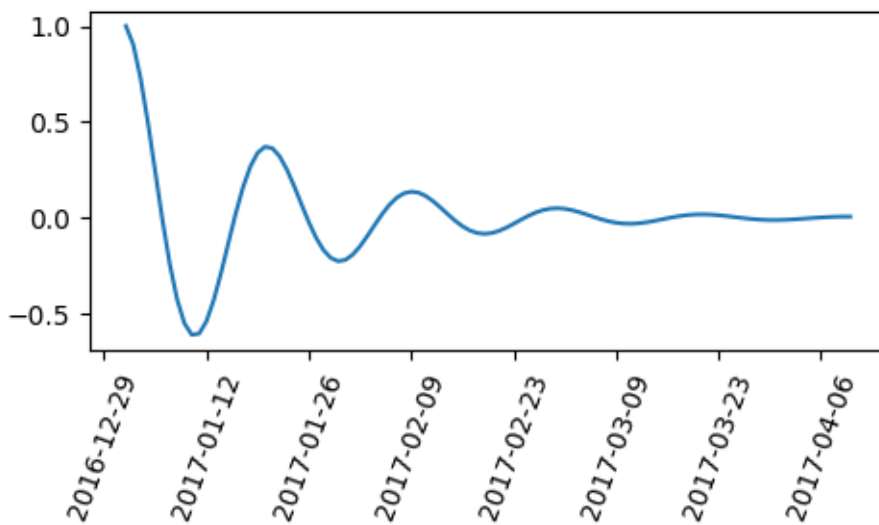
Matplotlib can accept `datetime.datetime` and `numpy.datetime64` objects as plotting arguments. Dates and times require special formatting, which can often benefit from manual intervention. In order to help, dates have special Locators and Formatters, defined in the `matplotlib.dates` module.

A simple example is as follows. Note how we have to rotate the tick labels so that they don't over-run each other.

```
import datetime

fig, ax = plt.subplots(figsize=(5, 3), tight_layout=True)
base = datetime.datetime(2017, 1, 1, 0, 0, 1)
time = [base + datetime.timedelta(days=x) for x in range(len(y1))]

ax.plot(time, y1)
ax.tick_params(axis='x', rotation=70)
plt.show()
```

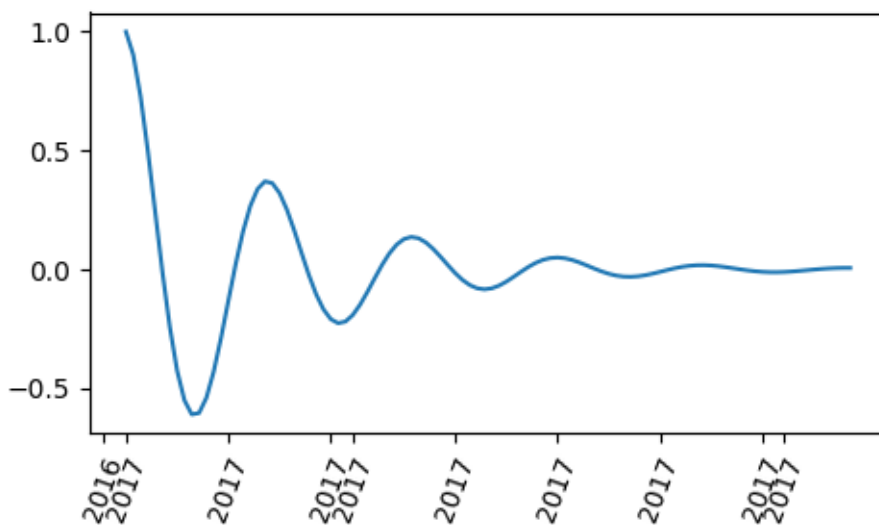


Maybe the format of the labels above is acceptable, but the choices is rather idiosyncratic. We can make the ticks fall on the start of the month by modifying `matplotlib.dates.AutoDateLocator`

```
import matplotlib.dates as mdates

locator = mdates.AutoDateLocator(interval_multiples=True)

fig, ax = plt.subplots(figsize=(5, 3), tight_layout=True)
ax.xaxis.set_major_locator(locator)
ax.plot(time, y1)
ax.tick_params(axis='x', rotation=70)
plt.show()
```

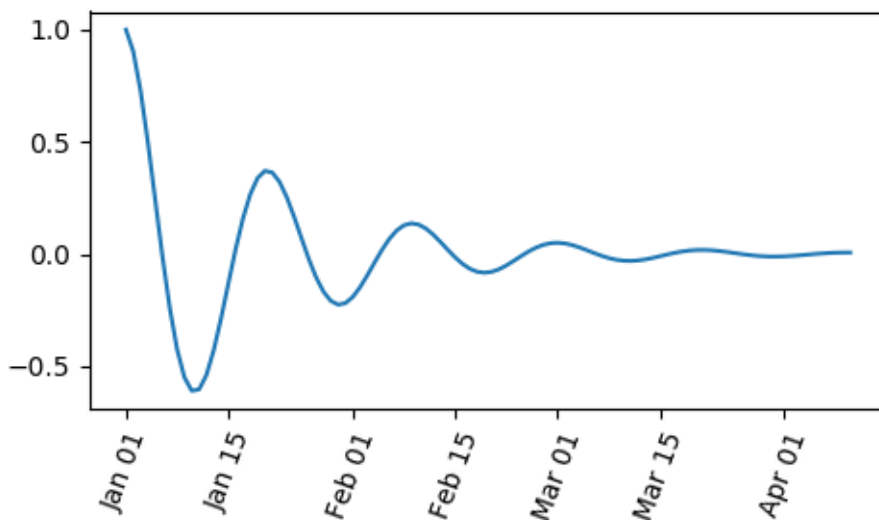


However, this changes the tick labels. The easiest fix is to pass a format to `matplotlib.dates.DateFormatter`. Also note that the 29th and the next month are very close together. We can fix

this by using the `dates.DayLocator` class, which allows us to specify a list of days of the month to use. Similar formatters are listed in the `matplotlib.dates` module.

```
locator = mdates.DayLocator(bymonthday=[1, 15])
formatter = mdates.DateFormatter('%b %d')

fig, ax = plt.subplots(figsize=(5, 3), tight_layout=True)
ax.xaxis.set_major_locator(locator)
ax.xaxis.set_major_formatter(formatter)
ax.plot(time, y1)
ax.tick_params(axis='x', rotation=70)
plt.show()
```



Legends and Annotations

- Legends: [Legend guide](#)
- Annotations: [Annotations](#)

3.6 Toolkits

These tutorials cover toolkits designed to extend the functionality of Matplotlib in order to accomplish specific goals.

Note: [Click here to download the full example code](#)

3.6.1 The mplot3d Toolkit

Generating 3D plots using the mplot3d toolkit.

Contents

- The mplot3d Toolkit
 - Getting started
 - * Line plots
 - * Scatter plots
 - * Wireframe plots
 - * Surface plots
 - * Tri-Surface plots
 - * Contour plots
 - * Filled contour plots
 - * Polygon plots
 - * Bar plots
 - * Quiver
 - * 2D plots in 3D
 - * Text
 - * Subplotting

Getting started

An Axes3D object is created just like any other axes using the `projection='3d'` keyword. Create a new `matplotlib.figure.Figure` and add a new axes to it of type `Axes3D`:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

New in version 1.0.0: This approach is the preferred method of creating a 3D axes.

Note: Prior to version 1.0.0, the method of creating a 3D axes was different. For those using older versions of matplotlib, change `ax = fig.add_subplot(111, projection='3d')` to `ax = Axes3D(fig)`.

See the mplot3d FAQ for more information about the mplot3d toolkit.

Line plots

`Axes3D.plot(xs, ys, *args, **kwargs)`
Plot 2D or 3D data.

Argument	Description
<code>xs, ys</code>	x, y coordinates of vertices
<code>zs</code>	z value(s), either one for all points or one for each point.
<code>zdir</code>	Which direction to use as z ('x', 'y' or 'z') when plotting a 2D set.

Other arguments are passed on to `plot()`

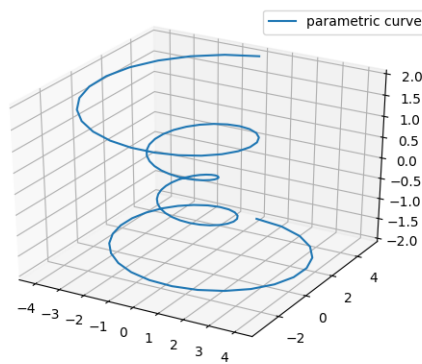


Fig. 48: Lines3d

Scatter plots

`Axes3D.scatter(xs, ys, zs=0, zdir='z', s=20, c=None, depthshade=True, *args, **kwargs)`
Create a scatter plot.

Argument	Description
xs, ys	Positions of data points.
zs	Either an array of the same length as xs and ys or a single value to place all points in the same plane. Default is 0.
zdir	Which direction to use as z ('x', 'y' or 'z') when plotting a 2D set.
s	Size in points ² . It is a scalar or an array of the same length as x and y.
c	A color. c can be a single color format string, or a sequence of color specifications of length N, or a sequence of N numbers to be mapped to colors using the cmap and norm specified via kwargs (see below). Note that c should not be a single numeric RGB or RGBA sequence because that is indistinguishable from an array of values to be colormapped. c can be a 2-D array in which the rows are RGB or RGBA, however, including the case of a single row to specify the same color for all points.
depths	Whether or not to shade the scatter markers to give the appearance of depth. Default is True.

Keyword arguments are passed on to `scatter()`.

Returns a `Patch3DCollection`

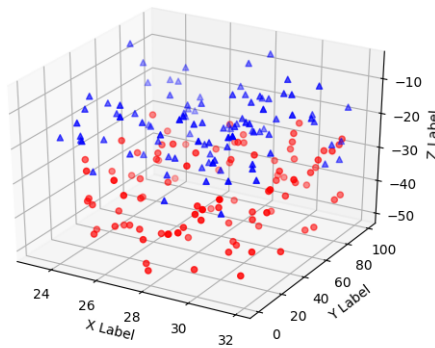


Fig. 49: Scatter3d

Wireframe plots

`Axes3D.plot_wireframe(X, Y, Z, *args, **kwargs)`
 Plot a 3D wireframe.

Note: The `rcount` and `ccount` kwargs, which both default to 50, determine the maximum number of samples used in each direction. If the input data is larger, it will be downsampled

(by slicing) to these numbers of points.

Parameters

X, Y, Z [2d arrays] Data values.

rcount, ccount [int] Maximum number of samples used in each direction. If the input data is larger, it will be downsampled (by slicing) to these numbers of points. Setting a count to zero causes the data to be not sampled in the corresponding direction, producing a 3D line plot rather than a wireframe plot. Defaults to 50.

New in version 2.0.

rstride, cstride [int] Downsampling stride in each direction. These arguments are mutually exclusive with rcount and ccount. If only one of rstride or cstride is set, the other defaults to 1. Setting a stride to zero causes the data to be not sampled in the corresponding direction, producing a 3D line plot rather than a wireframe plot.

'classic' mode uses a default of rstride = cstride = 1 instead of the new default of rcount = ccount = 50.

**kwargs : Other arguments are forwarded to Line3DCollection.

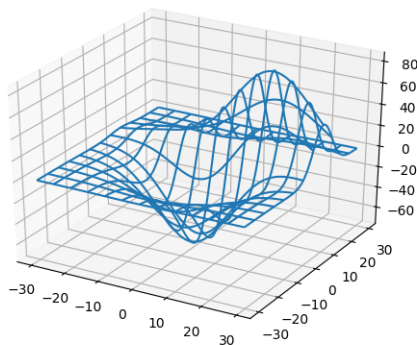


Fig. 50: Wire3d

Surface plots

`Axes3D.plot_surface(X, Y, Z, *args, **kwargs)`

Create a surface plot.

By default it will be colored in shades of a solid color, but it also supports color mapping by supplying the `cmap` argument.

Note: The `rcount` and `ccount` kwargs, which both default to 50, determine the maximum number of samples used in each direction. If the input data is larger, it will be downsampled (by slicing) to these numbers of points.

Parameters

`X, Y, Z` [2d arrays] Data values.

`rcount, ccount` [int] Maximum number of samples used in each direction. If the input data is larger, it will be downsampled (by slicing) to these numbers of points. Defaults to 50.

New in version 2.0.

`rstride, cstride` [int] Downsampling stride in each direction. These arguments are mutually exclusive with `rcount` and `ccount`. If only one of `rstride` or `cstride` is set, the other defaults to 10.

'classic' mode uses a default of `rstride = cstride = 10` instead of the new default of `rcount = ccount = 50`.

`color` [color-like] Color of the surface patches.

`cmap` [Colormap] Colormap of the surface patches.

`facecolors` [array-like of colors.] Colors of each individual patch.

`norm` [Normalize] Normalization for the colormap.

`vmin, vmax` [float] Bounds for the normalization.

`shade` [bool] Whether to shade the face colors.

****kwargs** : Other arguments are forwarded to `Poly3DCollection`.

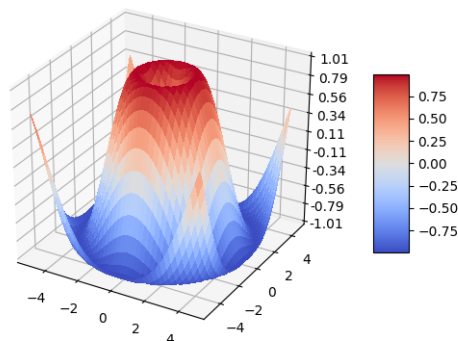


Fig. 51: Surface3d
 Surface3d 2
 Surface3d 3

Tri-Surface plots

`Axes3D.plot_trisurf(*args, **kwargs)`

Argument	Description
X, Y, Z	Data values as 1D arrays
color	Color of the surface patches
cmap	A colormap for the surface patches.
norm	An instance of Normalize to map values to colors
vmin	Minimum value to map
vmax	Maximum value to map
shade	Whether to shade the facecolors

The (optional) triangulation can be specified in one of two ways; either:

```
plot_trisurf(triangulation, ...)
```

where triangulation is a Triangulation object, or:

```
plot_trisurf(X, Y, ...)
plot_trisurf(X, Y, triangles, ...)
plot_trisurf(X, Y, triangles=triangles, ...)
```

in which case a Triangulation object will be created. See Triangulation for a explanation of these possibilities.

The remaining arguments are:

```
plot_trisurf(..., Z)
```

where Z is the array of values to contour, one per point in the triangulation.

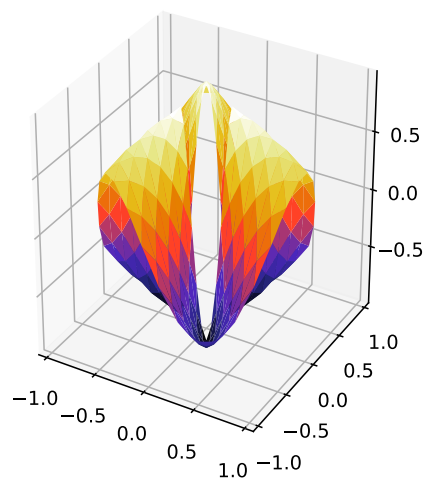
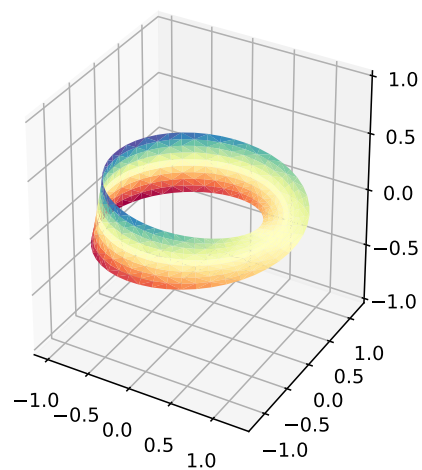
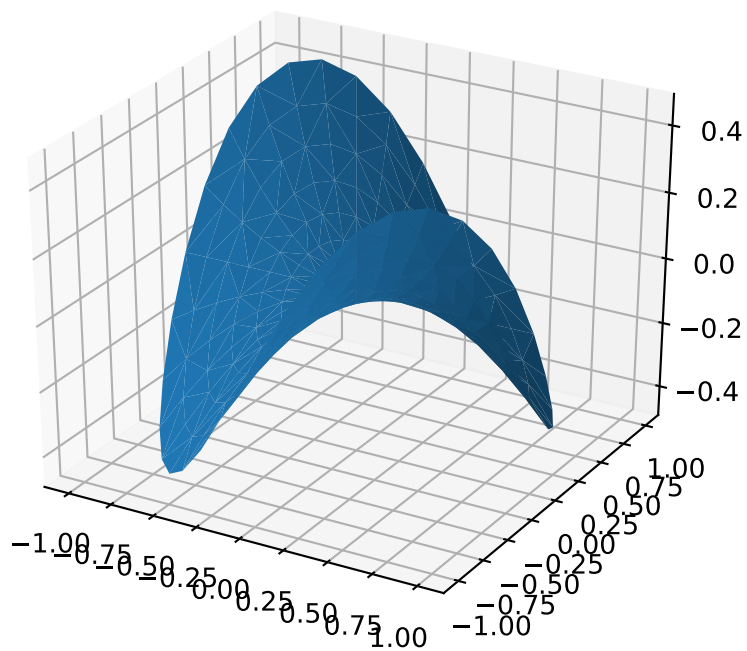
Other arguments are passed on to Poly3DCollection

Examples:

New in version 1.2.0: This plotting function was added for the v1.2.0 release.

Contour plots

`Axes3D.contour(X, Y, Z, *args, **kwargs)`
Create a 3D contour plot.



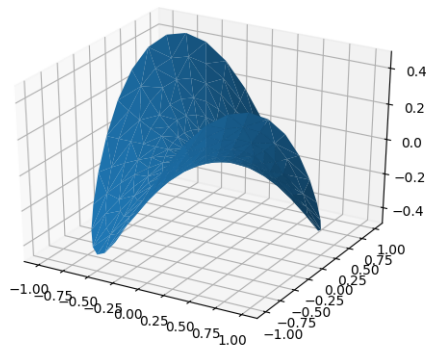


Fig. 52: Trisurf3d

Argument	Description
X, Y, Z	Data values as numpy.arrays
extend3d	Whether to extend contour in 3D (default: False)
stride	Stride (step size) for extending contour
zdir	The direction to use: x, y or z (default)
offset	If specified plot a projection of the contour lines on this position in plane normal to zdir

The positional and other keyword arguments are passed on to `contour()`

Returns a contour

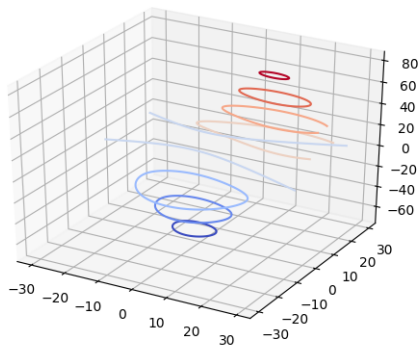


Fig. 53: Contour3d
Contour3d 2
Contour3d 3

Filled contour plots

`Axes3D.contourf(X, Y, Z, *args, **kwargs)`
 Create a 3D contourf plot.

Argument	Description
X, Y, Z	Data values as <code>numpy.array</code> s
zdir	The direction to use: x, y or z (default)
offset	If specified plot a projection of the filled contour on this position in plane normal to zdir

The positional and keyword arguments are passed on to `contourf()`

Returns a `contourf`

Changed in version 1.1.0: The `zdir` and `offset` kwargs were added.

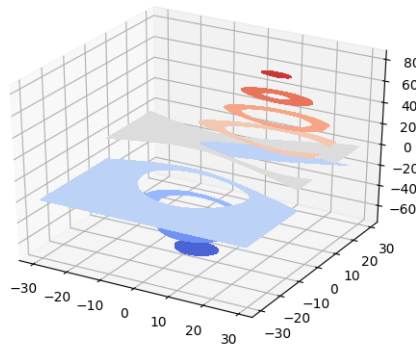


Fig. 54: Contourf3d
 Contourf3d 2

New in version 1.1.0: The feature demoed in the second `contourf3d` example was enabled as a result of a bugfix for version 1.1.0.

Polygon plots

`Axes3D.add_collection3d(col, zs=0, zdir='z')`
 Add a 3D collection object to the plot.

2D collection types are converted to a 3D version by modifying the object and adding z coordinate information.

Supported are:

- PolyCollection
- LineCollection
- PatchCollection

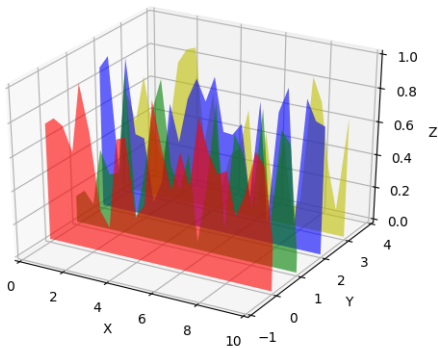


Fig. 55: Polys3d

Bar plots

`Axes3D.bar(left, height, zs=0, zdir='z', *args, **kwargs)`
Add 2D bar(s).

Argument	Description
left	The x coordinates of the left sides of the bars.
height	The height of the bars.
zs	Z coordinate of bars, if one value is specified they will all be placed at the same z.
zdir	Which direction to use as z ('x', 'y' or 'z') when plotting a 2D set.

Keyword arguments are passed onto `bar()`.

Returns a `Patch3DCollection`

Quiver

`Axes3D.quiver(*args, **kwargs)`
Plot a 3D field of arrows.

call signatures:

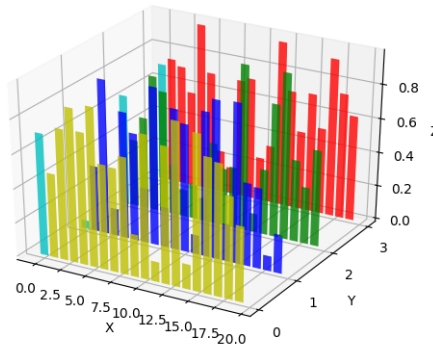


Fig. 56: Bars3d

```
quiver(X, Y, Z, U, V, W, **kwargs)
```

Arguments:

X, Y, Z: The x, y and z coordinates of the arrow locations (default is tail of arrow; see pivot kwarg)

U, V, W: The x, y and z components of the arrow vectors

The arguments could be array-like or scalars, so long as they can be broadcast together. The arguments can also be masked arrays. If an element in any of argument is masked, then that corresponding quiver element will not be plotted.

Keyword arguments:

length: [1.0 | float] The length of each quiver, default to 1.0, the unit is the same with the axes

arrow_length_ratio: [0.3 | float] The ratio of the arrow head with respect to the quiver, default to 0.3

pivot: ['tail' | 'middle' | 'tip'] The part of the arrow that is at the grid point; the arrow rotates about this point, hence the name pivot. Default is 'tail'

normalize: bool When True, all of the arrows will be the same length. This defaults to False, where the arrows will be different lengths depending on the values of u,v,w.

Any additional keyword arguments are delegated to LineCollection

2D plots in 3D

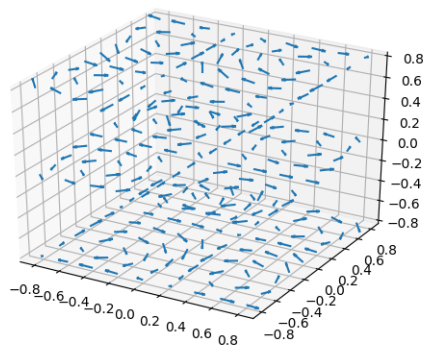


Fig. 57: Quiver3d

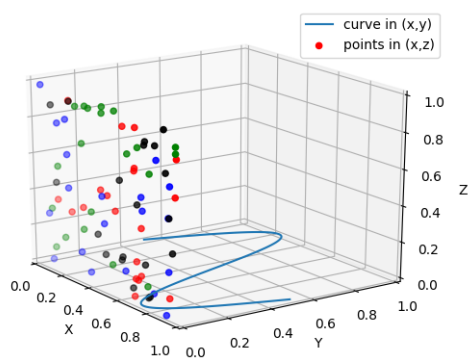


Fig. 58: 2dcollections3d

Text

`Axes3D.text(x, y, z, s, zdir=None, **kwargs)`

Add text to the plot. `kwargs` will be passed on to `Axes.text`, except for the `zdir` keyword, which sets the direction to be used as the z direction.

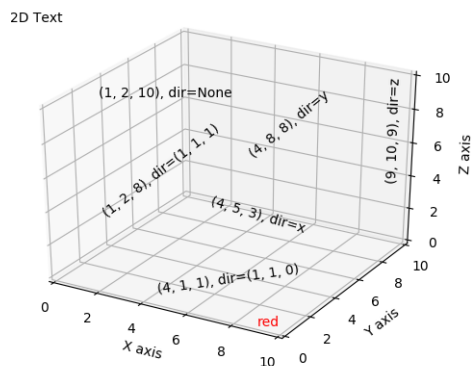


Fig. 59: Text3d

Subplotting

Having multiple 3D plots in a single figure is the same as it is for 2D plots. Also, you can have both 2D and 3D plots in the same figure.

New in version 1.0.0: Subplotting 3D plots was added in v1.0.0. Earlier version can not do this.

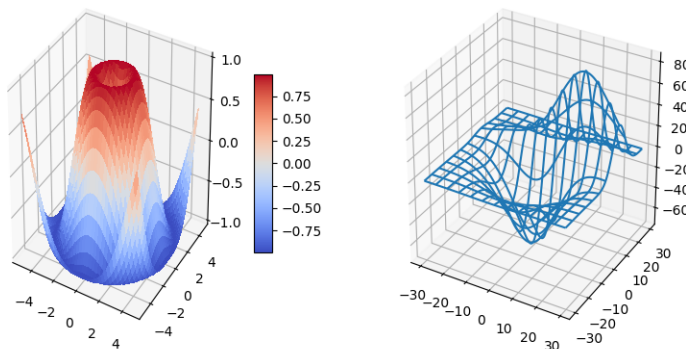


Fig. 60: Subplot3d
Mixed Subplots

Note: [Click here to download the full example code](#)

3.6.2 Overview of axisartist toolkit

The axisartist toolkit tutorial.

Warning: `axisartist` uses a custom `Axes` class (derived from the mpl's original `Axes` class). As a side effect, some commands (mostly tick-related) do not work.

The `axisartist` contains a custom `Axes` class that is meant to support curvilinear grids (e.g., the world coordinate system in astronomy). Unlike mpl's original `Axes` class which uses `Axes.xaxis` and `Axes.yaxis` to draw ticks, ticklines, etc., `axisartist` uses a special artist (`AxisArtist`) that can handle ticks, ticklines, etc. for curved coordinate systems.

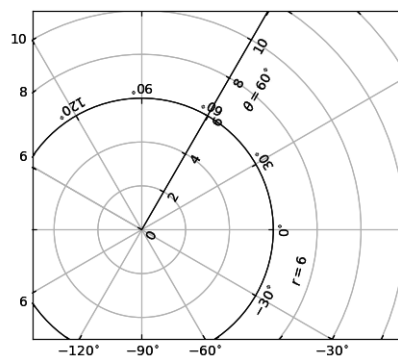


Fig. 61: Demo Floating Axis

Since it uses special artists, some Matplotlib commands that work on `Axes.xaxis` and `Axes.yaxis` may not work.

`axisartist`

The `axisartist` module provides a custom (and very experimental) `Axes` class, where each axis (left, right, top, and bottom) have a separate associated artist which is responsible for drawing the axis-line, ticks, ticklabels, and labels. You can also create your own axis, which can pass through a fixed position in the axes coordinate, or a fixed position in the data coordinate (i.e., the axis floats around when `viewlimit` changes).

The axes class, by default, has its `xaxis` and `yaxis` invisible, and has 4 additional artists which are responsible for drawing the 4 axis spines in "left", "right", "bottom", and "top". They are accessed

as `ax.axis["left"]`, `ax.axis["right"]`, and so on, i.e., `ax.axis` is a dictionary that contains artists (note that `ax.axis` is still a callable method and it behaves as an original `Axes.axis` method in Matplotlib).

To create an axes,

```
import mpl_toolkits.axisartist as AA
fig = plt.figure(1)
ax = AA.Axes(fig, [0.1, 0.1, 0.8, 0.8])
fig.add_axes(ax)
```

or to create a subplot

```
ax = AA.Subplot(fig, 111)
fig.add_subplot(ax)
```

For example, you can hide the right and top spines using:

```
ax.axis["right"].set_visible(False)
ax.axis["top"].set_visible(False)
```

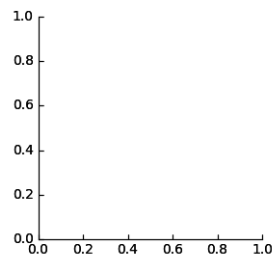


Fig. 62: Simple Axisline3

It is also possible to add a horizontal axis. For example, you may have an horizontal axis at $y=0$ (in data coordinate).

```
ax.axis["y=0"] = ax.new_floating_axis(nth_coord=0, value=0)
```

Or a fixed axis with some offset

```
# make new (right-side) yaxis, but with some offset
ax.axis["right2"] = ax.new_fixed_axis(loc="right",
    offset=(20, 0))
```

axisartist with ParasiteAxes

Most commands in the `axes_grid1` toolkit can take an `axes_class` keyword argument, and the commands create an axes of the given class. For example, to create a host subplot with `axisartist.Axes`,

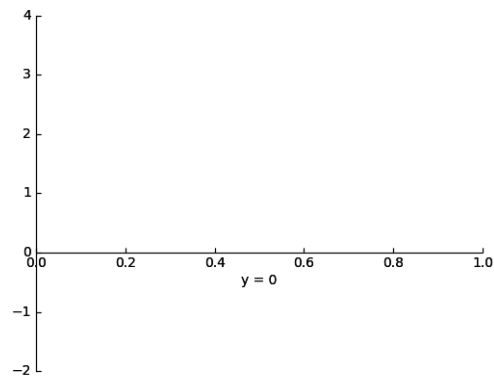


Fig. 63: Simple Axisartist1

```
import mpl_toolkits.axisartist as AA
from mpl_toolkits.axes_grid1 import host_subplot

host = host_subplot(111, axes_class=AA.Axes)
```

Here is an example that uses ParasiteAxes.

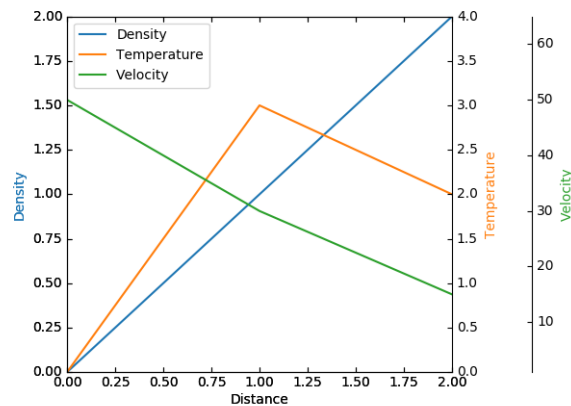


Fig. 64: Demo Parasite Axes2

Curvilinear Grid

The motivation behind the AxisArtist module is to support a curvilinear grid and ticks.

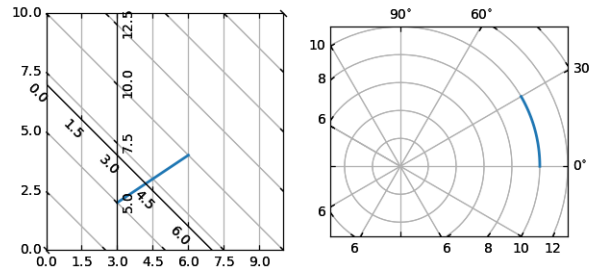


Fig. 65: Demo Curvilinear Grid

Floating Axes

AxisArtist also supports a Floating Axes whose outer axes are defined as floating axis.

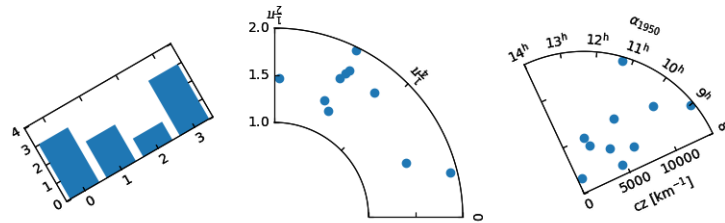


Fig. 66: Demo Floating Axes

axisartist namespace

The axisartist namespace includes a derived Axes implementation. The biggest difference is that the artists responsible to draw axis line, ticks, ticklabel and axis labels are separated out from the mpl's Axis class, which are much more than artists in the original mpl. This change was strongly motivated to support curvilinear grid. Here are a few things that `mpl_toolkits.axisartist.Axes` is different from original Axes from mpl.

- Axis elements (axis line(spine), ticks, ticklabel and axis labels) are drawn by a AxisArtist instance. Unlike Axis, left, right, top and bottom axis are drawn by separate artists. And each of them may have different tick location and different tick labels.
- gridlines are drawn by a Gridlines instance. The change was motivated that in curvilinear coordinate, a gridline may not cross axis-lines (i.e., no associated ticks). In the original Axes class, gridlines are tied to ticks.
- ticklines can be rotated if necessary (i.e, along the gridlines)

In summary, all these changes was to support

- a curvilinear grid.
- a floating axis

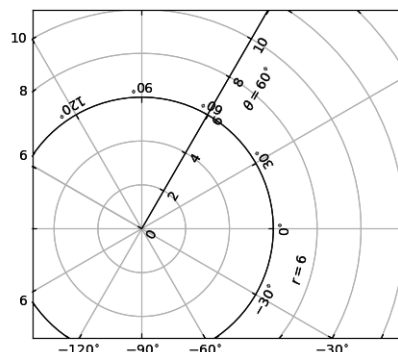


Fig. 67: Demo Floating Axis

`mpl_toolkits.axisartist.Axes` class defines a `axis` attribute, which is a dictionary of `AxisArtist` instances. By default, the dictionary has 4 `AxisArtist` instances, responsible for drawing of left, right, bottom and top axis.

`axis` and `yaxis` attributes are still available, however they are set to not visible. As separate artists are used for rendering axis, some axis-related method in `mpl` may have no effect. In addition to `AxisArtist` instances, the `mpl_toolkits.axisartist.Axes` will have `gridlines` attribute (`Gridlines`), which obviously draws grid lines.

In both `AxisArtist` and `Gridlines`, the calculation of tick and grid location is delegated to an instance of `GridHelper` class. `mpl_toolkits.axisartist.Axes` class uses `GridHelperRectlinear` as a grid helper. The `GridHelperRectlinear` class is a wrapper around the `xaxis` and `yaxis` of `mpl`'s original `Axes`, and it was meant to work as the way how `mpl`'s original axes works. For example, tick location changes using `set_ticks` method and etc. should work as expected. But change in artist properties (e.g., color) will not work in general, although some effort has been made so that some often-change attributes (color, etc.) are respected.

AxisArtist

`AxisArtist` can be considered as a container artist with following attributes which will draw ticks, labels, etc.

- `line`
- `major_ticks`, `major_ticklabels`
- `minor_ticks`, `minor_ticklabels`
- `offsetText`

- label

line

Derived from Line2d class. Responsible for drawing a spinal(?) line.

major_ticks, minor_ticks

Derived from Line2d class. Note that ticks are markers.

major_ticklabels, minor_ticklabels

Derived from Text. Note that it is not a list of Text artist, but a single artist (similar to a collection).

axislabel

Derived from Text.

Default AxisArtists

By default, following for axis artists are defined.:

```
ax.axis["left"], ax.axis["bottom"], ax.axis["right"], ax.axis["top"]
```

The ticklabels and axislabel of the top and the right axis are set to not visible.

For example, if you want to change the color attributes of major_ticklabels of the bottom x-axis

```
ax.axis["bottom"].major_ticklabels.set_color("b")
```

Similarly, to make ticklabels invisible

```
ax.axis["bottom"].major_ticklabels.set_visible(False)
```

AxisArtist provides a helper method to control the visibility of ticks, ticklabels, and label. To make ticklabel invisible,

```
ax.axis["bottom"].toggle(ticklabels=False)
```

To make all of ticks, ticklabels, and (axis) label invisible

```
ax.axis["bottom"].toggle(all=False)
```

To turn all off but ticks on

```
ax.axis["bottom"].toggle(all=False, ticks=True)
```

To turn all on but (axis) label off

```
ax.axis["bottom"].toggle(all=True, label=False))
```

ax.axis's `__getitem__` method can take multiple axis names. For example, to turn ticklabels of "top" and "right" axis on,

```
ax.axis["top", "right"].toggle(ticklabels=True))
```

Note that 'ax.axis["top", "right"]' returns a simple proxy object that translate above code to something like below.

```
for n in ["top", "right"]:
    ax.axis[n].toggle(ticklabels=True))
```

So, any return values in the for loop are ignored. And you should not use it anything more than a simple method.

Like the list indexing ":" means all items, i.e.,

```
ax.axis[:].major_ticks.set_color("r")
```

changes tick color in all axis.

HowTo

1. Changing tick locations and label.

Same as the original mpl's axes.:

```
ax.set_xticks([1,2,3])
```

2. Changing axis properties like color, etc.

Change the properties of appropriate artists. For example, to change the color of the ticklabels:

```
ax.axis["left"].major_ticklabels.set_color("r")
```

3. To change the attributes of multiple axis:

```
ax.axis["left", "bottom"].major_ticklabels.set_color("r")
```

or to change the attributes of all axis:

```
ax.axis[:].major_ticklabels.set_color("r")
```

4. To change the tick size (length), you need to use `axis.major_ticks.set_ticksize` method. To change the direction of the ticks (ticks are in opposite direction of ticklabels by default), use `axis.major_ticks.set_tick_out` method.

To change the pad between ticks and ticklabels, use `axis.major_ticklabels.set_pad` method.

To change the pad between ticklabels and axis label, `axis.label.set_pad` method.

Rotation and Alignment of TickLabels

This is also quite different from the original mpl and can be confusing. When you want to rotate the ticklabels, first consider using `"set_axis_direction"` method.

```
ax1.axis["left"].major_ticklabels.set_axis_direction("top")
ax1.axis["right"].label.set_axis_direction("left")
```

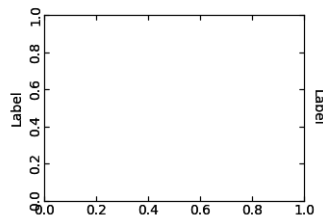


Fig. 68: Simple Axis Direction01

The parameter for `set_axis_direction` is one of `["left", "right", "bottom", "top"]`.

You must understand some underlying concept of directions.

1. There is a reference direction which is defined as the direction of the axis line with increasing coordinate. For example, the reference direction of the left x-axis is from bottom to top.



Fig. 69: Axis Direction Demo - Step 01

The direction, text angle, and alignments of the ticks, ticklabels and axis-label is determined with respect to the reference direction

2. `ticklabel_direction` is either the right-hand side (+) of the reference direction or the left-hand side (-).

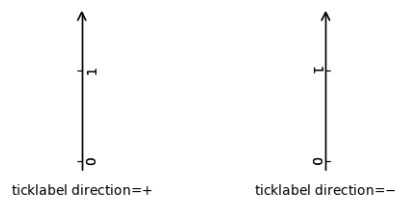


Fig. 70: Axis Direction Demo - Step 02



Fig. 71: Axis Direction Demo - Step 03

- 3. same for the label_direction
- 4. ticks are by default drawn toward the opposite direction of the ticklabels.
- 5. text rotation of ticklabels and label is determined in reference to the ticklabel_direction or label_direction, respectively. The rotation of ticklabels and label is anchored.

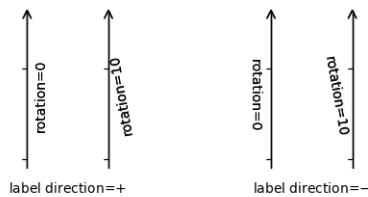


Fig. 72: Axis Direction Demo - Step 04

On the other hand, there is a concept of "axis_direction". This is a default setting of above properties for each, "bottom", "left", "top", and "right" axis.

?	?	left	bottom	right	top
axislabel	direction	'_'	'+'	'+'	'_'
axislabel	rotation	180	0	0	180
axislabel	va	center	top	center	bottom
axislabel	ha	right	center	right	center
ticklabel	direction	'_'	'+'	'+'	'_'
ticklabels	rotation	90	0	-90	180
ticklabel	ha	right	center	right	center
ticklabel	va	center	baseline	center	baseline

And, 'set_axis_direction("top")' means to adjust the text rotation etc, for settings suitable for "top" axis. The concept of axis direction can be more clear with curved axis.

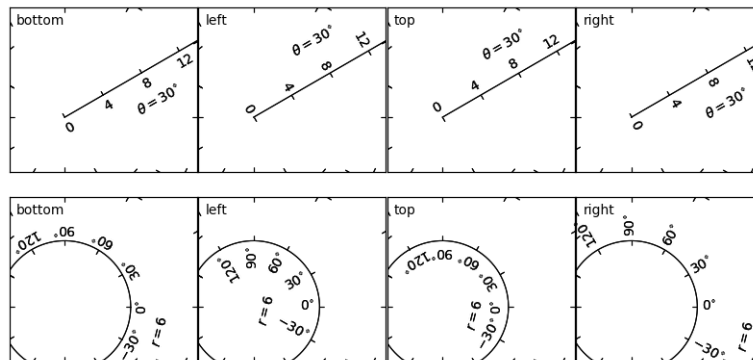


Fig. 73: Demo Axis Direction

The axis_direction can be adjusted in the AxisArtist level, or in the level of its child artists, i.e., ticks, ticklabels, and axis-label.

```
ax1.axis["left"].set_axis_direction("top")
```

changes axis_direction of all the associated artist with the "left" axis, while

```
ax1.axis["left"].major_ticklabels.set_axis_direction("top")
```

changes the axis_direction of only the major_ticklabels. Note that set_axis_direction in the AxisArtist level changes the ticklabel_direction and label_direction, while changing the axis_direction of ticks, ticklabels, and axis-label does not affect them.

If you want to make ticks outward and ticklabels inside the axes, use invert_ticklabel_direction method.

```
ax.axis[:].invert_ticklabel_direction()
```

A related method is "set_tick_out". It makes ticks outward (as a matter of fact, it makes ticks toward the opposite direction of the default direction).

```
ax.axis[:].major_ticks.set_tick_out(True)
```

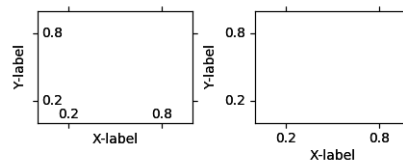


Fig. 74: Simple Axis Direction03

So, in summary,

- AxisArtist's methods
 - `set_axis_direction` : "left", "right", "bottom", or "top"
 - `set_ticklabel_direction` : "+" or "-"
 - `set_axislabel_direction` : "+" or "-"
 - `invert_ticklabel_direction`
- Ticks' methods (`major_ticks` and `minor_ticks`)
 - `set_tick_out` : True or False
 - `set_ticksize` : size in points
- TickLabels' methods (`major_ticklabels` and `minor_ticklabels`)
 - `set_axis_direction` : "left", "right", "bottom", or "top"
 - `set_rotation` : angle with respect to the reference direction
 - `set_ha` and `set_va` : see below
- AxisLabels' methods (`label`)
 - `set_axis_direction` : "left", "right", "bottom", or "top"
 - `set_rotation` : angle with respect to the reference direction
 - `set_ha` and `set_va`

Adjusting ticklabels alignment

Alignment of TickLabels are treated specially. See below

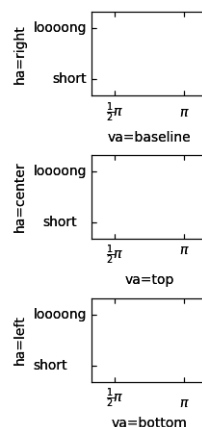


Fig. 75: Demo Ticklabel Alignment

Adjusting pad

To change the pad between ticks and ticklabels

```
ax.axis["left"].major_ticklabels.set_pad(10)
```

Or ticklabels and axis-label

```
ax.axis["left"].label.set_pad(10)
```

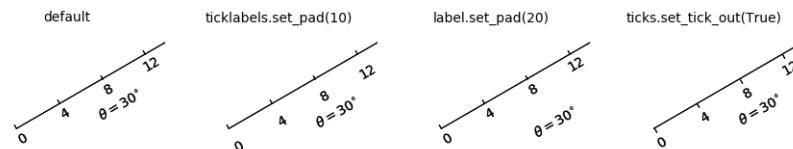


Fig. 76: Simple Axis Pad

GridHelper

To actually define a curvilinear coordinate, you have to use your own grid helper. A generalised version of grid helper class is supplied and this class should suffice in most of cases. A user may provide two functions which defines a transformation (and its inverse pair) from the curved coordinate to (rectilinear) image coordinate. Note that while ticks and grids are drawn for curved coordinate, the data transform of the axes itself (`ax.transData`) is still rectilinear (image) coordinate.

```
from mpl_toolkits.axisartist.grid_helper_curvilinear \
    import GridHelperCurveLinear
from mpl_toolkits.axisartist import Subplot

# from curved coordinate to rectilinear coordinate.
def tr(x, y):
    x, y = np.asarray(x), np.asarray(y)
    return x, y-x

# from rectilinear coordinate to curved coordinate.
def inv_tr(x,y):
    x, y = np.asarray(x), np.asarray(y)
    return x, y+x

grid_helper = GridHelperCurveLinear((tr, inv_tr))

ax1 = Subplot(fig, 1, 1, 1, grid_helper=grid_helper)
```

(continues on next page)

(continued from previous page)

```
fig.add_subplot(ax1)
```

You may use matplotlib's Transform instance instead (but a inverse transformation must be defined). Often, coordinate range in a curved coordinate system may have a limited range, or may have cycles. In those cases, a more customized version of grid helper is required.

```
import mpl_toolkits.axisartist.angle_helper as angle_helper

# PolarAxes.PolarTransform takes radian. However, we want our coordinate
# system in degree
tr = Affine2D().scale(np.pi/180., 1.) + PolarAxes.PolarTransform()

# extreme finder : find a range of coordinate.
# 20, 20 : number of sampling points along x, y direction
# The first coordinate (longitude, but theta in polar)
# has a cycle of 360 degree.
# The second coordinate (latitude, but radius in polar) has a minimum of 0
extreme_finder = angle_helper.ExtremeFinderCycle(20, 20,
                                                  lon_cycle = 360,
                                                  lat_cycle = None,
                                                  lon_minmax = None,
                                                  lat_minmax = (0, np.inf),
                                                  )

# Find a grid values appropriate for the coordinate (degree,
# minute, second). The argument is a approximate number of grids.
grid_locator1 = angle_helper.LocatorDMS(12)

# And also uses an appropriate formatter. Note that, the
# acceptable Locator and Formatter class is a bit different than
# that of mpl's, and you cannot directly use mpl's Locator and
# Formatter here (but may be possible in the future).
tick_formatter1 = angle_helper.FormatterDMS()

grid_helper = GridHelperCurveLinear(tr,
                                   extreme_finder=extreme_finder,
                                   grid_locator1=grid_locator1,
                                   tick_formatter1=tick_formatter1
                                   )
```

Again, the transData of the axes is still a rectilinear coordinate (image coordinate). You may manually do conversion between two coordinates, or you may use Parasite Axes for convenience.:

```
ax1 = SubplotHost(fig, 1, 2, 2, grid_helper=grid_helper)
```

(continues on next page)

(continued from previous page)

```
# A parasite axes with given transform
ax2 = ParasiteAxesAuxTrans(ax1, tr, "equal")
# note that ax2.transData == tr + ax1.transData
# Anything you draw in ax2 will match the ticks and grids of ax1.
ax1.parasites.append(ax2)
```

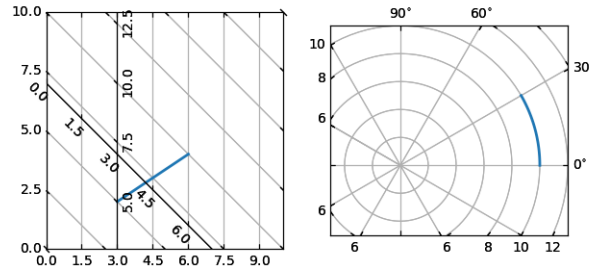


Fig. 77: Demo Curvilinear Grid

FloatingAxis

A floating axis is an axis one of whose data coordinate is fixed, i.e, its location is not fixed in Axes coordinate but changes as axes data limits changes. A floating axis can be created using `new_floating_axis` method. However, it is your responsibility that the resulting `AxisArtist` is properly added to the axes. A recommended way is to add it as an item of Axes's `axis` attribute.:

```
# floating axis whose first (index starts from 0) coordinate
# (theta) is fixed at 60

ax1.axis["lat"] = axis = ax1.new_floating_axis(0, 60)
axis.label.set_text(r"$\theta = 60^\circ$")
axis.label.set_visible(True)
```

See the first example of this page.

Current Limitations and TODO's

The code need more refinement. Here is a incomplete list of issues and TODO's

- No easy way to support a user customized tick location (for curvilinear grid). A new `Locator` class needs to be created.
- `FloatingAxis` may have coordinate limits, e.g., a floating axis of $x = 0$, but y only spans from 0 to 1.

- The location of axislabel of `FloatingAxis` needs to be optionally given as a coordinate value. ex, a floating axis of $x=0$ with label at $y=1$

Note: Click here to download the full example code

3.6.3 Overview of `axes_grid1` toolkit

Controlling the layout of plots with the `axes_grid` toolkit.

What is `axes_grid1` toolkit?

`axes_grid1` is a collection of helper classes to ease displaying (multiple) images with matplotlib. In matplotlib, the axes location (and size) is specified in the normalized figure coordinates, which may not be ideal for displaying images that needs to have a given aspect ratio. For example, it helps if you have a colorbar whose height always matches that of the image. [ImageGrid](#), [RGB Axes](#) and [AxesDivider](#) are helper classes that deals with adjusting the location of (multiple) Axes. They provides a framework to adjust the position of multiple axes at the drawing time. [ParasiteAxes](#) provides `twinx`(or `twiny`)-like features so that you can plot different data (e.g., different y-scale) in a same Axes. [AnchoredArtists](#) includes custom artists which are placed at some anchored position, like the legend.

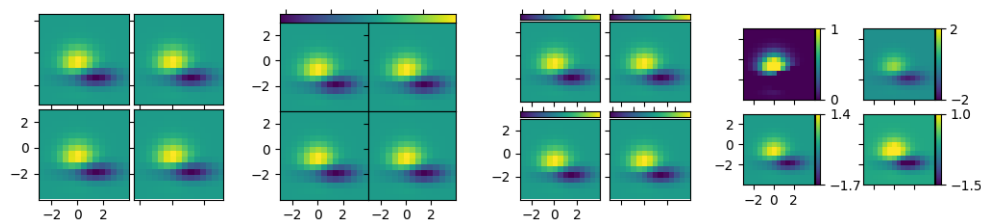


Fig. 78: Demo Axes Grid

`axes_grid1`

[ImageGrid](#)

A class that creates a grid of Axes. In matplotlib, the axes location (and size) is specified in the normalized figure coordinates. This may not be ideal for images that needs to be displayed with a given aspect ratio. For example, displaying images of a same size with some fixed padding between them cannot be easily done in matplotlib. `ImageGrid` is used in such case.

- The position of each axes is determined at the drawing time (see [AxesDivider](#)), so that the size of the entire grid fits in the given rectangle (like the aspect of axes). Note that in this example, the paddings between axes are fixed even if you changes the figure size.

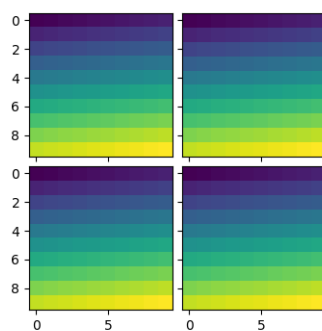


Fig. 79: Simple Axesgrid

- axes in the same column has a same axes width (in figure coordinate), and similarly, axes in the same row has a same height. The widths (height) of the axes in the same row (column) are scaled according to their view limits (xlim or ylim).

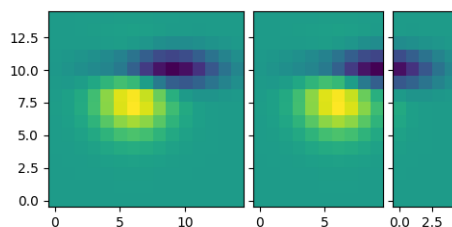


Fig. 80: Simple Axes Grid

- xaxis are shared among axes in a same column. Similarly, yaxis are shared among axes in a same row. Therefore, changing axis properties (view limits, tick location, etc. either by plot commands or using your mouse in interactive backends) of one axes will affect all other shared axes.

When initialized, ImageGrid creates given number (ngrid or ncols * nrows if ngrid is None) of Axes instances. A sequence-like interface is provided to access the individual Axes instances (e.g., grid[0] is the first Axes in the grid. See below for the order of axes).

ImageGrid takes following arguments,

Name	De- fault	Description
fig		
rect		
nrows_ncols		number of rows and cols. e.g., (2,2)
ngrids	None	number of grids. nrows x ncols if None
direction	"row"	increasing direction of axes number. [row column]
axes_pad	0.02	pad between axes in inches
add_all	True	Add axes to figures if True
share_all	False	xaxis & yaxis of all axes are shared if True
aspect	True	aspect of axes
label_mode	"L"	location of tick labels that will be displayed. "1" (only the lower left axes), "L" (left most and bottom most axes), or "all".
cbar_mode	None	[None single each]
cbar_location	"right"	[right top]
cbar_pad	None	pad between image axes and colorbar axes
cbar_size	"5%"	size of the colorbar
axes_class	None	

`rect` specifies the location of the grid. You can either specify coordinates of the rectangle to be used (e.g., (0.1, 0.1, 0.8, 0.8) as in the Axes), or the subplot-like position (e.g., "121").

`direction` means the increasing direction of the axes number.

`aspect` By default (False), widths and heights of axes in the grid are scaled independently. If True, they are scaled according to their data limits (similar to aspect parameter in mpl).

`share_all` if True, xaxis and yaxis of all axes are shared.

`direction` direction of increasing axes number. For "row",

grid[0]	grid[1]
grid[2]	grid[3]

For "column",

grid[0]	grid[2]
grid[1]	grid[3]

You can also create a colorbar (or colorbars). You can have colorbar for each axes (`cbar_mode="each"`), or you can have a single colorbar for the grid (`cbar_mode="single"`). The colorbar can be placed on your right, or top. The axes for each colorbar is stored as a `cbar_axes` attribute.

The examples below show what you can do with ImageGrid.

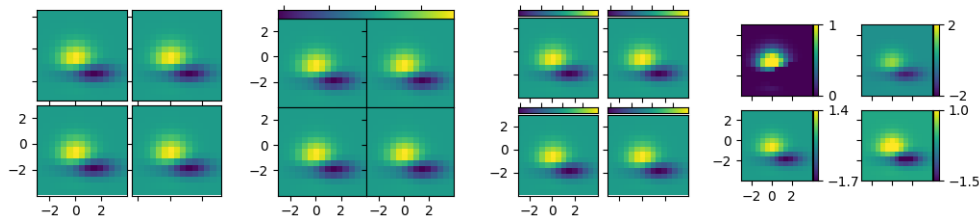


Fig. 81: Demo Axes Grid

AxesDivider Class

Behind the scene, the `ImageGrid` class and the `RGBAxes` class utilize the `AxesDivider` class, whose role is to calculate the location of the axes at drawing time. While a more about the `AxesDivider` is (will be) explained in (yet to be written) `AxesDividerGuide`, direct use of the `AxesDivider` class will not be necessary for most users. The `axes_divider` module provides a helper function `make_axes_locatable`, which can be useful. It takes a existing axes instance and create a divider for it.

```
ax = subplot(1,1,1)
divider = make_axes_locatable(ax)
```

`make_axes_locatable` returns an instance of the `AxesLocator` class, derived from the `Locator`. It provides `append_axes` method that creates a new axes on the given side of ("top", "right", "bottom" and "left") of the original axes.

`colorbar` whose height (or width) in sync with the master axes

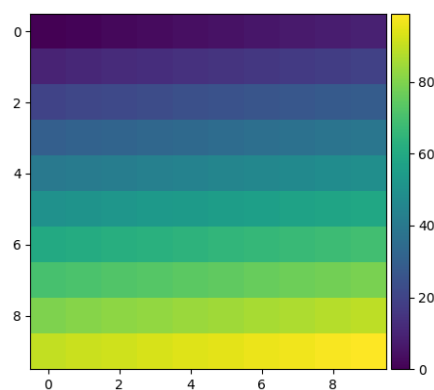


Fig. 82: Simple Colorbar

scatter_hist.py with AxesDivider

The "scatter_hist.py" example in mpl can be rewritten using make_axes_locatable.

```
axScatter = subplot(111)
axScatter.scatter(x, y)
axScatter.set_aspect(1.)

# create new axes on the right and on the top of the current axes.
divider = make_axes_locatable(axScatter)
axHistx = divider.append_axes("top", size=1.2, pad=0.1, sharex=axScatter)
axHisty = divider.append_axes("right", size=1.2, pad=0.1, sharey=axScatter)

# the scatter plot:
# histograms
bins = np.arange(-lim, lim + binwidth, binwidth)
axHistx.hist(x, bins=bins)
axHisty.hist(y, bins=bins, orientation='horizontal')
```

See the full source code below.

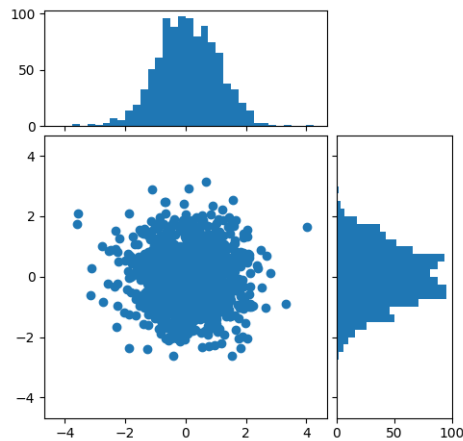


Fig. 83: Scatter Hist

The scatter_hist using the AxesDivider has some advantage over the original scatter_hist.py in mpl. For example, you can set the aspect ratio of the scatter plot, even with the x-axis or y-axis is shared accordingly.

ParasiteAxes

The ParasiteAxes is an axes whose location is identical to its host axes. The location is adjusted in the drawing time, thus it works even if the host change its location (e.g., images).

In most cases, you first create a host axes, which provides a few methods that can be used to create parasite axes. They are `twinx`, `twiny` (which are similar to `twinx` and `twiny` in the `matplotlib`) and `twin`. `twin` takes an arbitrary transformation that maps between the data coordinates of the host axes and the parasite axes. `draw` method of the parasite axes are never called. Instead, host axes collects artists in parasite axes and draw them as if they belong to the host axes, i.e., artists in parasite axes are merged to those of the host axes and then drawn according to their `zorder`. The host and parasite axes modifies some of the axes behavior. For example, color cycle for plot lines are shared between host and parasites. Also, the `legend` command in host, creates a legend that includes lines in the parasite axes. To create a host axes, you may use `host_subplot` or `host_axes` command.

Example 1. `twinx`

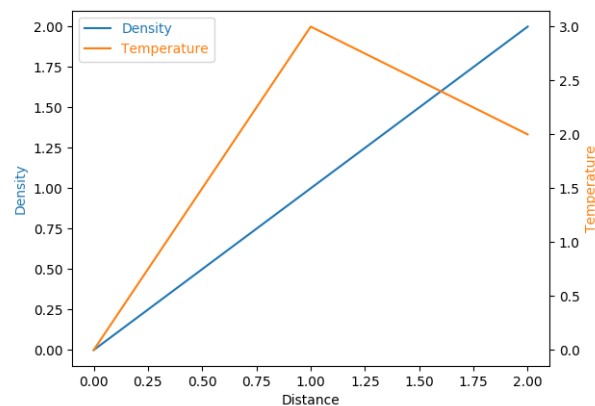


Fig. 84: Parasite Simple

Example 2. `twin`

`twin` without a transform argument assumes that the parasite axes has the same data transform as the host. This can be useful when you want the top(or right)-axis to have different tick-locations, tick-labels, or tick-formatter for bottom(or left)-axis.

```
ax2 = ax.twin() # now, ax2 is responsible for "top" axis and "right" axis
ax2.set_xticks([0., .5*np.pi, np.pi, 1.5*np.pi, 2*np.pi])
ax2.set_xticklabels(["0", r"$\frac{1}{2}\pi$",
                    r"$\pi$", r"$\frac{3}{2}\pi$", r"$2\pi$"])
```

A more sophisticated example using `twin`. Note that if you change the x-limit in the host axes, the x-limit of the parasite axes will change accordingly.

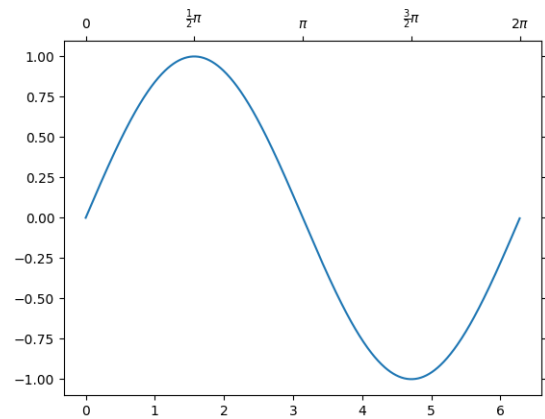


Fig. 85: Simple Axisline4

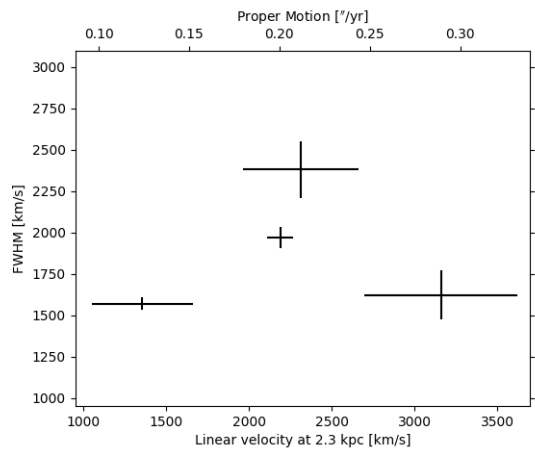


Fig. 86: Parasite Simple2

AnchoredArtists

It's a collection of artists whose location is anchored to the (axes) bbox, like the legend. It is derived from `OffsetBox` in `mpl`, and artist need to be drawn in the canvas coordinate. But, there is a limited support for an arbitrary transform. For example, the ellipse in the example below will have width and height in the data coordinate.

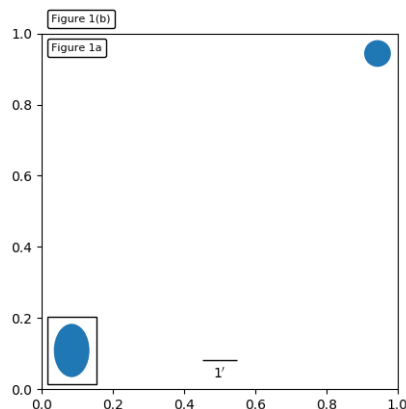


Fig. 87: Simple Anchored Artists

InsetLocator

`mpl_toolkits.axes_grid1.inset_locator` provides helper classes and functions to place your (inset) axes at the anchored position of the parent axes, similarly to `AnchoredArtist`.

Using `mpl_toolkits.axes_grid1.inset_locator.inset_axes()`, you can have inset axes whose size is either fixed, or a fixed proportion of the parent axes. For example,:

```
inset_axes = inset_axes(parent_axes,
                        width="30%", # width = 30% of parent_bbox
                        height=1., # height : 1 inch
                        loc=3)
```

creates an inset axes whose width is 30% of the parent axes and whose height is fixed at 1 inch.

You may create your inset whose size is determined so that the data scale of the inset axes to be that of the parent axes multiplied by some factor. For example,

```
inset_axes = zoomed_inset_axes(ax,
                               0.5, # zoom = 0.5
                               loc=1)
```

creates an inset axes whose data scale is half of the parent axes. Here is complete examples.

For example, `zoomed_inset_axes()` can be used when you want the inset represents the zoom-up of the small portion in the parent axes. And `mpl_toolkits/axes_grid/inset_locator` provides a helper

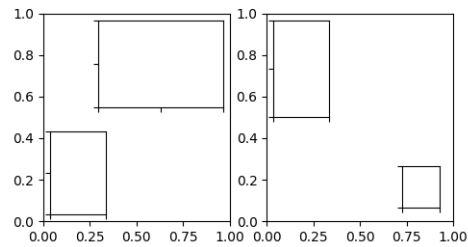


Fig. 88: Inset Locator Demo

function `mark_inset()` to mark the location of the area represented by the inset axes.

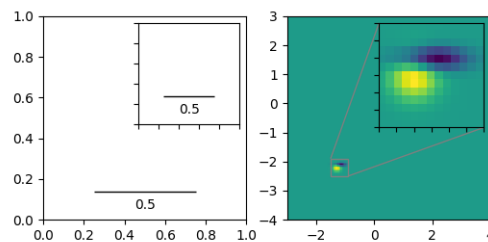


Fig. 89: Inset Locator Demo2

RGB Axes

`RGBAxes` is a helper class to conveniently show RGB composite images. Like `ImageGrid`, the location of axes are adjusted so that the area occupied by them fits in a given rectangle. Also, the xaxis and yaxis of each axes are shared.

```
from mpl_toolkits.axes_grid1.axes_rgb import RGBAxes

fig = plt.figure(1)
ax = RGBAxes(fig, [0.1, 0.1, 0.8, 0.8])

r, g, b = get_rgb() # r,g,b are 2-d images
ax.imshow_rgb(r, g, b,
              origin="lower", interpolation="nearest")
```

AxesDivider

The `axes_divider` module provides helper classes to adjust the axes positions of a set of images at drawing time.

- `axes_size` provides a class of units that are used to determine the size of each axes. For example, you can specify a fixed size.

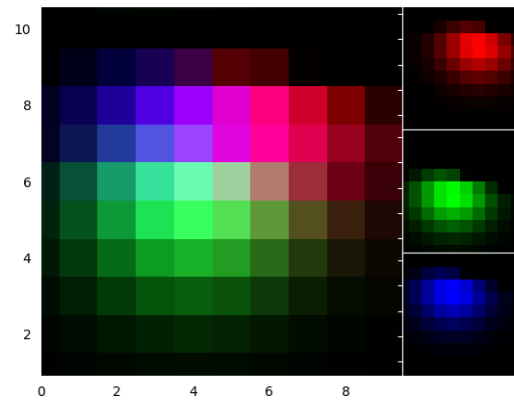


Fig. 90: Simple Rgb

- Divider is the class that calculates the axes position. It divides the given rectangular area into several areas. The divider is initialized by setting the lists of horizontal and vertical sizes on which the division will be based. Then use `new_locator()`, which returns a callable object that can be used to set the `axes_locator` of the axes.

First, initialize the divider by specifying its grids, i.e., horizontal and vertical.

for example,:

```
rect = [0.2, 0.2, 0.6, 0.6]
horiz=[h0, h1, h2, h3]
vert=[v0, v1, v2]
divider = Divider(fig, rect, horiz, vert)
```

where, `rect` is a bounds of the box that will be divided and `h0,..h3`, `v0,..v2` need to be an instance of classes in the `axes_size`. They have `get_size` method that returns a tuple of two floats. The first float is the relative size, and the second float is the absolute size. Consider a following grid.

v0			
v1			
h0,v2	h1	h2	h3

- `v0 => 0, 2`
- `v1 => 2, 0`
- `v2 => 3, 0`

The height of the bottom row is always 2 (`axes_divider` internally assumes that the unit is inches). The first and the second rows have a height ratio of 2:3. For example, if the total height of the grid is 6, then the first and second row will each occupy $2/(2+3)$ and $3/(2+3)$ of (6-1) inches. The widths of the horizontal columns will be similarly determined. When the aspect ratio is set, the total height (or width) will be adjusted accordingly.

The `mpl_toolkits.axes_grid1.axes_size` contains several classes that can be used to set the horizontal and vertical configurations. For example, for vertical configuration one could use:

```
from mpl_toolkits.axes_grid1.axes_size import Fixed, Scaled
vert = [Fixed(2), Scaled(2), Scaled(3)]
```

After you set up the divider object, then you create a locator instance that will be given to the axes object.:

```
locator = divider.new_locator(nx=0, ny=1)
ax.set_axes_locator(locator)
```

The return value of the `new_locator` method is an instance of the `AxesLocator` class. It is a callable object that returns the location and size of the cell at the first column and the second row. You may create a locator that spans over multiple cells.:

```
locator = divider.new_locator(nx=0, nx=2, ny=1)
```

The above locator, when called, will return the position and size of the cells spanning the first and second column and the first row. In this example, it will return `[0:2, 1]`.

See the example,

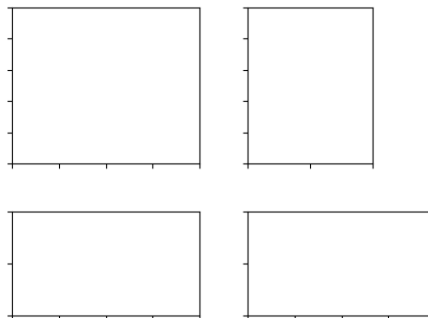


Fig. 91: Simple Axes Divider2

You can adjust the size of each axes according to its x or y data limits (`AxesX` and `AxesY`).

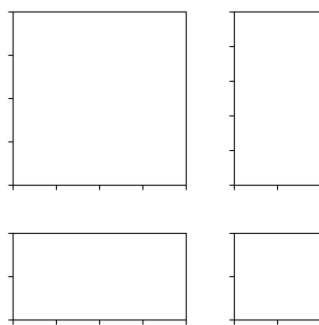


Fig. 92: Simple Axes Divider3

INTERACTIVE PLOTS

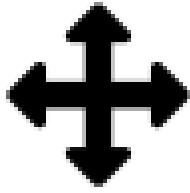
4.1 Interactive navigation



All figure windows come with a navigation toolbar, which can be used to navigate through the data set. Here is a description of each of the buttons at the bottom of the toolbar



The Home, Forward and Back buttons These are akin to a web browser's home, forward and back controls. Forward and Back are used to navigate back and forth between previously defined views. They have no meaning unless you have already navigated somewhere else using the pan and zoom buttons. This is analogous to trying to click Back on your web browser before visiting a new page or Forward before you have gone back to a page – nothing happens. Home always takes you to the first, default view of your data. Again, all of these buttons should feel very familiar to any user of a web browser.

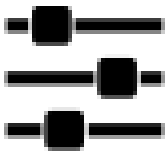


The Pan/Zoom button This button has two modes: pan and zoom. Click the toolbar button to activate panning and zooming, then put your mouse somewhere over an axes. Press the left mouse button and hold it to pan the figure, dragging it to a new position. When you release it, the data under the point where you pressed will be moved to the point where you released. If you press 'x' or 'y' while panning the motion will be constrained to the x or y axis, respectively. Press the right mouse button to zoom, dragging it to a new position. The x axis will be zoomed in proportionately to the rightward movement and zoomed out proportionately to the leftward movement. The same is true for the y axis and up/down motions. The point under your mouse when you begin the zoom remains stationary, allowing you to zoom in or out around that point as much as you wish. You can use the modifier keys 'x', 'y' or 'CONTROL' to constrain the zoom to the x axis, the y axis, or aspect ratio preserve, respectively.

With polar plots, the pan and zoom functionality behaves differently. The radius axis labels can be dragged using the left mouse button. The radius scale can be zoomed in and out using the right mouse button.



The Zoom-to-rectangle button Click this toolbar button to activate this mode. Put your mouse somewhere over an axes and press a mouse button. Define a rectangular region by dragging the mouse while holding the button to a new location. When using the left mouse button, the axes view limits will be zoomed to the defined region. When using the right mouse button, the axes view limits will be zoomed out, placing the original axes in the defined region.



The Subplot-configuration button Use this tool to configure the appearance of the subplot: you can stretch or compress the left, right, top, or bottom side of the subplot, or the space between the rows or space between the columns.



The Save button Click this button to launch a file save dialog. You can save files with the following extensions: png, ps, eps, svg and pdf.

4.1.1 Navigation Keyboard Shortcuts

The following table holds all the default keys, which can be overwritten by use of your matplotlibrc (#keymap.*).

Command	Keyboard Shortcut(s)
Home/Reset	h or r or home
Back	c or left arrow or backspace
Forward	v or right arrow
Pan/Zoom	p
Zoom-to-rect	o
Save	ctrl + s
Toggle fullscreen	f or ctrl + f
Close plot	ctrl + w
Close all plots	shift + w
Constrain pan/zoom to x axis	hold x when panning/zooming with mouse
Constrain pan/zoom to y axis	hold y when panning/zooming with mouse
Preserve aspect ratio	hold CONTROL when panning/zooming with mouse
Toggle major grids	g when mouse is over an axes
Toggle minor grids	G when mouse is over an axes
Toggle x axis scale (log/linear)	L or k when mouse is over an axes
Toggle y axis scale (log/linear)	l when mouse is over an axes

If you are using matplotlib.pyplot the toolbar will be created automatically for every figure. If you are writing your own user interface code, you can add the toolbar as a widget. The exact syntax depends on your UI, but we have examples for every supported UI in the matplotlib/examples/user_interfaces directory. Here is some example code for GTK:

```
import gtk

from matplotlib.figure import Figure
from matplotlib.backends.backend_gtkagg import FigureCanvasGTKAgg as FigureCanvas
from matplotlib.backends.backend_gtkagg import NavigationToolbar2GTKAgg as
↳NavigationToolbar

win = gtk.Window()
```

(continues on next page)

(continued from previous page)

```
win.connect("destroy", lambda x: gtk.main_quit())
win.set_default_size(400,300)
win.set_title("Embedding in GTK")

vbox = gtk.VBox()
win.add(vbox)

fig = Figure(figsize=(5,4), dpi=100)
ax = fig.add_subplot(111)
ax.plot([1,2,3])

canvas = FigureCanvas(fig) # a gtk.DrawingArea
vbox.pack_start(canvas)
toolbar = NavigationToolbar(canvas, win)
vbox.pack_start(toolbar, False, False)

win.show_all()
gtk.main()
```

4.2 Using matplotlib in a python shell

Warning: This page is significantly out of date

By default, matplotlib defers drawing until the end of the script because drawing can be an expensive operation, and you may not want to update the plot every time a single property is changed, only once after all the properties have changed.

But when working from the python shell, you usually do want to update the plot with every command, e.g., after changing the xlabel(), or the marker style of a line. While this is simple in concept, in practice it can be tricky, because matplotlib is a graphical user interface application under the hood, and there are some tricks to make the applications work right in a python shell.

4.2.1 IPython to the rescue

Note: The mode described here still exists for historical reasons, but it is highly advised not to use. It pollutes namespaces with functions that will shadow python built-in and can lead to hard to track bugs. To get IPython integration without imports the use of the %matplotlib magic is preferred. See [ipython documentation](#) .

Fortunately, [ipython](#), an enhanced interactive python shell, has figured out all of these tricks, and is matplotlib aware, so when you start ipython in the pylab mode.

```
johnh@flag:~> ipython
Python 2.4.5 (#4, Apr 12 2008, 09:09:16)
IPython 0.9.0 -- An enhanced Interactive Python.
```

```
In [1]: %pylab
```

```
Welcome to pylab, a matplotlib-based Python environment.
For more information, type 'help(pylab)'.
```

```
In [2]: x = randn(10000)
```

```
In [3]: hist(x, 100)
```

it sets everything up for you so interactive plotting works as you would expect it to. Call `figure()` and a figure window pops up, call `plot()` and your data appears in the figure window.

Note in the example above that we did not import any matplotlib names because in pylab mode, ipython will import them automatically. ipython also turns on interactive mode for you, which causes every pyplot command to trigger a figure update, and also provides a matplotlib aware run command to run matplotlib scripts efficiently. ipython will turn off interactive mode during a run command, and then restore the interactive state at the end of the run so you can continue tweaking the figure manually.

There has been a lot of recent work to embed ipython, with pylab support, into various GUI applications, so check on the ipython mailing [list](#) for the latest status.

4.2.2 Other python interpreters

If you can't use ipython, and still want to use matplotlib/pylab from an interactive python shell, e.g., the plain-ole standard python interactive interpreter, you are going to need to understand what a matplotlib backend is [What is a backend?](#).

With the TkAgg backend, which uses the Tkinter user interface toolkit, you can use matplotlib from an arbitrary non-gui python shell. Just set your backend : TkAgg and interactive : True in your matplotlibrc file (see [Customizing Matplotlib with style sheets and rcParams](#)) and fire up python. Then:

```
>>> from pylab import *
>>> plot([1,2,3])
>>> xlabel('hi mom')
```

should work out of the box. This is also likely to work with recent versions of the qt4agg and gtkagg backends, and with the macosx backend on the Macintosh. Note, in batch mode, i.e. when making figures from scripts, interactive mode can be slow since it redraws the figure with each command. So you may want to think carefully before making this the default behavior via the matplotlibrc file instead of using the functions listed in the next section.

Gui shells are at best problematic, because they have to run a mainloop, but interactive plotting also involves a mainloop. Ipython has sorted all this out for the primary matplotlib backends.

There may be other shells and IDEs that also work with matplotlib in interactive mode, but one obvious candidate does not: the python IDLE IDE is a Tkinter gui app that does not support pylab interactive mode, regardless of backend.

4.2.3 Controlling interactive updating

The interactive property of the pyplot interface controls whether a figure canvas is drawn on every pyplot command. If interactive is False, then the figure state is updated on every plot command, but will only be drawn on explicit calls to draw(). When interactive is True, then every pyplot command triggers a draw.

The pyplot interface provides 4 commands that are useful for interactive control.

isinteractive() returns the interactive setting True|False

ion() turns interactive mode on

ioff() turns interactive mode off

draw() forces a figure redraw

When working with a big figure in which drawing is expensive, you may want to turn matplotlib's interactive setting off temporarily to avoid the performance hit:

```
>>> #create big-expensive-figure
>>> ioff()    # turn updates off
>>> title('now how much would you pay?')
>>> xticklabels(fontsize=20, color='green')
>>> draw()    # force a draw
>>> savefig('alldone', dpi=300)
>>> close()
>>> ion()     # turn updating back on
>>> plot(rand(20), mfc='g', mec='r', ms=40, mew=4, ls='--', lw=3)
```

4.3 Event handling and picking

matplotlib works with a number of user interface toolkits (wxpython, tkinter, qt4, gtk, and macosx) and in order to support features like interactive panning and zooming of figures, it is helpful to the developers to have an API for interacting with the figure via key presses and mouse movements that is "GUI neutral" so we don't have to repeat a lot of code across the different user interfaces. Although the event handling API is GUI neutral, it is based on the GTK model, which was the first user interface matplotlib supported. The events that are triggered are also a bit richer vis-a-vis matplotlib than standard GUI events, including information like which matplotlib.axes.Axes the event occurred in. The events also understand the matplotlib coordinate system, and report event locations in both pixel and data coordinates.

4.3.1 Event connections

To receive events, you need to write a callback function and then connect your function to the event manager, which is part of the FigureCanvasBase. Here is a simple example that prints the location of the mouse click and which button was pressed:

```
fig, ax = plt.subplots()
ax.plot(np.random.rand(10))

def onclick(event):
    print('%s click: button=%d, x=%d, y=%d, xdata=%f, ydata=%f' %
          ('double' if event.dblclick else 'single', event.button,
            event.x, event.y, event.xdata, event.ydata))

cid = fig.canvas.mpl_connect('button_press_event', onclick)
```

The FigureCanvas method `mpl_connect()` returns a connection id which is simply an integer. When you want to disconnect the callback, just call:

```
fig.canvas.mpl_disconnect(cid)
```

Note: The canvas retains only weak references to the callbacks. Therefore if a callback is a method of a class instance, you need to retain a reference to that instance. Otherwise the instance will be garbage-collected and the callback will vanish.

Here are the events that you can connect to, the class instances that are sent back to you when the event occurs, and the event descriptions

Event name	Class and description
'button_press_event'	MouseEvent - mouse button is pressed
'button_release_event'	MouseEvent - mouse button is released
'draw_event'	DrawEvent - canvas draw (but before screen update)
'key_press_event'	KeyEvent - key is pressed
'key_release_event'	KeyEvent - key is released
'motion_notify_event'	MouseEvent - mouse motion
'pick_event'	PickEvent - an object in the canvas is selected
'resize_event'	ResizeEvent - figure canvas is resized
'scroll_event'	MouseEvent - mouse scroll wheel is rolled
'figure_enter_event'	LocationEvent - mouse enters a new figure
'figure_leave_event'	LocationEvent - mouse leaves a figure
'axes_enter_event'	LocationEvent - mouse enters a new axes
'axes_leave_event'	LocationEvent - mouse leaves an axes

4.3.2 Event attributes

All matplotlib events inherit from the base class `matplotlib.backend_bases.Event`, which store the attributes:

- `name` the event name
- `canvas` the `FigureCanvas` instance generating the event
- `guiEvent` the GUI event that triggered the matplotlib event

The most common events that are the bread and butter of event handling are key press/release events and mouse press/release and movement events. The `KeyEvent` and `MouseEvent` classes that handle these events are both derived from the `LocationEvent`, which has the following attributes

- `x` x position - pixels from left of canvas
- `y` y position - pixels from bottom of canvas
- `inaxes` the `Axes` instance if mouse is over axes
- `xdata` x coord of mouse in data coords
- `ydata` y coord of mouse in data coords

Let's look a simple example of a canvas, where a simple line segment is created every time a mouse is pressed:

```
from matplotlib import pyplot as plt

class LineBuilder:
    def __init__(self, line):
        self.line = line
        self.xs = list(line.get_xdata())
        self.ys = list(line.get_ydata())
        self.cid = line.figure.canvas.mpl_connect('button_press_event', self)

    def __call__(self, event):
        print('click', event)
        if event.inaxes != self.line.axes: return
        self.xs.append(event.xdata)
        self.ys.append(event.ydata)
        self.line.set_data(self.xs, self.ys)
        self.line.figure.canvas.draw()

fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_title('click to build line segments')
line, = ax.plot([0], [0]) # empty line
linebuilder = LineBuilder(line)

plt.show()
```

The `MouseEvent` that we just used is a `LocationEvent`, so we have access to the data and pixel coordinates in `event.x` and `event.xdata`. In addition to the `LocationEvent` attributes, it has

`button` button pressed: None, 1, 2, 3, 'up', 'down' (up and down are used for scroll events)
`key` the key pressed: None, any character, 'shift', 'win', or 'control'

Draggable rectangle exercise

Write draggable rectangle class that is initialized with a `Rectangle` instance but will move its `x,y` location when dragged. Hint: you will need to store the original `xy` location of the rectangle which is stored as `rect.xy` and connect to the `press`, `motion` and `release` mouse events. When the mouse is pressed, check to see if the click occurs over your rectangle (see `matplotlib.patches.Rectangle.contains()`) and if it does, store the rectangle `xy` and the location of the mouse click in data coords. In the `motion` event callback, compute the `deltax` and `delay` of the mouse movement, and add those deltas to the origin of the rectangle you stored. The redraw the figure. On the button release event, just reset all the button press data you stored as `None`.

Here is the solution:

```
import numpy as np
import matplotlib.pyplot as plt

class DraggableRectangle:
    def __init__(self, rect):
        self.rect = rect
        self.press = None

    def connect(self):
        'connect to all the events we need'
        self.cidpress = self.rect.figure.canvas.mpl_connect(
            'button_press_event', self.on_press)
        self.cidrelease = self.rect.figure.canvas.mpl_connect(
            'button_release_event', self.on_release)
        self.cidmotion = self.rect.figure.canvas.mpl_connect(
            'motion_notify_event', self.on_motion)

    def on_press(self, event):
        'on button press we will see if the mouse is over us and store some data'
        if event.inaxes != self.rect.axes: return

        contains, attrd = self.rect.contains(event)
        if not contains: return
        print('event contains', self.rect.xy)
        x0, y0 = self.rect.xy
        self.press = x0, y0, event.xdata, event.ydata
```

(continues on next page)

(continued from previous page)

```

def on_motion(self, event):
    'on motion we will move the rect if the mouse is over us'
    if self.press is None: return
    if event.inaxes != self.rect.axes: return
    x0, y0, xpress, ypress = self.press
    dx = event.xdata - xpress
    dy = event.ydata - ypress
    #print('x0=%f, xpress=%f, event.xdata=%f, dx=%f, x0+dx=%f' %
    #      (x0, xpress, event.xdata, dx, x0+dx))
    self.rect.set_x(x0+dx)
    self.rect.set_y(y0+dy)

    self.rect.figure.canvas.draw()

def on_release(self, event):
    'on release we reset the press data'
    self.press = None
    self.rect.figure.canvas.draw()

def disconnect(self):
    'disconnect all the stored connection ids'
    self.rect.figure.canvas.mpl_disconnect(self.cidpress)
    self.rect.figure.canvas.mpl_disconnect(self.cidrelease)
    self.rect.figure.canvas.mpl_disconnect(self.cidmotion)

fig = plt.figure()
ax = fig.add_subplot(111)
rects = ax.bar(range(10), 20*np.random.rand(10))
drs = []
for rect in rects:
    dr = DraggableRectangle(rect)
    dr.connect()
    drs.append(dr)

plt.show()

```

Extra credit: use the animation blit techniques discussed in the [animations recipe](#) to make the animated drawing faster and smoother.

Extra credit solution:

```

# draggable rectangle with the animation blit techniques; see
# http://www.scipy.org/Cookbook/Matplotlib/Animations
import numpy as np
import matplotlib.pyplot as plt

```

(continues on next page)

(continued from previous page)

```

class DraggableRectangle:
    lock = None # only one can be animated at a time
    def __init__(self, rect):
        self.rect = rect
        self.press = None
        self.background = None

    def connect(self):
        'connect to all the events we need'
        self.cidpress = self.rect.figure.canvas.mpl_connect(
            'button_press_event', self.on_press)
        self.cidrelease = self.rect.figure.canvas.mpl_connect(
            'button_release_event', self.on_release)
        self.cidmotion = self.rect.figure.canvas.mpl_connect(
            'motion_notify_event', self.on_motion)

    def on_press(self, event):
        'on button press we will see if the mouse is over us and store some data'
        if event.inaxes != self.rect.axes: return
        if DraggableRectangle.lock is not None: return
        contains, attrd = self.rect.contains(event)
        if not contains: return
        print('event contains', self.rect.xy)
        x0, y0 = self.rect.xy
        self.press = x0, y0, event.xdata, event.ydata
        DraggableRectangle.lock = self

        # draw everything but the selected rectangle and store the pixel buffer
        canvas = self.rect.figure.canvas
        axes = self.rect.axes
        self.rect.set_animated(True)
        canvas.draw()
        self.background = canvas.copy_from_bbox(self.rect.axes.bbox)

        # now redraw just the rectangle
        axes.draw_artist(self.rect)

        # and blit just the redrawn area
        canvas.blit(axes.bbox)

    def on_motion(self, event):
        'on motion we will move the rect if the mouse is over us'
        if DraggableRectangle.lock is not self:
            return

```

(continues on next page)

(continued from previous page)

```

    if event.inaxes != self.rect.axes: return
    x0, y0, xpress, ypress = self.press
    dx = event.xdata - xpress
    dy = event.ydata - ypress
    self.rect.set__x(x0+dx)
    self.rect.set__y(y0+dy)

    canvas = self.rect.figure.canvas
    axes = self.rect.axes
    # restore the background region
    canvas.restore__region(self.background)

    # redraw just the current rectangle
    axes.draw__artist(self.rect)

    # blit just the redrawn area
    canvas.blit(axes.bbox)

def on__release(self, event):
    'on release we reset the press data'
    if DraggableRectangle.lock is not self:
        return

    self.press = None
    DraggableRectangle.lock = None

    # turn off the rect animation property and reset the background
    self.rect.set__animated(False)
    self.background = None

    # redraw the full figure
    self.rect.figure.canvas.draw()

def disconnect(self):
    'disconnect all the stored connection ids'
    self.rect.figure.canvas.mpl__disconnect(self.cidpress)
    self.rect.figure.canvas.mpl__disconnect(self.cidrelease)
    self.rect.figure.canvas.mpl__disconnect(self.cidmotion)

fig = plt.figure()
ax = fig.add_subplot(111)
rects = ax.bar(range(10), 20*np.random.rand(10))
drs = []
for rect in rects:
    dr = DraggableRectangle(rect)

```

(continues on next page)

(continued from previous page)

```

dr.connect()
drs.append(dr)

plt.show()

```

4.3.3 Mouse enter and leave

If you want to be notified when the mouse enters or leaves a figure or axes, you can connect to the figure/axes enter/leave events. Here is a simple example that changes the colors of the axes and figure background that the mouse is over:

```

"""
Illustrate the figure and axes enter and leave events by changing the
frame colors on enter and leave
"""

import matplotlib.pyplot as plt

def enter_axes(event):
    print('enter_axes', event.inaxes)
    event.inaxes.patch.set_facecolor('yellow')
    event.canvas.draw()

def leave_axes(event):
    print('leave_axes', event.inaxes)
    event.inaxes.patch.set_facecolor('white')
    event.canvas.draw()

def enter_figure(event):
    print('enter_figure', event.canvas.figure)
    event.canvas.figure.patch.set_facecolor('red')
    event.canvas.draw()

def leave_figure(event):
    print('leave_figure', event.canvas.figure)
    event.canvas.figure.patch.set_facecolor('grey')
    event.canvas.draw()

fig1 = plt.figure()
fig1.suptitle('mouse hover over figure or axes to trigger events')
ax1 = fig1.add_subplot(211)
ax2 = fig1.add_subplot(212)

fig1.canvas.mpl_connect('figure_enter_event', enter_figure)
fig1.canvas.mpl_connect('figure_leave_event', leave_figure)
fig1.canvas.mpl_connect('axes_enter_event', enter_axes)

```

(continues on next page)

(continued from previous page)

```
fig1.canvas.mpl_connect('axes_leave_event', leave_axes)

fig2 = plt.figure()
fig2.suptitle('mouse hover over figure or axes to trigger events')
ax1 = fig2.add_subplot(211)
ax2 = fig2.add_subplot(212)

fig2.canvas.mpl_connect('figure_enter_event', enter_figure)
fig2.canvas.mpl_connect('figure_leave_event', leave_figure)
fig2.canvas.mpl_connect('axes_enter_event', enter_axes)
fig2.canvas.mpl_connect('axes_leave_event', leave_axes)

plt.show()
```

4.3.4 Object picking

You can enable picking by setting the picker property of an Artist (e.g., a matplotlib Line2D, Text, Patch, Polygon, AxesImage, etc...)

There are a variety of meanings of the picker property:

- None picking is disabled for this artist (default)
- boolean if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- float if picker is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if its data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event.
- function if picker is callable, it is a user supplied function which determines whether the artist is hit by the mouse event. The signature is `hit, props = picker(artist, mouseevent)` to determine the hit test. If the mouse event is over the artist, return `hit=True` and `props` is a dictionary of properties you want added to the `PickEvent` attributes

After you have enabled an artist for picking by setting the picker property, you need to connect to the figure canvas `pick_event` to get pick callbacks on mouse press events. e.g.:

```
def pick_handler(event):
    mouseevent = event.mouseevent
    artist = event.artist
    # now do something with this...
```

The `PickEvent` which is passed to your callback is always fired with two attributes:

`mouseevent` the mouse event that generate the pick event. The `mouse event` in turn has attributes like `x` and `y` (the coords in display space, e.g., pixels from left, bottom) and `xdata`, `ydata` (the coords in data space). Additionally, you can get information about which buttons were pressed, which keys were pressed, which Axes the mouse is over, etc. See `matplotlib.backend_bases.MouseEvent` for details.

`artist` the Artist that generated the pick event.

Additionally, certain artists like `Line2D` and `PatchCollection` may attach additional meta data like the indices into the data that meet the picker criteria (e.g., all the points in the line that are within the specified epsilon tolerance)

Simple picking example

In the example below, we set the line picker property to a scalar, so it represents a tolerance in points (72 points per inch). The `onpick` callback function will be called when the pick event is within the tolerance distance from the line, and has the indices of the data vertices that are within the pick distance tolerance. Our `onpick` callback function simply prints the data that are under the pick location. Different matplotlib Artists can attach different data to the `PickEvent`. For example, `Line2D` attaches the `ind` property, which are the indices into the line data under the pick point. See `pick()` for details on the `PickEvent` properties of the line. Here is the code:

```
import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_title('click on points')

line, = ax.plot(np.random.rand(100), 'o', picker=5) # 5 points tolerance

def onpick(event):
    thisline = event.artist
    xdata = thisline.get_xdata()
    ydata = thisline.get_ydata()
    ind = event.ind
    points = tuple(zip(xdata[ind], ydata[ind]))
    print('onpick points:', points)

fig.canvas.mpl_connect('pick_event', onpick)

plt.show()
```


Picking exercise

Create a data set of 100 arrays of 1000 Gaussian random numbers and compute the sample mean and standard deviation of each of them (hint: numpy arrays have a mean and std method) and make a xy marker plot of the 100 means vs the 100 standard deviations. Connect the line created by the plot command to the pick event, and plot the original time series of the data that generated the clicked on points. If more than one point is within the tolerance of the clicked on point, you can use multiple subplots to plot the multiple time series.

Exercise solution:

```
"""
compute the mean and stddev of 100 data sets and plot mean vs stddev.
When you click on one of the mu, sigma points, plot the raw data from
the dataset that generated the mean and stddev
"""

import numpy as np
import matplotlib.pyplot as plt

X = np.random.rand(100, 1000)
xs = np.mean(X, axis=1)
ys = np.std(X, axis=1)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_title('click on point to plot time series')
line, = ax.plot(xs, ys, 'o', picker=5) # 5 points tolerance

def onpick(event):

    if event.artist!=line: return True

    N = len(event.ind)
    if not N: return True

    figi = plt.figure()
    for subplotnum, dataind in enumerate(event.ind):
        ax = figi.add_subplot(N,1,subplotnum+1)
        ax.plot(X[dataind])
        ax.text(0.05, 0.9, 'mu=%1.3f\sigma=%1.3f'%(xs[dataind], ys[dataind]),
                transform=ax.transAxes, va='top')
        ax.set_ylim(-0.5, 1.5)
    figi.show()
    return True
```

(continues on next page)

(continued from previous page)

```
fig.canvas.mpl_connect('pick_event', onpick)  
  
plt.show()
```


WHAT'S NEW IN MATPLOTLIB

For a list of all of the issues and pull requests since the last revision, see the [GitHub Stats](#).

Table of Contents

- What's new in Matplotlib
 - New in Matplotlib 2.2.4
 - * `:math:` directive renamed to `:mathmpl:`
 - New in Matplotlib 2.2
 - * Constrained Layout Manager
 - New `plt.figure` and `plt.subplots` kwarg: `constrained_layout`
 - New `ax.set__position` behaviour
 - New figure kwarg for `GridSpec`
 - * `xlabels` and `ylabels` can now be automatically aligned
 - * Axes legends now included in `tight_bbox`
 - * `Cividis` colormap
 - * New style colorblind-friendly color cycle
 - * Support for `numpy.datetime64`
 - * Writing animations with `Pillow`
 - * Slider UI widget can snap to discrete values
 - * `capstyle` and `joinstyle` attributes added to `Collection`
 - * `pad` kwarg added to `ax.set_title`
 - * Comparison of 2 colors in Matplotlib
 - * Autoscaling a polar plot snaps to the origin
 - * `PathLike` support
 - * `Axes.tick_params` can set gridline properties

- * `Axes.imshow` clips RGB values to the valid range
 - * Properties in `matplotlibrc` to place xaxis and yaxis tick labels
 - * PGI bindings for `gtk3`
 - * Cairo rendering for Qt, WX, and Tk canvases
 - * Added support for QT in new `ToolManager`
 - * TkAgg backend reworked to support PyPy
 - * Python logging library used for debug output
 - * Improved repr for `Transforms`
- [Previous Whats New](#)

5.1 New in Matplotlib 2.2.4

5.1.1 `:math:` directive renamed to `:mathmpl:`

The `:math:` rst role provided by `matplotlib.sphinxext.mathmpl` has been renamed to `:mathmpl:` to avoid conflicting with the `:math:` role that Sphinx 1.8 provides by default. (`:mathmpl:` uses Matplotlib to render math expressions to images embedded in html, whereas Sphinx uses MathJax.)

When using Sphinx<1.8, both names (`:math:` and `:mathmpl:`) remain available for backcompatibility.

5.2 New in Matplotlib 2.2

5.2.1 Constrained Layout Manager

Warning: Constrained Layout is experimental. The behaviour and API are subject to change, or the whole functionality may be removed without a deprecation period.

A new method to automatically decide spacing between subplots and their organizing `GridSpec` instances has been added. It is meant to replace the venerable `tight_layout` method. It is invoked via a new `constrained_layout=True` kwarg to `Figure` or subplots.

There are new rcParams for this package, and spacing can be more finely tuned with the new `set_constrained_layout_pads`.

Features include:

- Automatic spacing for subplots with a fixed-size padding in inches around subplots and all their decorators, and space between as a fraction of subplot size between subplots.

- Spacing for suptitle, and colorbars that are attached to more than one axes.
- Nested GridSpec layouts using GridSpecFromSubplotSpec.

For more details and capabilities please see the new tutorial: [Constrained Layout Guide](#)

Note the new API to access this:

New `plt.figure` and `plt.subplots` kwarg: `constrained_layout`

`figure()` and `subplots()` can now be called with `constrained_layout=True` kwarg to enable `constrained_layout`.

New `ax.set__position` behaviour

`set__position()` now makes the specified axis no longer responsive to `constrained_layout`, consistent with the idea that the user wants to place an axis manually.

Internally, this means that old `ax.set__position` calls inside the library are changed to private `ax.__set__position` calls so that `constrained_layout` will still work with these axes.

New figure kwarg for GridSpec

In order to facilitate `constrained_layout`, GridSpec now accepts a figure keyword. This is backwards compatible, in that not supplying this will simply cause `constrained_layout` to not operate on the subplots organized by this GridSpec instance. Routines that use GridSpec (e.g. `fig.subplots`) have been modified to pass the figure to GridSpec.

5.2.2 xlabels and ylabels can now be automatically aligned

Subplot axes ylabels can be misaligned horizontally if the tick labels are very different widths. The same can happen to xlabels if the ticklabels are rotated on one subplot (for instance). The new methods on the Figure class: `Figure.align_xlabels` and `Figure.align_ylabels` will now align these labels horizontally or vertically. If the user only wants to align some axes, a list of axes can be passed. If no list is passed, the algorithm looks at all the labels on the figure.

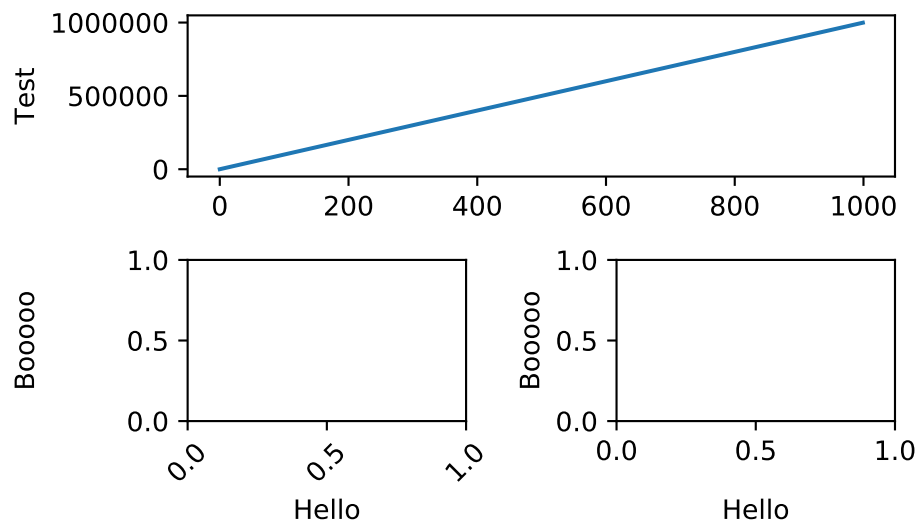
Only labels that have the same subplot locations are aligned. i.e. the ylabels are aligned only if the subplots are in the same column of the subplot layout.

Alignment is persistent and automatic after these are called.

A convenience wrapper `Figure.align_labels` calls both functions at once.

5.2.3 Axes legends now included in `tight_bbox`

Legends created via `ax.legend` can sometimes overspill the limits of the axis. Tools like `fig.tight_layout()` and `fig.savefig(bbox_inches='tight')` would clip these legends. A change was made to include them in the tight calculations.



5.2.4 Cividis colormap

A new dark blue/yellow colormap named 'cividis' was added. Like viridis, cividis is perceptually uniform and colorblind friendly. However, cividis also goes a step further: not only is it usable by colorblind users, it should actually look effectively identical to colorblind and non-colorblind users. For more details see [Nuñez J, Anderton C, and Renslow R: "Optimizing colormaps with consideration for color vision deficiency to enable accurate interpretation of scientific data"](#).

5.2.5 New style colorblind-friendly color cycle

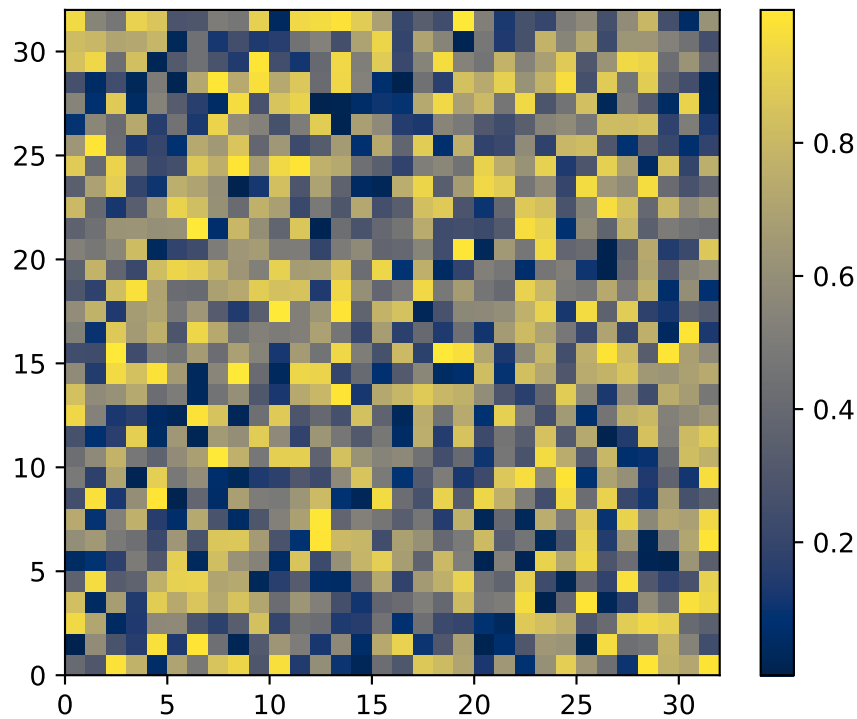
A new style defining a color cycle has been added, `tableau-colorblind10`, to provide another option for colorblind-friendly plots. A demonstration of this new style can be found in the [reference](#) of style sheets. To load this color cycle in place of the default one:

```
import matplotlib.pyplot as plt
plt.style.use('tableau-colorblind10')
```

5.2.6 Support for `numpy.datetime64`

Matplotlib has supported `datetime.datetime` dates for a long time in `matplotlib.dates`. We now support `numpy.datetime64` dates as well. Anywhere that `datetime.datetime` could be used, `numpy.datetime64` can be used. eg:

```
time = np.arange('2005-02-01', '2005-02-02', dtype='datetime64[h]')
plt.plot(time)
```



5.2.7 Writing animations with Pillow

It is now possible to use Pillow as an animation writer. Supported output formats are currently gif (Pillow \geq 3.4) and webp (Pillow \geq 5.0). Use e.g. as

```
from __future__ import division

from matplotlib import pyplot as plt
from matplotlib.animation import FuncAnimation, PillowWriter

fig, ax = plt.subplots()
line, = plt.plot([0, 1])

def animate(i):
    line.set_ydata([0, i / 20])
    return [line]

anim = FuncAnimation(fig, animate, 20, blit=True)
anim.save("movie.gif", writer=PillowWriter(fps=24))
plt.show()
```


5.2.8 Slider UI widget can snap to discrete values

The slider UI widget can take the optional argument `valstep`. Doing so forces the slider to take on only discrete values, starting from `valmin` and counting up to `valmax` with steps of size `valstep`.

If `closedmax==True`, then the slider will snap to `valmax` as well.

5.2.9 `capstyle` and `joinstyle` attributes added to `Collection`

The `Collection` class now has customizable `capstyle` and `joinstyle` attributes. This allows the user for example to set the `capstyle` of errorbars.

5.2.10 `pad` kwarg added to `ax.set_title`

The method `axes.set_title` now has a `pad` kwarg, that specifies the distance from the top of an axes to where the title is drawn. The units of `pad` is points, and the default is the value of the (already-existing) `rcParams['axes.titlepad']`.

5.2.11 Comparison of 2 colors in Matplotlib

As the colors in Matplotlib can be specified with a wide variety of ways, the `matplotlib.colors.same_color` method has been added which checks if two colors are the same.

5.2.12 Autoscaling a polar plot snaps to the origin

Setting the limits automatically in a polar plot now snaps the radial limit to zero if the automatic limit is nearby. This means plotting from zero doesn't automatically scale to include small negative values on the radial axis.

The limits can still be set manually in the usual way using `set_ylim`.

5.2.13 `PathLike` support

On Python 3.6+, `savefig`, `imsave`, `imread`, and animation writers now accept `os.PathLikes` as input.

5.2.14 `Axes.tick_params` can set gridline properties

Tick objects hold gridlines as well as the tick mark and its label. `Axis.set_tick_params`, `Axes.tick_params` and `pyplot.tick_params` now have keyword arguments `'grid_color'`, `'grid_alpha'`, `'grid_linewidth'`, and `'grid_linestyle'` for overriding the defaults in `rcParams`: `'grid.color'`, etc.

5.2.15 Axes.imshow clips RGB values to the valid range

When `Axes.imshow` is passed an RGB or RGBA value with out-of-range values, it now logs a warning and clips them to the valid range. The old behaviour, wrapping back in to the range, often hid outliers and made interpreting RGB images unreliable.

5.2.16 Properties in matplotlibrc to place xaxis and yaxis tick labels

Introducing four new boolean properties in `matplotlibrc` for default positions of xaxis and yaxis tick labels, namely, `xtick.labeltop`, `xtick.labelbottom`, `ytick.labelright` and `ytick.labelleft`. These can also be changed in `rcParams`.

5.2.17 PGI bindings for gtk3

The GTK3 backends can now use `PGI` instead of `PyGObject`. `PGI` is a fairly incomplete binding for `GObject`, thus its use is not recommended; its main benefit is its availability on Travis (thus allowing CI testing for the `gtk3agg` and `gtk3cairo` backends).

The binding selection rules are as follows: - if `gi` has already been imported, use it; else - if `pgi` has already been imported, use it; else - if `gi` can be imported, use it; else - if `pgi` can be imported, use it; else - error out.

Thus, to force usage of `PGI` when both bindings are installed, import it first.

5.2.18 Cairo rendering for Qt, WX, and Tk canvases

The new `Qt4Cairo`, `Qt5Cairo`, `WXCairo`, and `TkCairo` backends allow Qt, Wx, and Tk canvases to use Cairo rendering instead of Agg.

5.2.19 Added support for QT in new ToolManager

Now it is possible to use the `ToolManager` with Qt5 For example

```
import matplotlib
matplotlib.use('QT5AGG') matplotlib.rcParams['toolbar'] = 'toolmanager' import
matplotlib.pyplot as plt
plt.plot([1,2,3]) plt.show()
```

Treat the new Tool classes experimental for now, the API will likely change and perhaps the `rcParam` as well

The main example `examples/user_interfaces/toolmanager_sgskip.py` shows more details, just adjust the header to use QT instead of GTK3

5.2.20 TkAgg backend reworked to support PyPy

PyPy can now plot using the TkAgg backend, supported on PyPy 5.9 and greater (both PyPy for python 2.7 and PyPy for python 3.5).

5.2.21 Python logging library used for debug output

Matplotlib has in the past (sporadically) used an internal verbose-output reporter. This version converts those calls to using the standard python `logging` library.

Support for the old rcParams `verbose.level` and `verbose.fileo` is dropped.

The command-line options `--verbose-helpful` and `--verbose-debug` are still accepted, but deprecated. They are now equivalent to setting `logging.INFO` and `logging.DEBUG`.

The logger's root name is `matplotlib` and can be accessed from programs as:

```
import logging
mlog = logging.getLogger('matplotlib')
```

Instructions for basic usage are in [Troubleshooting](#) and for developers in [Contributing](#).

5.2.22 Improved repr for Transforms

Transforms now indent their `reprs` in a more legible manner:

```
In [1]: l, = plt.plot([]); l.get_transform()
Out[1]:
CompositeGenericTransform(
  TransformWrapper(
    BlendedAffine2D(
      IdentityTransform(),
      IdentityTransform()),
  CompositeGenericTransform(
    BboxTransformFrom(
      TransformedBbox(
        Bbox(x0=-0.055000000000000001, y0=-0.055000000000000001, x1=0.
→055000000000000001, y1=0.055000000000000001),
      TransformWrapper(
        BlendedAffine2D(
          IdentityTransform(),
          IdentityTransform()))),
    BboxTransformTo(
      TransformedBbox(
        Bbox(x0=0.125, y0=0.10999999999999999, x1=0.9, y1=0.88),
        BboxTransformTo(
          TransformedBbox(
```

(continues on next page)

- 2014-05-16 Fixed the positioning of multi-line text in the PGF backend.
- 2014-05-14 Added `Axes.add_image()` as the standard way to add `AxesImage` instances to `Axes`. This improves the consistency with `add_artist()`, `add_collection()`, `add_container()`, `add_line()`, `add_patch()`, and `add_table()`.
- 2014-05-02 Added colorblind-friendly colormap, named 'Wistia'.
- 2014-04-27 Improved input clean up in `Axes.{h|v}lines` Coerce input into a 1D ndarrays (after dealing with units).
- 2014-04-27 removed un-needed cast to float in stem
- 2014-04-23 Updated references to "ipython -pylab" The preferred method for invoking pylab is now using the "%pylab" magic. -Chris G.
- 2014-04-22 Added (re-)generate a simple automatic legend to "Figure Options" dialog of the Qt4Agg backend.
- 2014-04-22 Added an example showing the difference between `interpolation = 'none'` and `interpolation = 'nearest'` in `imshow()` when saving vector graphics files.
- 2014-04-22 Added violin plotting functions. See `Axes.violinplot`, `Axes.violin`, `cbook.violin_stats` and `mlab.GaussianKDE` for details.
- 2014-04-10 Fixed the triangular marker rendering error. The "Up" triangle was rendered instead of "Right" triangle and vice-versa.
- 2014-04-08 Fixed a bug in `parasite_axes.py` by making a list out of a generator at line 263.
- 2014-04-02 Added `clipon=False` to patch creation of wedges and shadows in pie.
- 2014-02-25 In `backend_qt4agg` changed from using `update -> repaint` under windows. See comment in source near `self.__priv_update` for longer explanation.
- 2014-03-27 Added tests for pie `ccw` parameter. Removed pdf and svg images from tests for pie `linewidth` parameter.
- 2014-03-24 Changed the behaviour of axes to not ignore leading or trailing patches of height 0 (or width 0) while calculating the x and y axis limits. Patches having both `height == 0` and `width == 0` are ignored.
- 2014-03-24 Added bool kwarg (`manage_xticks`) to `boxplot` to enable/disable the management of the xlimits and ticks when making a boxplot. Default in True which maintains current behavior by default.
- 2014-03-23 Fixed a bug in `projections/polar.py` by making sure that the `theta` value being calculated when given the mouse coordinates stays within the range of 0 and $2 * \pi$.
- 2014-03-22 Added the keyword arguments `wedgeprops` and `textprops` to pie. Users can control the wedge and text properties of the pie in more detail, if they choose.
- 2014-03-17 Bug was fixed in `append_axes` from the `AxesDivider` class would not append axes in the right location with respect to the reference locator axes
- 2014-03-13 Add parameter 'clockwise' to function pie, True by default.
- 2014-02-28 Added 'origin' kwarg to spy

- 2014-02-27 Implemented separate horizontal/vertical axes padding to the `ImageGrid` in the `Axes-Grid` toolkit
- 2014-02-27 Allowed `markevery` property of `matplotlib.lines.Line2D` to be, an `int`, `numpy` fancy index, slice object, or float. The float behaviour turns on markers at approximately equal display-coordinate-distances along the line.
- 2014-02-25 In `backend_qt4agg` changed from using `update` -> `repaint` under windows. See comment in source near `self._priv_update` for longer explanation.
- 2014-01-02 `triplot` now returns the artist it adds and support of line and marker kwargs has been improved. GBY
- 2013-12-30 Made `streamplot` grid size consistent for different types of `density` argument. A 30x30 grid is now used for both `density=1` and `density=(1, 1)`.
- 2013-12-03 Added a pure boxplot-drawing method that allow a more complete customization of boxplots. It takes a list of dicts contains stats. Also created a function (`cbook.boxplot_stats`) that generates the stats needed.
- 2013-11-28 Added `qhull` extension module to perform Delaunay triangulation more robustly than before. It is used by `tri.Triangulation` (and hence all `pyplot.tri*` methods) and `mlab.griddata`. Deprecated `matplotlib.delaunay` module. - IMT
- 2013-11-05 Add power-law normalization method. This is useful for, e.g., showing small populations in a "hist2d" histogram.
- 2013-10-27 Added `get_rlabel_position` and `set_rlabel_position` methods to `PolarAxes` to control angular position of radial tick labels.
- 2013-10-06 Add stride-based functions to `mlab` for easy creation of 2D arrays with less memory.
- 2013-10-06 Improve window and `detrend` functions in `mlab`, particular support for 2D arrays.
- 2013-10-06 Improve performance of all spectrum-related `mlab` functions and plots.
- 2013-10-06 Added support for magnitude, phase, and angle spectrums to `axes.specgram`, and support for magnitude, phase, angle, and complex spectrums to `mlab-specgram`.
- 2013-10-06 Added `magnitude_spectrum`, `angle_spectrum`, and `phase_spectrum` plots, as well as `magnitude_spectrum`, `angle_spectrum`, `phase_spectrum`, and `complex_spectrum` functions to `mlab`
- 2013-07-12 Added support for datetime axes to 2d plots. Axis values are passed through `Axes.convert_xunits/Axes.convert_yunits` before being used by `contour/contourf`, `pcolormesh` and `pcolor`.
- 2013-07-12 Allowed `matplotlib.dates.date2num`, `matplotlib.dates.num2date`, and `matplotlib.dates.datestr2num` to accept n-d inputs. Also factored in support for n-d arrays to `matplotlib.dates.DateConverter` and `matplotlib.units.Registry`.
- 2013-06-26 Refactored the axes module: the axes module is now a folder, containing the following submodule:
- `_subplots.py`, containing all the subplots helper methods

- `_base.py`, containing several private methods and a new `_AxesBase` class. This `_AxesBase` class contains all the methods that are not directly linked to plots of the "old" Axes
- `_axes.py` contains the Axes class. This class now inherits from `_AxesBase`: it contains all "plotting" methods and labelling methods.

This refactoring should not affect the API. Only private methods are not importable from the axes module anymore.

2013-05-18 Added support for arbitrary rasterization resolutions to the SVG backend. Previously the resolution was hard coded to 72 dpi. Now the backend class takes a `image_dpi` argument for its constructor, adjusts the image bounding box accordingly and forwards a magnification factor to the image renderer. The code and results now resemble those of the PDF backend. - MW

2013-05-08 Changed behavior of `hist` when given `stacked=True` and `normed=True`. Histograms are now stacked first, then the sum is normalized. Previously, each histogram was normalized, then they were stacked.

2013-04-25 Changed all instances of:

`from matplotlib import MatplotlibDeprecationWarning as mplDeprecation` to:

`from cbook import mplDeprecation`

and removed the import into the matplotlib namespace in `__init__.py` Thomas Caswell

2013-04-15 Added `'axes.xmargin'` and `'axes.ymargin'` to `rcParams` to set default margins on auto-scaling. - TAC

2013-04-16 Added `patheffect` support for Line2D objects. -JJL

2013-03-31 Added support for arbitrary unstructured user-specified triangulations to `Axes3D.tricontour[f]` - Damon McDougall

2013-03-19 Added support for passing `linestyle` kwarg to `step` so all plot kwargs are passed to the underlying plot call. -TAC

2013-02-25 Added classes `CubicTriInterpolator`, `UniformTriRefiner`, `TriAnalyzer` to `matplotlib.tri` module. - GBy

2013-01-23 Add `'savefig.directory'` to `rcParams` to remember and fill in the last directory saved to for figure save dialogs - Martin Spacek

2013-01-13 Add `eventplot` method to axes and `pyplot` and `EventCollection` class to collections.

2013-01-08 Added two extra titles to axes which are flush with the left and right edges of the plot respectively. Andrew Dawson

2013-01-07 Add `framealpha` keyword argument to `legend` - PO

2013-01-16 Till Stensitzki added a baseline feature to `stackplot`

2012-12-22 Added classes for interpolation within triangular grids (`LinearTriInterpolator`) and to find the triangles in which points lie (`TrapezoidMapTriFinder`) to `matplotlib.tri` module. - IMT

2012-12-05 Added MatplotlibDeprecationWarning class for signaling deprecation. Matplotlib developers can use this class as follows:

```
from matplotlib import MatplotlibDeprecationWarning as mplDeprecation
```

In light of the fact that Python builtin DeprecationWarnings are ignored by default as of Python 2.7, this class was put in to allow for the signaling of deprecation, but via UserWarnings which are not ignored by default. - PI

2012-11-27 Added the mtext parameter for supplying matplotlib.text.Text instances to RendererBase.draw_text and RendererBase.draw_text. This allows backends to utilize additional text attributes, like the alignment of text elements. - pworth

2012-11-26 deprecate matplotlib/mpl.py, which was used only in pylab.py and is now replaced by the more suitable import matplotlib as mpl. - PI

2012-11-25 Make rc_context available via pyplot interface - PI

2012-11-16 plt.set_cm no longer throws errors if there is not already an active colorable artist, such as an image, and just sets up the colormap to use from that point forward. - PI

2012-11-16 Added the function _get_rgba_face, which is identical to _get_rgb_face except it return a (r,g,b,a) tuple, to line2D. Modified Line2D.draw to use _get_rgba_face to get the markerface color so that any alpha set by markerfacecolor will be respected. - Thomas Caswell

2012-11-13 Add a symmetric log normalization class to colors.py. Also added some tests for the normalization class. Till Stensitzki

2012-11-12 Make axes.stem take at least one argument. Uses a default range(n) when the first arg not provided. Damon McDougall

2012-11-09 Make plt.subplot() without arguments act as subplot(111) - PI

2012-11-08 Replaced plt.figure and plt.subplot calls by the newer, more convenient single call to plt.subplots() in the documentation examples - PI

2012-10-05 Add support for saving animations as animated GIFs. - JVDP

2012-08-11 Fix path-closing bug in patches.Polygon, so that regardless of whether the path is the initial one or was subsequently set by set_xy(), get_xy() will return a closed path if and only if get_closed() is True. Thanks to Jacob Vanderplas. - EF

2012-08-05 When a norm is passed to contourf, either or both of the vmin, vmax attributes of that norm are now respected. Formerly they were respected only if both were specified. In addition, vmin and/or vmax can now be passed to contourf directly as kwargs. - EF

2012-07-24 Contourf handles the extend kwarg by mapping the extended ranges outside the normed 0-1 range so that they are handled by colormap colors determined by the set_under and set_over methods. Previously the extended ranges were mapped to 0 or 1 so that the "under" and "over" colormap colors were ignored. This change also increases slightly the color contrast for a given set of contour levels. - EF

2012-06-24 Make use of mathtext in tick labels configurable - DSD

2012-06-05 Images loaded through PIL are now ordered correctly - CG

2012-06-02 Add new Axes method and pyplot function, hist2d. - PO

- 2012-05-31 Remove support for 'cairo.<format>' style of backend specification. Deprecate 'cairo.format' and 'savefig.extension' rcParams and replace with 'savefig.format'. - Martin Spacek
- 2012-05-29 pcolormesh now obeys the passed in "edgecolor" kwarg. To support this, the "shading" argument to pcolormesh now only takes "flat" or "gouraud". To achieve the old "faceted" behavior, pass "edgecolors='k'". - MGD
- 2012-05-22 Added radius kwarg to pie charts. - HH
- 2012-05-22 Collections now have a setting "offset_position" to select whether the offsets are given in "screen" coordinates (default, following the old behavior) or "data" coordinates. This is currently used internally to improve the performance of hexbin.

As a result, the "draw_path_collection" backend methods have grown a new argument "offset_position". - MGD
- 2012-05-04 Add a new argument to pie charts - startingangle - that allows one to specify the angle offset for the first wedge of the chart. - EP
- 2012-05-03 symlog scale now obeys the logarithmic base. Previously, it was completely ignored and always treated as base e. - MGD
- 2012-05-03 Allow linscalex/y keyword to symlog scale that allows the size of the linear portion relative to the logarithmic portion to be adjusted. - MGD
- 2012-04-14 Added new plot style: stackplot. This new feature supports stacked area plots. - Damon McDougall
- 2012-04-06 When path clipping changes a LINETO to a MOVETO, it also changes any CLOSEPOLY command to a LINETO to the initial point. This fixes a problem with pdf and svg where the CLOSEPOLY would then draw a line to the latest MOVETO position instead of the intended initial position. - JKS
- 2012-03-27 Add support to ImageGrid for placing colorbars only at one edge of each column/row. - RMM
- 2012-03-07 Refactor movie writing into useful classes that make use of pipes to write image data to ffmpeg or mencoder. Also improve settings for these and the ability to pass custom options. - RMM
- 2012-02-29 errorevery keyword added to errorbar to enable errorbar subsampling. fixes issue #600.
- 2012-02-28 Added plot_trisurf to the mplot3d toolkit. This supports plotting three dimensional surfaces on an irregular grid. - Damon McDougall
- 2012-01-23 The radius labels in polar plots no longer use a fixed padding, but use a different alignment depending on the quadrant they are in. This fixes numerical problems when (rmax - rmin) gets too small. - MGD
- 2012-01-08 Add axes.streamplot to plot streamlines of a velocity field. Adapted from Tom Flanagan streamplot implementation. -TSY
- 2011-12-29 ps and pdf markers are now stroked only if the line width is nonzero for consistency with agg, fixes issue #621. - JKS

- 2011-12-27 Work around an EINTR bug in some versions of subprocess. - JKS
- 2011-10-25 added support for `operatorname` to `mathtext`, including the ability to insert spaces, such as `$operatorname{arg,max}$` - PI
- 2011-08-18 Change api of `Axes.get_tightbbox` and add an optional `keyword` parameter `call_axes_locator`. - JJL
- 2011-07-29 A new rcParam `"axes.formatter.use_locale"` was added, that, when `True`, will use the current locale to format tick labels. This means that, for example, in the `fr_FR` locale, `'` will be used as a decimal separator. - MGD
- 2011-07-15 The set of markers available in the `plot()` and `scatter()` commands has been unified. In general, this gives more options to both than were previously available, however, there is one backward-incompatible change to the markers in `scatter`:
- `"d"` used to mean "diamond", it now means "narrow diamond". `"D"` can be used for a "diamond".
- MGD
- 2011-07-13 Fix numerical problems in symlog scale, particularly when `linthresh <= 1.0`. Symlog plots may look different if one was depending on the old broken behavior - MGD
- 2011-07-10 Fixed argument handling error in `tripcolor/triplot/tricontour`, issue #203. - IMT
- 2011-07-08 Many functions added to `mplot3d.axes3d` to bring `Axes3D` objects more feature-parity with regular `Axes` objects. Significant revisions to the documentation as well. - BVR
- 2011-07-07 Added compatibility with IPython strategy for picking a version of Qt4 support, and an rcParam for making the choice explicitly: `backend.qt4`. - EF
- 2011-07-07 Modified `AutoMinorLocator` to improve automatic choice of the number of minor intervals per major interval, and to allow one to specify this number via a kwarg. - EF
- 2011-06-28 3D versions of `scatter`, `plot`, `plot_wireframe`, `plot_surface`, `bar3d`, and some other functions now support empty inputs. - BVR
- 2011-06-22 Add `set_theta_offset`, `set_theta_direction` and `set_theta_zero_location` to polar axes to control the location of 0 and directionality of theta. - MGD
- 2011-06-22 Add `axes.labelweight` parameter to set font weight to axis labels - MGD.
- 2011-06-20 Add pause function to `pyplot`. - EF
- 2011-06-16 Added bottom keyword parameter for the stem command. Also, implemented a legend handler for the stem plot. - JJL
- 2011-06-16 Added `legend.frameon` rcParams. - Mike Kaufman
- 2011-05-31 Made `backend_qt4` compatible with PySide. - Gerald Storer
- 2011-04-17 Disable keyboard auto-repeat in qt4 backend by ignoring key events resulting from auto-repeat. This makes constrained zoom/pan work. - EF
- 2011-04-14 `interpolation="nearest"` always interpolate images. A new mode `"none"` is introduced for no interpolation - JJL

- 2011-04-03 Fixed broken pick interface to AsteriskCollection objects used by scatter. - EF
- 2011-04-01 The plot directive Sphinx extension now supports all of the features in the Numpy fork of that extension. These include doctest formatting, an 'include-source' option, and a number of new configuration options. - MGD
- 2011-03-29 Wrapped ViewVCCachedServer definition in a factory function. This class now inherits from urllib2.HTTPSHandler in order to fetch data from github, but HTTPSHandler is not defined if python was built without SSL support. - DSD
- 2011-03-10 Update pytz version to 2011c, thanks to Simon Cross. - JKS
- 2011-03-06 Add standalone tests.py test runner script. - JKS
- 2011-03-06 Set edgecolor to 'face' for scatter asterisk-type symbols; this fixes a bug in which these symbols were not responding to the c kwarg. The symbols have no face area, so only the edgecolor is visible. - EF
- 2011-02-27 Support libpng version 1.5.x; suggestion by Michael Albert. Changed installation specification to a minimum of libpng version 1.2. - EF
- 2011-02-20 clabel accepts a callable as an fmt kwarg; modified patch by Daniel Hyams. - EF
- 2011-02-18 scatter([], []) is now valid. Also fixed issues with empty collections - BVR
- 2011-02-07 Quick workaround for dviread bug #3175113 - JKS
- 2011-02-05 Add cbook memory monitoring for Windows, using tasklist. - EF
- 2011-02-05 Speed up Normalize and LogNorm by using in-place operations and by using float32 for float32 inputs and for ints of 2 bytes or shorter; based on patch by Christoph Gohlke. - EF
- 2011-02-04 Changed imshow to use rgba as uint8 from start to finish, instead of going through an intermediate step as double precision; thanks to Christoph Gohlke. - EF
- 2011-01-13 Added zdir and offset arguments to contourf3d to bring contourf3d in feature parity with contour3d. - BVR
- 2011-01-04 Tag 1.0.1 for release at r8896
- 2011-01-03 Added display of ticker offset to 3d plots. - BVR
- 2011-01-03 Turn off tick labeling on interior subplots for pyplots.subplots when sharex/sharey is True. - JDH
- 2010-12-29 Implement axes_divider.HBox and VBox. -JJL
- 2010-11-22 Fixed error with Hammer projection. - BVR
- 2010-11-12 Fixed the placement and angle of axis labels in 3D plots. - BVR
- 2010-11-07 New rc parameters examples.download and examples.directory allow bypassing the download mechanism in get_sample_data. - JKS
- 2010-10-04 Fix JPEG saving bug: only accept the kwargs documented by PIL for JPEG files. - JKS

- 2010-09-15 Remove unused `_wxagg` extension and `numerix.h`. - EF
- 2010-08-25 Add new framework for doing animations with examples.- RM
- 2010-08-21 Remove unused and inappropriate methods from Tick classes: `set_view_interval`, `get_minpos`, and `get_data_interval` are properly found in the Axis class and don't need to be duplicated in XTick and YTick. - EF
- 2010-08-21 Change `Axis.set_view_interval()` so that when updating an existing interval, it respects the orientation of that interval, and can enlarge but not reduce the interval. This fixes a bug in which `Axis.set_ticks` would change the view limits of an inverted axis. Whether `set_ticks` should be affecting the `viewLim` at all remains an open question. - EF
- 2010-08-16 Handle NaN's correctly in path analysis routines. Fixes a bug where the best location for a legend was not calculated correctly when the line contains NaNs. - MGD
- 2010-08-14 Fix bug in patch alpha handling, and in bar color kwarg - EF
- 2010-08-12 Removed all traces of `numerix` module after 17 months of deprecation warnings. - EF
- 2010-08-05 Added keyword arguments `'thetaunits'` and `'runits'` for polar plots. Fixed `PolarAxes` so that when it set default Formatters, it marked them as such. Fixed `semilogx` and `semilogy` to no longer blindly reset the ticker information on the non-log axis. `Axes.arrow` can now accept unitized data. - JRE
- 2010-08-03 Add support for `MPLSETUPCFG` variable for custom `setup.cfg` filename. Used by sage buildbot to build an mpl w/ no gui support - JDH
- 2010-08-01 Create directory specified by `MPLCONFIGDIR` if it does not exist. - ADS
- 2010-07-20 Return Qt4's default cursor when leaving the canvas - DSD
- 2010-07-06 Tagging for mpl 1.0 at r8502
- 2010-07-05 Added Ben Root's patch to put 3D plots in arbitrary axes, allowing you to mix 3d and 2d in different axes/subplots or to have multiple 3D plots in one figure. See `examples/mplot3d/subplot3d_demo.py` - JDH
- 2010-07-05 Preferred kwarg names in `set_xlim` are now `'left'` and `'right'`; in `set_ylim`, `'bottom'` and `'top'`; original kwargs are still accepted without complaint. - EF
- 2010-07-05 TkAgg and FtkAgg backends are now consistent with other interactive backends: when used in scripts from the command line (not from `ipython -pylab`), show blocks, and can be called more than once. - EF
- 2010-07-02 Modified `CXX/WrapPython.h` to fix "swab bug" on solaris so mpl can compile on Solaris with CXX6 in the trunk. Closes tracker bug 3022815 - JDH
- 2010-06-30 Added `autoscale` convenience method and corresponding pyplot function for simplified control of autoscaling; and changed `axis`, `set_xlim`, and `set_ylim` so that by default, they turn off the autoscaling on the relevant axis or axes. Therefore one can call `set_xlim` before plotting a line, for example, and the limits will be retained. - EF
- 2010-06-20 Added `Axes.tick_params` and corresponding pyplot function to control tick and tick label appearance after an Axes has been created. - EF

- 2010-06-09 Allow Axes.grid to control minor gridlines; allow Axes.grid and Axis.grid to control major and minor gridlines in the same method call. - EF
- 2010-06-06 Change the way we do split/dividend adjustments in finance.py to handle dividends and fix the zero division bug reported in sf bug 2949906 and 2123566. Note that volume is not adjusted because the Yahoo CSV does not distinguish between share split and dividend adjustments making it near impossible to get volume adjustment right (unless we want to guess based on the size of the adjustment or scrape the html tables, which we don't) - JDH
- 2010-06-06 Updated dateutil to 1.5 and pytz to 2010h.
- 2010-06-02 Add error_kw kwarg to Axes.bar(). - EF
- 2010-06-01 Fix pcolormesh() and QuadMesh to pass on kwargs as appropriate. - RM
- 2010-05-18 Merge mpl_toolkits.gridspec into the main tree. - JJJ
- 2010-05-04 Improve backend_qt4 so it displays figures with the correct size - DSD
- 2010-04-20 Added generic support for connecting to a timer for events. This adds TimerBase, TimerGTK, TimerQT, TimerWx, and TimerTk to the backends and a new_timer() method to each backend's canvas to allow ease of creating a new timer. - RM
- 2010-04-20 Added margins() Axes method and pyplot function. - EF
- 2010-04-18 update the axes_grid documentation. -JJJ
- 2010-04-18 Control MaxNLocator parameters after instantiation, and via Axes.locator_params method, with corresponding pyplot function. -EF
- 2010-04-18 Control ScalarFormatter offsets directly and via the Axes.ticklabel_format() method, and add that to pyplot. -EF
- 2010-04-16 Add a close_event to the backends. -RM
- 2010-04-06 modify axes_grid examples to use axes_grid1 and axisartist. -JJJ
- 2010-04-06 rebase axes_grid using axes_grid1 and axisartist modules. -JJJ
- 2010-04-06 axes_grid toolkit is splitted into two separate modules, axes_grid1 and axisartist. - JJJ
- 2010-04-05 Speed up import: import pytz only if and when it is needed. It is not needed if the rc timezone is UTC. - EF
- 2010-04-03 Added color kwarg to Axes.hist(), based on work by Jeff Klukas. - EF
- 2010-03-24 refactor colorbar code so that no cla() is necessary when mappable is changed. -JJJ
- 2010-03-22 fix incorrect rubber band during the zoom mode when mouse leaves the axes. -JJJ
- 2010-03-21 x/y key during the zoom mode only changes the x/y limits. -JJJ
- 2010-03-20 Added pyplot.sca() function suggested by JJJ. - EF
- 2010-03-20 Added conditional support for new Tooltip API in gtk backend. - EF

2010-03-20 Changed `plt.fig_subplot()` to `plt.subplots()` after discussion on `list`, and changed its API to return axes as a numpy object array (with control of dimensions via `squeeze` keyword). FP.

2010-03-13 Manually brought in commits from branch:

```
-----
r8191 | leejjoon | 2010-03-13 17:27:57 -0500 (Sat, 13 Mar 2010) | 1 line

fix the bug that handles for scatter are incorrectly set when dpi!=72.
Thanks to Ray Speth for the bug report.
```

2010-03-03 Manually brought in commits from branch via `diff/patch` (`svnmerge` is broken):

```
-----
r8175 | leejjoon | 2010-03-03 10:03:30 -0800 (Wed, 03 Mar 2010) | 1 line

fix arguments of allow_rasterization.draw_wrapper

-----
r8174 | jdh2358 | 2010-03-03 09:15:58 -0800 (Wed, 03 Mar 2010) | 1 line

added support for favicon in docs build

-----
r8173 | jdh2358 | 2010-03-03 08:56:16 -0800 (Wed, 03 Mar 2010) | 1 line

applied Mattias get_bounds patch

-----
r8172 | jdh2358 | 2010-03-03 08:31:42 -0800 (Wed, 03 Mar 2010) | 1 line

fix svnmerge download instructions

-----
r8171 | jdh2358 | 2010-03-03 07:47:48 -0800 (Wed, 03 Mar 2010) | 1 line
```

2010-02-25 add `annotation_demo3.py` that demonstrates new functionality. -JJL

2010-02-25 refactor `Annotation` to support arbitrary `Transform` as `xycoords` or `textcoords`. Also, if a tuple of two coordinates is provided, they are interpreted as coordinates for each `x` and `y` position. -JJL

2010-02-24 Added `pyplot.fig_subplot()`, to create a figure and a group of subplots in a single call. This offers an easier pattern than manually making figures and calling `add_subplot()` multiple times. FP

2010-02-17 Added Gokhan's and Mattias' customizable keybindings patch for the toolbar. You can now set the `keymap.*` properties in the `matplotlibrc` file. Newbindings were added for toggling log scaling on the x-axis. JDH

2010-02-16 Committed TJ's filled marker patch for `left|right|bottom|top|full` filled markers. See `examples/pylab_examples/filledmarker_demo.py`. JDH

2010-02-11 Added `'bootstrap'` option to `boxplot`. This allows bootstrap estimates of median con-

- fidence intervals. Based on an initial patch by Paul Hobson. - ADS
- 2010-02-06 Added setup.cfg "basedirlist" option to override setting in setupext.py "basedir" dictionary; added "gnu0" platform requested by Benjamin Drung. - EF
- 2010-02-06 Added 'xy' scaling option to EllipseCollection. - EF
- 2010-02-03 Made plot_directive use a custom PlotWarning category, so that warnings can be turned into fatal errors easily if desired. - FP
- 2010-01-29 Added draggable method to Legend to allow mouse drag placement. Thanks Adam Fraser. JDH
- 2010-01-25 Fixed a bug reported by Olle Engdegard, when using histograms with stepfilled and log=True - MM
- 2010-01-16 Upgraded CXX to 6.1.1 - JDH
- 2009-01-16 Don't create minor ticks on top of existing major ticks. Patch by Neil Crighton. - ADS
- 2009-01-16 Ensure three minor ticks always drawn (SF# 2924245). Patch by Neil Crighton. - ADS
- 2010-01-16 Applied patch by Ian Thomas to fix two contouring problems: now contourf handles interior masked regions, and the boundaries of line and filled contours coincide. - EF
- 2009-01-11 The color of legend patch follows the rc parameters axes.facecolor and axes.edgecolor. -JLL
- 2009-01-11 adjustable of Axes can be "box-forced" which allow sharing axes. -JLL
- 2009-01-11 Add add_click and pop_click methods in BlockingContourLabeler. -JLL
- 2010-01-03 Added rcParams['axes.color_cycle'] - EF
- 2010-01-03 Added Pierre's qt4 formlayout editor and toolbar button - JDH
- 2009-12-31 Add support for using math text as marker symbols (Thanks to tcb)
- MGD
- 2009-12-31 Commit a workaround for a regression in PyQt4-4.6.{0,1} - DSD
- 2009-12-22 Fix cmap data for gist_earth_r, etc. -JLL
- 2009-12-20 spines: put spines in data coordinates, add set_bounds() call. -ADS
- 2009-12-18 Don't limit notch size in boxplot to q1-q3 range, as this is effectively making the data look better than it is. - ADS
- 2009-12-18 mlab.prctile handles even-length data, such that the median is the mean of the two middle values. - ADS
- 2009-12-15 Add raw-image (unsampled) support for the ps backend. - JLL
- 2009-12-14 Add patch_artist kwarg to boxplot, but keep old default. Convert boxplot_demo2.py to use the new patch_artist. - ADS

2009-12-06 `axes_grid`: reimplemented `AxisArtist` with `FloatingAxes` support. Added new examples. - JJL

2009-12-01 Applied Laurent Dufrechou's patch to improve blitting with the qt4 backend - DSD

2009-11-13 The pdf backend now allows changing the contents of a pdf file's information dictionary via `PdfPages.infodict`. - JKS

2009-11-12 `font_manager.py` should no longer cause EINTR on Python 2.6 (but will on the 2.5 version of subprocess). Also the `fc-list` command in that file was fixed so now it should actually find the list of fontconfig fonts. - JKS

2009-11-10 Single images, and all images in renderers with `option_image_nocomposite` (i.e. agg, macosx and the svg backend when `rcParams['svg.image_noscale']` is True), are now drawn respecting the zorder relative to other artists. (Note that there may now be inconsistencies across backends when more than one image is drawn at varying zorders, but this change introduces correct behavior for the backends in which it's easy to do so.)

2009-10-21 Make `AutoDateLocator` more configurable by adding options to control the maximum and minimum number of ticks. Also add control of the intervals to be used for ticking. This does not change behavior but opens previously hard-coded behavior to runtime modification'. - RMM

2009-10-19 Add "path_effects" support for Text and Patch. See `examples/pylab_examples/patheffect_demo.py` -JJL

2009-10-19 Add "use_clabeltext" option to `clabel`. If True, clabels will be created with `ClabelText` class, which recalculates rotation angle of the label during the drawing time. -JJL

2009-10-16 Make `AutoDateFormatter` actually use any specified `timezone` setting. This was only working correctly when no `timezone` was specified. - RMM

2009-09-27 Beginnings of a capability to test the pdf backend. - JKS

2009-09-27 Add a `savefig.extension` rcparam to control the default `filename` extension used by `savefig`. - JKS

2009-09-21 Tagged for release 0.99.1

2009-09-20 Fix `usetex` spacing errors in pdf backend. - JKS

2009-09-20 Add Sphinx extension to highlight IPython console sessions, originally authored (I think) by Michael Droetboom. - FP

2009-09-20 Fix off-by-one error in `dviread.Tfm`, and additionally protect against exceptions in case a dvi font is missing some metrics. - JKS

2009-09-15 Implement `draw_text` and `draw_tex` method of `backend_base` using the `textpath` module. Implement `draw_tex` method of the svg backend. - JJL

2009-09-15 Don't fail on AFM files containing floating-point bounding boxes - JKS

2009-09-13 `AxesGrid` [add modified version of `colorbar`. Add `colorbar` location howto. - JJL

- 2009-09-07 AxesGrid [implemented axisline style.] Added a demo examples/axes_grid/demo_axisline_style.py- JJL
- 2009-09-04 Make the textpath class as a separate module (textpath.py). Add support for math-text and tex.- JJL
- 2009-09-01 Added support for Gouraud interpolated triangles. pcolormesh now accepts shading='gouraud' as an option. - MGD
- 2009-08-29 Added matplotlib.testing package, which contains a Nose plugin and a decorator that lets tests be marked as KnownFailures - ADS
- 2009-08-20 Added scaled dict to AutoDateFormatter for customized scales - JDH
- 2009-08-15 Pyplot interface: the current image is now tracked at the figure and axes level, addressing tracker item 1656374. - EF
- 2009-08-15 Docstrings are now manipulated with decorators defined in a new module, docstring.py, thanks to Jason Coombs. - EF
- 2009-08-14 Add support for image filtering for agg back end. See the example demo_agg_filter.py. -JJL
- 2009-08-09 AnnotationBbox added. Similar to Annotation, but works with OffsetBox instead of Text. See the example demo_annotation_box.py. -JJL
- 2009-08-07 BboxImage implemented. Two examples, demo_bboximage.py and demo_ribbon_box.py added. - JJL
- 2009-08-07 In an effort to simplify the backend API, all clipping rectangles and paths are now passed in using GraphicsContext objects, even on collections and images. Therefore:
- ```
draw_path_collection(self, master_transform, cliprect, clippath,
 clippath_trans, paths, all_transforms, offsets, offsetTrans, facecolors,
 edgecolors, linewidths, linestyle, antialiaseds, urls)
```
- becomes:
- ```
draw_path_collection(self, gc, master_transform, paths, all_transforms, offsets,
                    offsetTrans, facecolors, edgecolors, linewidths, linestyle, antialiaseds, urls)
```
- ```
draw_quad_mesh(self, master_transform, cliprect, clippath, clippath_trans,
 meshWidth, meshHeight, coordinates, offsets, offsetTrans, facecolors, antialiased, showedges)
```
- becomes:
- ```
draw_quad_mesh(self, gc, master_transform, meshWidth, meshHeight,
                coordinates, offsets, offsetTrans, facecolors, antialiased, showedges)
```
- ```
draw_image(self, x, y, im, bbox, clippath=None, clippath_trans=None)
```
- becomes:
- ```
draw_image(self, gc, x, y, im)
```
- MGD

2009-08-06 Tagging the 0.99.0 release at svn r7397 - JDH

- fixed an alpha colormapping bug posted on sf 2832575
- fix typo in axes_divider.py. use nanmin, nanmax in angle_helper.py (patch by Christoph Gohlke)
- remove dup gui event in enter/leave events in gtk
- lots of fixes for os x binaries (Thanks Russell Owen)
- attach gtk events to mpl events – fixes sf bug 2816580
- applied sf patch 2815064 (middle button events for wx) and patch 2818092 (resize events for wx)
- fixed boilerplate.py so it doesn't break the ReST docs.
- removed a couple of cases of mlab.load
- fixed rec2csv win32 file handle bug from sf patch 2831018
- added two examples from Josh Hemann: examples/pylab_examples/barchart_demo2.py and examples/pylab_examples/boxplot_demo2.py
- handled sf bugs 2831556 and 2830525; better bar error messages and backend driver configs
- added miktex win32 patch from sf patch 2820194
- apply sf patches 2830233 and 2823885 for osx setup and 64 bit; thanks Michiel

2009-08-04 Made cbook.get_sample_data make use of the ETag and Last-Modified headers of mod_dav_svn. - JKS

2009-08-03 Add PathCollection; modify contourf to use complex paths instead of simple paths with cuts. - EF

2009-08-03 Fixed boilerplate.py so it doesn't break the ReST docs. - JKS

2009-08-03 pylab no longer provides a load and save function. These are available in matplotlib.mlab, or you can use numpy.loadtxt and numpy.savetxt for text files, or np.save and np.load for binary numpy arrays. - JDH

2009-07-31 Added cbook.get_sample_data for urllib enabled fetching and cacheing of data needed for examples. See examples/misc/sample_data_demo.py - JDH

2009-07-31 Tagging 0.99.0.rc1 at 7314 - MGD

2009-07-30 Add set_cmap and register_cmap, and improve get_cmap, to provide convenient handling of user-generated colormaps. Reorganized _cm and cm modules. - EF

2009-07-28 Quiver speed improved, thanks to tip by Ray Speth. -EF

2009-07-27 Simplify argument handling code for plot method. -EF

2009-07-25 Allow "plot(1, 2, 'r*')" to work. - EF

2009-07-22 Added an 'interp' keyword to griddata so the faster linear interpolation method can be chosen. Default is 'nn', so default behavior (using natural neighbor method) is unchanged (JSW)

- 2009-07-22 Improved boilerplate.py so that it generates the correct signatures for pyplot functions. - JKS
- 2009-07-19 Fixed the docstring of Axes.step to reflect the correct meaning of the kwargs "pre" and "post" - See SF bug https://sourceforge.net/tracker/index.php?func=detail&aid=2823304&group_id=80706&atid=560720 - JDH
- 2009-07-18 Fix support for hatches without color fills to pdf and svg backends. Add an example of that to hatch_demo.py. - JKS
- 2009-07-17 Removed fossils from swig version of agg backend. - EF
- 2009-07-14 initial submission of the annotation guide. -JJL
- 2009-07-14 axes_grid [minor improvements in anchored_artists and] inset_locator. -JJL
- 2009-07-14 Fix a few bugs in ConnectionStyle algorithms. Add ConnectionPatch class. -JJL
- 2009-07-11 Added a fillstyle Line2D property for half filled markers - see examples/pylab_examples/fillstyle_demo.py JDH
- 2009-07-08 Attempt to improve performance of qt4 backend, do not call QApplication.processEvents while processing an event. Thanks Ole Streicher for tracking this down - DSD
- 2009-06-24 Add withheader option to mlab.rec2csv and changed use_mrecords default to False in mlab.csv2rec since this is partially broken - JDH
- 2009-06-24 backend_agg.draw_marker quantizes the main path (as in the draw_path). - JJL
- 2009-06-24 axes_grid: floating axis support added. - JJL
- 2009-06-14 Add new command line options to backend_driver.py to support running only some directories of tests - JKS
- 2009-06-13 partial cleanup of mlab and its importation in pylab - EF
- 2009-06-13 Introduce a rotation_mode property for the Text artist. See examples/pylab_examples/demo_text_rotation_mode.py -JJL
- 2009-06-07 add support for bz2 files per sf support request 2794556 - JDH
- 2009-06-06 added a properties method to the artist and inspector to return a dict mapping property name -> value; see sf feature request 2792183 - JDH
- 2009-06-06 added Neil's auto minor tick patch; sf patch #2789713 - JDH
- 2009-06-06 do not apply alpha to rgba color conversion if input is already rgba - JDH
- 2009-06-03 axes_grid [Initial check-in of curvilinear grid support. See] examples/axes_grid/demo_curvilinear_grid.py - JJL
- 2009-06-01 Add set_color method to Patch - EF
- 2009-06-01 Spine is now derived from Patch - ADS
- 2009-06-01 use cbook.is_string_like() instead of isinstance() for spines - ADS
- 2009-06-01 cla() support for spines - ADS
- 2009-06-01 Removed support for gtk < 2.4. - EF

- 2009-05-29 Improved the `animation_blit_qt4` example, which was a mix of the object-oriented and pylab interfaces. It is now strictly object-oriented - DSD
- 2009-05-28 Fix `axes_grid` toolkit to work with spine patch by ADS. - JJL
- 2009-05-28 Applied fbianco's patch to handle scroll wheel events in the qt4 backend - DSD
- 2009-05-26 Add support for "axis spines" to have arbitrary location. -ADS
- 2009-05-20 Add an empty `matplotlibrc` to the `tests/` directory so that running tests will use the default set of `rcparams` rather than the user's config. - RMM
- 2009-05-19 `Axis.grid()`: allow use of `which='major,minor'` to have grid on major and minor ticks. -ADS
- 2009-05-18 Make `psd()`, `csd()`, and `cohere()` wrap properly for complex/two-sided versions, like `specgram()` (SF #2791686) - RMM
- 2009-05-18 Fix the linespacing bug of multiline text (#1239682). See `examples/pylab_examples/multiline.py` -JJL
- 2009-05-18 Add `annotation_clip` attr. for `text.Annotation` class. If `True`, annotation is only drawn when the annotated point is inside the axes area. -JJL
- 2009-05-17 Fix bug(#2749174) that some properties of minor ticks are not conserved -JJL
- 2009-05-17 applied Michiel's sf patch 2790638 to turn off gtk event loop in `setuext` for `pygtk>=2.15.10` - JDH
- 2009-05-17 applied Michiel's sf patch 2792742 to speed up Cairo and `macosx` collections; speedups can be 20x. Also fixes some bugs in which gc got into inconsistent state
-
- 2008-05-17 Release 0.98.5.3 at r7107 from the branch - JDH
- 2009-05-13 An optional offset and bbox support in `restore_bbox`. Add `animation_blit_gtk2.py`. -JJL
- 2009-05-13 `psfrag` in `backend_ps` now uses `baseline-alignment` when `preview.sty` is used ((default is `bottom-alignment`). Also, a small api improvement in `OffsetBox`-JJL
- 2009-05-13 When the x-coordinate of a line is monotonically increasing, it is now automatically clipped at the stage of generating the transformed path in the `draw` method; this greatly speeds up zooming and panning when one is looking at a short segment of a long time series, for example. - EF
- 2009-05-11 `aspect=1` in log-log plot gives square decades. -JJL
- 2009-05-08 `clabel` takes new kwarg, `rightside_up`; if `False`, labels will not be flipped to keep them rightside-up. This allows the use of `clabel` to make streamfunction arrows, as requested by Evan Mason. - EF
- 2009-05-07 'labelpad' can now be passed when setting x/y labels. This allows controlling the spacing between the label and its axis. - RMM
- 2009-05-06 `print_ps` now uses mixed-mode renderer. `Axes.draw` rasterize artists whose `zorder` smaller than `rasterization_zorder`. -JJL

- 2009-05-06 Per-artist Rasterization, originally by Eric Bruning. -JJ
- 2009-05-05 Add an example that shows how to make a plot that updates using data from another process. Thanks to Robert Cimrman - RMM
- 2009-05-05 Add Axes.get_legend_handles_labels method. - JJL
- 2009-05-04 Fix bug that Text.Annotation is still drawn while set to not visible. - JJL
- 2009-05-04 Added TJ's fill_betweenx patch - JDH
- 2009-05-02 Added options to plotfile based on question from Joseph Smidt and patch by Matthias Michler. - EF
- 2009-05-01 Changed add_artist and similar Axes methods to return their argument. - EF
- 2009-04-30 Incorrect eps bbox for landscape mode fixed - JJL
- 2009-04-28 Fixed incorrect bbox of eps output when usetex=True. - JJL
- 2009-04-24 Changed use of os.open* to instead use subprocess.Popen. os.popen* are deprecated in 2.6 and are removed in 3.0. - RMM
- 2009-04-20 Worked on axes_grid documentation. Added axes_grid.inset_locator. - JJL
- 2009-04-17 Initial check-in of the axes_grid toolkit. - JJL
- 2009-04-17 Added a support for bbox_to_anchor in offsetbox.AnchoredOffsetbox. Improved a documentation. - JJL
- 2009-04-16 Fixed a offsetbox bug that multiline texts are not correctly aligned. - JJL
- 2009-04-16 Fixed a bug in mixed mode renderer that images produced by an rasterizing backend are placed with incorrect size. - JJL
- 2009-04-14 Added Jonathan Taylor's Reinier Heeres' port of John Porters' mplot3d to svn trunk. Package in mpl_toolkits.mplot3d and demo is examples/mplot3d/demo.py. Thanks Reiner
- 2009-04-06 The pdf backend now escapes newlines and linefeeds in strings. Fixes sf bug #2708559; thanks to Tiago Pereira for the report.
- 2009-04-06 texmanager.make_dvi now raises an error if LaTeX failed to create an output file. Thanks to Joao Luis Silva for reporting this. - JKS
- 2009-04-05 _png.read_png() reads 12 bit PNGs (patch from Tobias Wood) - ADS
- 2009-04-04 Allow log axis scale to clip non-positive values to small positive value; this is useful for errorbars. - EF
- 2009-03-28 Make images handle nan in their array argument. A helper, cbook.safe_masked_invalid() was added. - EF
- 2009-03-25 Make contour and contourf handle nan in their Z argument. - EF
- 2009-03-20 Add AuxTransformBox in offsetbox.py to support some transformation. anchored_text.py example is enhanced and renamed (anchored_artists.py). - JJL
- 2009-03-20 Add "bar" connection style for annotation - JJL
- 2009-03-17 Fix bugs in edge color handling by contourf, found by Jae-Joon Lee. - EF

- 2009-03-14 Added 'LightSource' class to colors module for creating shaded relief maps. shading_example.py added to illustrate usage. - JSW
- 2009-03-11 Ensure wx version \geq 2.8; thanks to Sandro Tosi and Chris Barker. - EF
- 2009-03-10 Fix join style bug in pdf. - JKS
- 2009-03-07 Add pyplot access to figure number list - EF
- 2009-02-28 hashing of FontProperties accounts current rcParams - JJL
- 2009-02-28 Prevent double-rendering of shared axis in twinx, twiny - EF
- 2009-02-26 Add optional bbox_to_anchor argument for legend class - JJL
- 2009-02-26 Support image clipping in pdf backend. - JKS
- 2009-02-25 Improve tick location subset choice in FixedLocator. - EF
- 2009-02-24 Deprecate numerix, and strip out all but the numpy part of the code. - EF
- 2009-02-21 Improve scatter argument handling; add an early error message, allow inputs to have more than one dimension. - EF
- 2009-02-16 Move plot_directive.py to the installed source tree. Add support for inline code content - MGD
- 2009-02-16 Move mathmpl.py to the installed source tree so it is available to other projects. - MGD
- 2009-02-14 Added the legend title support - JJL
- 2009-02-10 Fixed a bug in backend_pdf so it doesn't break when the setting pdf.use14corefonts=True is used. Added test case in unit/test_pdf_use14corefonts.py. - NGR
- 2009-02-08 Added a new imsave function to image.py and exposed it in the pyplot interface - GR
- 2009-02-04 Some reorganization of the legend code. anchored_text.py added as an example. - JJL
- 2009-02-04 Add extent keyword arg to hexbin - ADS
- 2009-02-04 Fix bug in mathtext related to dots and ldots - MGD
- 2009-02-03 Change default joinstyle to round - MGD
- 2009-02-02 Reduce number of marker XObjects in pdf output - JKS
- 2009-02-02 Change default resolution on polar plot to 1 - MGD
- 2009-02-02 Avoid malloc errors in ttconv for fonts that don't have e.g., PostName (a version of Tahoma triggered this) - JKS
- 2009-01-30 Remove support for pyExceclerator in exceltools - use xlwt instead - JDH
- 2009-01-29 Document 'resolution' kwarg for polar plots. Support it when using pyplot.polar, not just Figure.add_axes. - MGD
- 2009-01-29 Rework the nan-handling/clipping/quantizing/simplification framework so each is an independent part of a pipeline. Expose the C++-implementation of all of this so it can

- be used from all Python backends. Add rcParam "path.simplify_threshold" to control the threshold of similarity below which vertices will be removed.
- 2009-01-26 Improved tight bbox option of the savefig. - JJL
- 2009-01-26 Make curves and NaNs play nice together - MGD
- 2009-01-21 Changed the defaults of acorr and xcorr to use usevlines=True, maxlags=10 and normed=True since these are the best defaults
- 2009-01-19 Fix bug in quiver argument handling. - EF
- 2009-01-19 Fix bug in backend_gtk: don't delete nonexistent toolbar. - EF
- 2009-01-16 Implement bbox_inches option for savefig. If bbox_inches is "tight", try to determine the tight bounding box. - JJL
- 2009-01-16 Fix bug in is_string_like so it doesn't raise an unnecessary exception. - EF
- 2009-01-16 Fix an infinite recursion in the unit registry when searching for a converter for a sequence of strings. Add a corresponding test. - RM
- 2009-01-16 Bugfix of C typedef of MPL_Int64 that was failing on Windows XP 64 bit, as reported by George Goussard on numpy mailing list. - ADS
- 2009-01-16 Added helper function LinearSegmentedColormap.from_list to facilitate building simple custom colormaps. See examples/pylab_examples/custom_cmap_fromlist.py - JDH
- 2009-01-16 Applied Michiel's patch for macosx backend to fix rounding bug. Closed sf bug 2508440 - JSW
- 2009-01-10 Applied Michiel's hatch patch for macosx backend and draw_idle patch for qt. Closes sf patched 2497785 and 2468809 - JDH
- 2009-01-10 Fix bug in pan/zoom with log coordinates. - EF
- 2009-01-06 Fix bug in setting of dashed negative contours. - EF
- 2009-01-06 Be fault tolerant when len(linestyles)>Nlev in contour. - MM
- 2009-01-06 Added marginals kwarg to hexbin to plot marginal densities - JDH
- 2009-01-06 Change user-visible multipage pdf object to PdfPages to avoid accidents with the file-like PdfFile. - JKS
- 2009-01-05 Fix a bug in pdf usetex: allow using non-embedded fonts. - JKS
- 2009-01-05 optional use of preview.sty in usetex mode. - JJL
- 2009-01-02 Allow multipage pdf files. - JKS
- 2008-12-31 Improve pdf usetex by adding support for font effects (slanting and extending). - JKS
- 2008-12-29 Fix a bug in pdf usetex support, which occurred if the same Type-1 font was used with different encodings, e.g., with Minion Pro and MnSymbol. - JKS
- 2008-12-20 fix the dpi-dependent offset of Shadow. - JJL
- 2008-12-20 fix the hatch bug in the pdf backend. minor update in docs and example - JJL

2008-12-19 Add `axes_locator` attribute in `Axes`. Two examples are added.

- JJL

2008-12-19 Update `Axes.legend` documentation. `/api/api_changes.rst` is also updated to describe changes in keyword parameters. Issue a warning if old keyword parameters are used. - JJL

2008-12-18 add new arrow style, a line + filled triangles. -JJL

2008-12-18 Re-Released 0.98.5.2 from `v0_98_5_maint` at r6679 Released 0.98.5.2 from `v0_98_5_maint` at r6667

2008-12-18 Removed `configobj`, experimental traits and `doc/mpl_data` link - JDH

2008-12-18 Fix bug where a line with `NULL` data limits prevents subsequent data limits from calculating correctly - MGD

2008-12-17 Major documentation generator changes - MGD

2008-12-17 Applied macosx backend patch with support for path collections, quadmesh, etc... - JDH

2008-12-17 fix dpi-dependent behavior of text bbox and arrow in `annotate` -JJL

2008-12-17 Add group id support in `artist`. Two examples which demonstrate `svg` filter are added. -JJL

2008-12-16 Another attempt to fix dpi-dependent behavior of `Legend`. -JJL

2008-12-16 Fixed dpi-dependent behavior of `Legend` and `fancybox` in `Text`.

2008-12-16 Added `markevery` property to `Line2D` to support subsampling of markers - JDH

2008-12-15 Removed `mpl_data` symlink in docs. On platforms that do not support symlinks, these become copies, and the font files are large, so the distro becomes unnecessarily bloated. Keeping the `mpl_examples` dir because relative links are harder for the plot directive and the `*.py` files are not so large. - JDH

2008-12-15 Fix `$` in non-math text with `usetex` off. Document differences between `usetex` on/off - MGD

2008-12-15 Fix anti-aliasing when auto-snapping - MGD

2008-12-15 Fix grid lines not moving correctly during pan and zoom - MGD

2008-12-12 Preparations to eliminate `maskedarray` `rcParams` key: its use will now generate a warning. Similarly, importing the obsolete `numerix.numpy` will generate a warning. - EF

2008-12-12 Added support for the `numpy.histogram()` `weights` parameter to the `axes.hist()` method. Docs taken from `numpy` - MM

2008-12-12 Fixed warning in `hist()` with `numpy 1.2` - MM

2008-12-12 Removed external packages: `configobj` and `enthought.traits` which are only required by the experimental traitled config and are somewhat out of date. If needed, install them independently, see:

<http://code.enthought.com/pages/traits.html>

and:

<http://www.voidspace.org.uk/python/configobj.html>

2008-12-12 Added support to assign labels to histograms of multiple data. - MM

2008-12-11 Released 0.98.5 at svn r6573

2008-12-11 Use subprocess.Popen instead of os.popen in dviread (Windows problem reported by Jorgen Stenarson) - JKS

2008-12-10 Added Michael's font_manager fix and Jae-Joon's figure/subplot fix. Bumped version number to 0.98.5 - JDH

2008-12-09 Released 0.98.4 at svn r6536

2008-12-08 Added mdehoon's native macosx backend from sf patch 2179017 - JDH

2008-12-08 Removed the prints in the set_*style commands. Return the list of pprinted strings instead - JDH

2008-12-08 Some of the changes Michael made to improve the output of the property tables in the rest docs broke or made difficult to use some of the interactive doc helpers, e.g., setp and getp. Having all the rest markup in the ipython shell also confused the docstrings. I added a new rc param docstring.harcopy, to format the docstrings differently for hardcopy and other use. Ther ArtistInspector could use a little refactoring now since there is duplication of effort between the rest out put and the non-rest output - JDH

2008-12-08 Updated spectral methods (psd, csd, etc.) to scale one-sided densities by a factor of 2 and, optionally, scale all densities by the sampling frequency. This gives better MatLab compatibility. -RM

2008-12-08 Fixed alignment of ticks in colorbars. -MGD

2008-12-07 drop the deprecated "new" keyword of np.histogram() for numpy 1.2 or later. -JJL

2008-12-06 Fixed a bug in svg backend that new_figure_manager() ignores keywords arguments such as figsize, etc. -JJL

2008-12-05 Fixed a bug that the handlelength of the new legend class set too short when num-points=1 -JJL

2008-12-04 Added support for data with units (e.g., dates) to Axes.fill_between. -RM

2008-12-04 Added fancybox keyword to legend. Also applied some changes for better look, including baseline adjustment of the multiline texts so that it is center aligned. -JJL

2008-12-02 The transmuter classes in the patches.py are reorganized as subclasses of the Style classes. A few more box and arrow styles are added. -JJL

2008-12-02 Fixed a bug in the new legend class that didn't allowed a tuple of coordinate vlaues as loc. -JJL

- 2008-12-02 Improve checks for external dependencies, using subprocess (instead of deprecated popen*) and distutils (for version checking) - DSD
- 2008-11-30 Reimplementation of the legend which supports baseline alignment, multi-column, and expand mode. - JJL
- 2008-12-01 Fixed histogram autoscaling bug when bins or range are given explicitly (fixes Debian bug 503148) - MM
- 2008-11-25 Added rcParam axes.unicode_minus which allows plain hyphen for minus when False - JDH
- 2008-11-25 Added scatterpoints support in Legend. patch by Erik Tollerud - JJL
- 2008-11-24 Fix crash in log ticking. - MGD
- 2008-11-20 Added static helper method BrokenHBarCollection.span_where and Axes/pyplot method fill_between. See examples/pylab/fill_between.py - JDH
- 2008-11-12 Add x_isdata and y_isdata attributes to Artist instances, and use them to determine whether either or both coordinates are used when updating dataLim. This is used to fix autoscaling problems that had been triggered by axhline, axhspan, axvline, axvspan. - EF
- 2008-11-11 Update the psd(), csd(), cohere(), and specgram() methods of Axes and the csd() cohere(), and specgram() functions in mlab to be in sync with the changes to psd(). In fact, under the hood, these all call the same core to do computations. - RM
- 2008-11-11 Add 'pad_to' and 'sides' parameters to mlab.psd() to allow controlling of zero padding and returning of negative frequency components, respectively. These are added in a way that does not change the API. - RM
- 2008-11-10 Fix handling of c kwarg by scatter; generalize is_string_like to accept numpy and numpy.ma string array scalars. - RM and EF
- 2008-11-09 Fix a possible EINTR problem in dviread, which might help when saving pdf files from the qt backend. - JKS
- 2008-11-05 Fix bug with zoom to rectangle and twin axes - MGD
- 2008-10-24 Added Jae Joon's fancy arrow, box and annotation enhancements - see examples/pylab_examples/annotation_demo2.py
- 2008-10-23 Autoscaling is now supported with shared axes - EF
- 2008-10-23 Fixed exception in dviread that happened with Minion - JKS
- 2008-10-21 set_xlim, ylim now return a copy of the viewlim array to avoid modify inplace surprises
- 2008-10-20 Added image thumbnail generating function matplotlib.image.thumbnail. See examples/misc/image_thumbnail.py - JDH
- 2008-10-20 Applied scatleg patch based on ideas and work by Erik Tollerud and Jae-Joon Lee. - MM
- 2008-10-11 Fixed bug in pdf backend: if you pass a file object for output instead of a filename, e.g., in a web app, we now flush the object at the end. - JKS

- 2008-10-08 Add path simplification support to paths with gaps. - EF
- 2008-10-05 Fix problem with AFM files that don't specify the font's full name or family name. - JKS
- 2008-10-04 Added 'scilimits' kwarg to Axes.ticklabel_format() method, for easy access to the set_powerlimits method of the major ScalarFormatter. - EF
- 2008-10-04 Experimental new kwarg borderpad to replace pad in legend, based on suggestion by Jae-Joon Lee. - EF
- 2008-09-27 Allow spy to ignore zero values in sparse arrays, based on patch by Tony Yu. Also fixed plot to handle empty data arrays, and fixed handling of markers in figlegend. - EF
- 2008-09-24 Introduce drawstyles for lines. Transparently split linestyles like 'steps-' into drawstyle 'steps' and linestyle '-'. Legends always use drawstyle 'default'. - MM
- 2008-09-18 Fixed quiver and quiverkey bugs (failure to scale properly when resizing) and added additional methods for determining the arrow angles - EF
- 2008-09-18 Fix polar interpolation to handle negative values of theta - MGD
- 2008-09-14 Reorganized cbook and mlab methods related to numerical calculations that have little to do with the goals of those two modules into a separate module numerical_methods.py. Also, added ability to select points and stop point selection with keyboard in ginput and manual contour labeling code. Finally, fixed contour labeling bug. - DMK
- 2008-09-11 Fix backtick in Postscript output. - MGD
- 2008-09-10 [2089958] Path simplification for vector output backends. Leverage the simplification code exposed through path_to_polygons to simplify certain well-behaved paths in the vector backends (PDF, PS and SVG). "path.simplify" must be set to True in matplotlibrc for this to work. - MGD
- 2008-09-10 Add "filled" kwarg to Path.intersects_path and Path.intersects_bbox. - MGD
- 2008-09-07 Changed full arrows slightly to avoid an xpdf rendering problem reported by Friedrich Hagedorn. - JKS
- 2008-09-07 Fix conversion of quadratic to cubic Bezier curves in PDF and PS backends. Patch by Jae-Joon Lee. - JKS
- 2008-09-06 Added 5-point star marker to plot command - EF
- 2008-09-05 Fix hatching in PS backend - MGD
- 2008-09-03 Fix log with base 2 - MGD
- 2008-09-01 Added support for bilinear interpolation in NonUniformImage; patch by Gregory Lelens. - EF
- 2008-08-28 Added support for multiple histograms with data of different length - MM
- 2008-08-28 Fix step plots with log scale - MGD
- 2008-08-28 Fix masked arrays with markers in non-Agg backends - MGD
- 2008-08-28 Fix clip_on kwarg so it actually works correctly - MGD

2008-08-25 Fix locale problems in SVG backend - MGD

2008-08-22 fix quiver so masked values are not plotted - JSW

2008-08-18 improve interactive pan/zoom in qt4 backend on windows - DSD

2008-08-11 Fix more bugs in NaN/inf handling. In particular, path simplification (which does not handle NaNs or infs) will be turned off automatically when infs or NaNs are present. Also masked arrays are now converted to arrays with NaNs for consistent handling of masks and NaNs - MGD and EF

2008-08-03 Released 0.98.3 at svn r5947

2008-08-01 Backported memory leak fixes in `_ttconv.cpp` - MGD

2008-07-31 Added masked array support to `griddata`. - JSW

2008-07-26 Added optional `C` and `reduce_C_function` arguments to `axes.hexbin()`. This allows `hexbin` to accumulate the values of `C` based on the `x,y` coordinates and display in hexagonal bins. - ADS

2008-07-24 Deprecated (`raise NotImplementedError`) all the `mlab2` functions from `matplotlib.mlab` out of concern that some of them were not clean room implementations. JDH

2008-07-24 Rewrite of a significant portion of the `clabel` code (class `ContourLabeler`) to improve inlining. - DMK

2008-07-22 Added `Barbs` polygon collection (similar to `Quiver`) for plotting wind barbs. Added corresponding helpers to `Axes` and `pyplot` as well. (`examples/pylab_examples/barb_demo.py` shows it off.) - RMM

2008-07-21 Added `scikits.delaunay` as `matplotlib.delaunay`. Added `griddata` function in `matplotlib.mlab`, with example (`griddata_demo.py`) in `pylab_examples`. `griddata` function will use `mpl_toolkits._natgrid` if installed. - JSW

2008-07-21 Re-introduced `offset_copy` that works in the context of the new transforms. - MGD

2008-07-21 Committed patch by Ryan May to add `get_offsets` and `set_offsets` to `Collections` base class - EF

2008-07-21 Changed the "asarray" strategy in `image.py` so that colormapping of masked input should work for all image types (thanks Klaus Zimmerman) - EF

2008-07-20 Rewrote `cbook.delete_masked_points` and corresponding unit test to support `rgb` color array inputs, `datetime` inputs, etc. - EF

2008-07-20 Renamed `unit/axes_unit.py` to `cbook_unit.py` and modified in accord with Ryan's move of `delete_masked_points` from `axes` to `cbook`. - EF

2008-07-18 Check for `nan` and `inf` in `axes.delete_masked_points()`. This should help `hexbin` and `scatter` deal with `nans`. - ADS

2008-07-17 Added ability to manually select contour label locations. Also added a `waitforbuttonpress` function. - DMK

- 2008-07-17 Fix bug with NaNs at end of path (thanks, Andrew Straw for the report) - MGD
 - 2008-07-16 Improve error handling in texmanager, thanks to Ian Henry for reporting - DSD
 - 2008-07-12 Added support for external backends with the "module://my_backend" syntax - JDH
 - 2008-07-11 Fix memory leak related to shared axes. Grouper should store weak references. - MGD
 - 2008-07-10 Bugfix: crash displaying fontconfig pattern - MGD
 - 2008-07-10 Bugfix: [2013963] update_datalim_bounds in Axes not works - MGD
 - 2008-07-10 Bugfix: [2014183] multiple imshow() causes gray edges - MGD
 - 2008-07-09 Fix rectangular axes patch on polar plots bug - MGD
 - 2008-07-09 Improve mathtext radical rendering - MGD
 - 2008-07-08 Improve mathtext superscript placement - MGD
 - 2008-07-07 Fix custom scales in pcolormesh (thanks Matthew Turk) - MGD
 - 2008-07-03 Implemented findobj method for artist and pyplot - see examples/pylab_examples/findobj_demo.py - JDH
 - 2008-06-30 Another attempt to fix TextWithDash - DSD
 - 2008-06-30 Removed Qt4 NavigationToolbar2.destroy – it appears to have been unnecessary and caused a bug reported by P. Raybaut - DSD
 - 2008-06-27 Fixed tick positioning bug - MM
 - 2008-06-27 Fix dashed text bug where text was at the wrong end of the dash - MGD
 - 2008-06-26 Fix mathtext bug for expressions like x_{\leftarrow} - MGD
 - 2008-06-26 Fix direction of horizontal/vertical hatches - MGD
 - 2008-06-25 Figure.figurePatch renamed Figure.patch, Axes.axesPatch renamed Axes.patch, Axes.axesFrame renamed Axes.frame, Axes.get_frame, which returns Axes.patch, is deprecated. Examples and users guide updated - JDH
 - 2008-06-25 Fix rendering quality of pcolor - MGD
-
- 2008-06-24 Released 0.98.2 at svn r5667 - (source only for debian) JDH
 - 2008-06-24 Added "transparent" kwarg to savefig. - MGD
 - 2008-06-24 Applied Stefan's patch to draw a single centered marker over a line with num-points==1 - JDH
 - 2008-06-23 Use splines to render circles in scatter plots - MGD
-
- 2008-06-22 Released 0.98.1 at revision 5637
-

- 2008-06-22 Removed axes3d support and replaced it with a `NotImplementedError` for one release cycle
- 2008-06-21 fix marker placement bug in `backend_ps` - DSD
- 2008-06-20 [1978629] scale documentation missing/incorrect for `log` - MGD
- 2008-06-20 Added closed kwarg to `PolyCollection`. Fixes bug [1994535] still missing lines on graph with `svn` (r 5548). - MGD
- 2008-06-20 Added `set/get_closed` method to `Polygon`; fixes error in `hist` - MM
- 2008-06-19 Use relative font sizes (e.g., 'medium' and 'large') in `rcsetup.py` and `matplotlib.brc.template` so that text will be scaled by default when changing `rcParams['font.size']` - EF
- 2008-06-17 Add a generic `PatchCollection` class that can contain any kind of patch. - MGD
- 2008-06-13 Change pie chart label alignment to avoid having labels overwrite the pie - MGD
- 2008-06-12 Added some helper functions to the `mathtext` parser to return bitmap arrays or write pngs to make it easier to use `mathtext` outside the context of an `mpl` figure. modified the `mathpng sphinxext` to use the `mathtext` png save functionality - see `examples/api/mathtext_asarray.py` - JDH
- 2008-06-11 Use `matplotlib.mathtext` to render math expressions in online docs - MGD
- 2008-06-11 Move PNG loading/saving to its own extension module, and remove duplicate code in `__backend_agg.cpp` and `__image.cpp` that does the same thing - MGD
- 2008-06-11 Numerous `mathtext` bugfixes, primarily related to dpi-independence - MGD
- 2008-06-10 `Bar` now applies the label only to the first patch only, and sets `'_nolegend_'` for the other patch labels. This lets `autolegend` work as expected for `hist` and `bar` - see https://sourceforge.net/tracker/index.php?func=detail&aid=1986597&group_id=80706&atid=560720 JDH
- 2008-06-10 Fix text baseline alignment bug. [1985420] Repair of baseline alignment in `Text._get_layout`. Thanks Stan West - MGD
- 2008-06-09 Committed Gregor's image resample patch to downsampling images with new `rc-param` `image.resample` - JDH
- 2008-06-09 Don't install `Enthought.Traits` along with `matplotlib`. For `matplotlib` developers convenience, it can still be installed by setting an option in `setup.cfg` while we figure decide if there is a future for the traited config - DSD
- 2008-06-09 Added range keyword arg to `hist()` - MM
- 2008-06-07 Moved list of backends to `rcsetup.py`; made use of lower case for backend names consistent; use `validate_backend` when importing backends subpackage - EF
- 2008-06-06 `hist()` revision, applied ideas proposed by Erik Tollerud and Olle Engdegard: make `histtype='step'` unfilled by default and introduce `histtype='stepfilled'`; use default color cycle; introduce reverse cumulative histogram; new `align` keyword - MM
- 2008-06-06 Fix closed polygon patch and also provide the option to not close the polygon - MGD

- 2008-06-05 Fix some dpi-changing-related problems with PolyCollection, as called by Axes.scatter() - MGD
 - 2008-06-05 Fix image drawing so there is no extra space to the right or bottom - MGD
 - 2006-06-04 Added a figure title command `suptitle` as a Figure method and pyplot command – see `examples/figure_title.py` - JDH
 - 2008-06-02 Added support for log to hist with `histtype='step'` and fixed a bug for log-scale stacked histograms - MM
-
- 2008-05-29 Released 0.98.0 at revision 5314
 - 2008-05-29 `matplotlib.image.imread` now no longer always returns RGBA – if the image is luminance or RGB, it will return a MxN or MxNx3 array if possible. Also `uint8` is no longer always forced to float.
 - 2008-05-29 Implement path clipping in PS backend - JDH
 - 2008-05-29 Fixed two bugs in `texmanager.py`: improved comparison of `dvipng` versions fixed a bug introduced when `get_grey` method was added - DSD
 - 2008-05-28 Fix crashing of PDFs in `xpdf` and `ghostscript` when two-byte characters are used with Type 3 fonts - MGD
 - 2008-05-28 Allow keyword args to configure widget properties as requested in http://sourceforge.net/tracker/index.php?func=detail&aid=1866207&group_id=80706&atid=560722 - JDH
 - 2008-05-28 Replaced `'-'` with `u'u2212'` for minus sign as requested in http://sourceforge.net/tracker/index.php?func=detail&aid=1962574&group_id=80706&atid=560720
 - 2008-05-28 zero width/height Rectangles no longer influence the autoscaler. Useful for log histograms with empty bins - JDH
 - 2008-05-28 Fix rendering of composite glyphs in Type 3 conversion (particularly as evidenced in the Eunjin.ttf Korean font) Thanks Jae-Joon Lee for finding this!
 - 2008-05-27 Rewrote the `cm.ScalarMappable` callback infrastructure to use `cbook.CallbackRegistry` rather than custom callback handling. Amy users of `add_observer/notify` of the `cm.ScalarMappable` should use the `cm.ScalarMappable.callbacksSM CallbackRegistry` instead. JDH
 - 2008-05-27 Fix `TkAgg` build on Ubuntu 8.04 (and hopefully a more general solution for other platforms, too.)
 - 2008-05-24 Added PIL support for loading images to `imread` (if PIL is available) - JDH
 - 2008-05-23 Provided a function and a method for controlling the plot color cycle. - EF
 - 2008-05-23 Major revision of `hist()`. Can handle 2D arrays and create stacked histogram plots; keyword `'width'` deprecated and `rwidth` (relative width) introduced; `align='edge'` changed to center of bin - MM
 - 2008-05-22 Added support for ReST-based documentation using Sphinx. Documents are located in `doc/`, and are broken up into a users guide and an API reference. To build, run the

make.py files. Sphinx-0.4 is needed to build generate xml, which will be useful for rendering equations with mathml, use sphinx from svn until 0.4 is released - DSD

2008-05-21 Fix segfault in TkAgg backend - MGD

2008-05-21 Fix a "local variable unreferenced" bug in plotfile - MM

2008-05-19 Fix crash when Windows can not access the registry to determine font path [Bug 1966974, thanks Patrik Simons] - MGD

2008-05-16 removed some unneeded code w/ the python 2.4 requirement. cbook no longer provides compatibility for reversed, enumerate, set or izip. removed lib/subprocess, mpl1, sandbox/units, and the swig code. This stuff should remain on the maintenance branch for archival purposes. JDH

2008-05-16 Reorganized examples dir - JDH

2008-05-16 Added 'elinewidth' keyword arg to errorbar, based on patch by Christopher Brown - MM

2008-05-16 Added 'cumulative' keyword arg to hist to plot cumulative histograms. For normed hists, this is normalized to one - MM

2008-05-15 Fix Tk backend segfault on some machines - MGD

2008-05-14 Don't use stat on Windows (fixes font embedding problem) - MGD

2008-05-09 Fix /singlequote (') in Postscript backend - MGD

2008-05-08 Fix kerning in SVG when embedding character outlines - MGD

2008-05-07 Switched to future numpy histogram semantic in hist - MM

2008-05-06 Fix strange colors when blitting in QtAgg and Qt4Agg - MGD

2008-05-05 pass notify_axes_change to the figure's add_axobserver in the qt backends, like we do for the other backends. Thanks Glenn Jones for the report - DSD

2008-05-02 Added step histograms, based on patch by Erik Tollerud. - MM

2008-05-02 On PyQt <= 3.14 there is no way to determine the underlying Qt version. [1851364] - MGD

2008-05-02 Don't call sys.exit() when pyemf is not found [1924199] - MGD

2008-05-02 Update _subprocess.c from upstream Python 2.5.2 to get a few memory and reference-counting-related bugfixes. See bug 1949978. - MGD

2008-04-30 Added some record array editing widgets for gtk - see examples/rec_edit*.py - JDH

2008-04-29 Fix bug in mlab.sqrtn - MM

2008-04-28 Fix bug in SVG text with Mozilla-based viewers (the symbol tag is not supported) - MGD

2008-04-27 Applied patch by Michiel de Hoon to add hexbin axes method and pyplot function - EF

- 2008-04-25 Enforce python \geq 2.4; remove subprocess build - EF
- 2008-04-25 Enforce the numpy requirement at build time - JDH
- 2008-04-24 Make numpy 1.1 and python 2.3 required when importing matplotlib - EF
- 2008-04-24 Fix compilation issues on VS2003 (Thanks Martin Spacek for all the help) - MGD
- 2008-04-24 Fix sub/superscripts when the size of the font has been changed - MGD
- 2008-04-22 Use "svg.embed_char_paths" consistently everywhere - MGD
- 2008-04-20 Add support to MaxNLocator for symmetric axis autoscaling. - EF
- 2008-04-20 Fix double-zoom bug. - MM
- 2008-04-15 Speed up color mapping. - EF
- 2008-04-12 Speed up zooming and panning of dense images. - EF
- 2008-04-11 Fix global font rcParam setting after initialization time. - MGD
- 2008-04-11 Revert commits 5002 and 5031, which were intended to avoid an unnecessary call to draw(). 5002 broke saving figures before show(). 5031 fixed the problem created in 5002, but broke interactive plotting. Unnecessary call to draw still needs resolution - DSD
- 2008-04-07 Improve color validation in rc handling, suggested by Lev Givon - EF
- 2008-04-02 Allow to use both linestyle definition arguments, '-' and 'solid' etc. in plots/collections - MM
- 2008-03-27 Fix saving to Unicode filenames with Agg backend (other backends appear to already work...) (Thanks, Christopher Barker) - MGD
- 2008-03-26 Fix SVG backend bug that prevents copying and pasting in Inkscape (thanks Kaushik Ghose) - MGD
- 2008-03-24 Removed an unnecessary call to draw() in the backend_qt* mouseReleaseEvent. Thanks to Ted Drain - DSD
- 2008-03-23 Fix a pdf backend bug which sometimes caused the outermost gsave to not be balanced with a grestore. - JKS
- 2008-03-20 Fixed a minor bug in ContourSet._process_linestyles when len(linestyles)==Nlev - MM
- 2008-03-19 Changed ma import statements to "from numpy import ma"; this should work with past and future versions of numpy, whereas "import numpy.ma as ma" will work only with numpy \geq 1.05, and "import numerix.npyma as ma" is obsolete now that maskedarray is replacing the earlier implementation, as of numpy 1.05.
- 2008-03-14 Removed an apparently unnecessary call to FigureCanvasAgg.draw in backend_qt*agg. Thanks to Ted Drain - DSD
- 2008-03-10 Workaround a bug in backend_qt4agg's blitting due to a buffer width/bbox width mismatch in _backend_agg's copy_from_bbox - DSD
- 2008-02-29 Fix class Wx toolbar pan and zoom functions (Thanks Jeff Peery) - MGD

- 2008-02-16 Added some new rec array functionality to mlab (rec_summarize, rec2txt and rec_groupby). See examples/rec_groupby_demo.py. Thanks to Tim M for rec2txt.
- 2008-02-12 Applied Erik Tollerud's span selector patch - JDH
- 2008-02-11 Update plotting() doc string to refer to getp/setp. - JKS
- 2008-02-10 Fixed a problem with square roots in the pdf backend with usetex. - JKS
- 2008-02-08 Fixed minor __str__ bugs so getp(gca()) works. - JKS
- 2008-02-05 Added getters for title, xlabel, ylabel, as requested by Brandon Kieth - EF
- 2008-02-05 Applied Gael's ginput patch and created examples/ginput_demo.py - JDH
- 2008-02-03 Expose interpnames, a list of valid interpolation methods, as an AxesImage class attribute. - EF
- 2008-02-03 Added BoundaryNorm, with examples in colorbar_only.py and image_masked.py. - EF
- 2008-02-03 Force dpi=72 in pdf backend to fix picture size bug. - JKS
- 2008-02-01 Fix doubly-included font problem in Postscript backend - MGD
- 2008-02-01 Fix reference leak in ft2font Glyph objects. - MGD
- 2008-01-31 Don't use unicode strings with usetex by default - DSD
- 2008-01-31 Fix text spacing problems in PDF backend with some fonts, such as STIXGeneral.
- 2008-01-31 Fix sqrt with radical number (broken by making [and] work below) - MGD
- 2008-01-27 Applied Martin Teichmann's patch to improve the Qt4 backend. Uses Qt's builtin toolbars and statusbars. See bug 1828848 - DSD
- 2008-01-10 Moved toolkits to mpl_toolkits, made mpl_toolkits a namespace package - JSWHIT
- 2008-01-10 Use setup.cfg to set the default parameters (tkagg, numpy) when building windows installers - DSD
- 2008-01-10 Fix bug displaying [and] in mathtext - MGD
- 2008-01-10 Fix bug when displaying a tick value offset with scientific notation. (Manifests itself as a warning that the times symbol can not be found). - MGD
- 2008-01-10 Use setup.cfg to set the default parameters (tkagg, numpy) when building windows installers - DSD
-
- 2008-01-06 Released 0.91.2 at revision 4802
- 2007-12-26 Reduce too-late use of matplotlib.use() to a warning instead of an exception, for backwards compatibility - EF
- 2007-12-25 Fix bug in errorbar, identified by Noriko Minakawa - EF
- 2007-12-25 Changed masked array importing to work with the upcoming numpy 1.05 (now the maskedarray branch) as well as with earlier versions. - EF

- 2007-12-16 rec2csv saves doubles without losing precision. Also, it does not close filehandles passed in open. - JDH,ADS
- 2007-12-13 Moved rec2gtk to matplotlib.toolkits.gtktools and rec2excel to matplotlib.toolkits.exceltools - JDH
- 2007-12-12 Support alpha-blended text in the Agg and Svg backends - MGD
- 2007-12-10 Fix SVG text rendering bug. - MGD
- 2007-12-10 Increase accuracy of circle and ellipse drawing by using an 8-piece bezier approximation, rather than a 4-piece one. Fix PDF, SVG and Cairo backends so they can draw paths (meaning ellipses as well). - MGD
- 2007-12-07 Issue a warning when drawing an image on a non-linear axis. - MGD
- 2007-12-06 let widgets.Cursor initialize to the lower x and y bounds rather than 0,0, which can cause havoc for dates and other transforms - DSD
- 2007-12-06 updated references to mpl data directories for py2exe - DSD
- 2007-12-06 fixed a bug in rcsetup, see bug 1845057 - DSD
- 2007-12-05 Fix how fonts are cached to avoid loading the same one multiple times. (This was a regression since 0.90 caused by the refactoring of font_manager.py) - MGD
- 2007-12-05 Support arbitrary rotation of usetex text in Agg backend. - MGD
- 2007-12-04 Support '|' as a character in mathtext - MGD

2007-11-27 Released 0.91.1 at revision 4517

2007-11-27 Released 0.91.0 at revision 4478

- 2007-11-13 All backends now support writing to a file-like object, not just a regular file. savefig() can be passed a file-like object in place of a file path. - MGD
- 2007-11-13 Improved the default backend selection at build time: SVG -> Agg -> TkAgg -> WXAgg -> GTK -> GTKAgg. The last usable backend in this progression will be chosen in the default config file. If a backend is defined in setup.cfg, that will be the default backend - DSD
- 2007-11-13 Improved creation of default config files at build time for traitlets config package - DSD
- 2007-11-12 Exposed all the build options in setup.cfg. These options are read into a dict called "options" by setupext.py. Also, added "-mpl" tags to the version strings for packages provided by matplotlib. Versions provided by mpl will be identified and updated on subsequent installs - DSD
- 2007-11-12 Added support for STIX fonts. A new rcParam, mathtext.fontset, can be used to choose between:
- 'cm': The TeX/LaTeX Computer Modern fonts
 - 'stix': The STIX fonts (see stixfonts.org)

'stixsans': The STIX fonts, using sans-serif glyphs by default

'custom': A generic Unicode font, in which case the mathtext font must be specified using `mathtext.bf`, `mathtext.it`, `mathtext.sf` etc.

Added a new example, `stix_fonts_demo.py` to show how to access different fonts and unusual symbols.

- MGD

2007-11-12 Options to disable building backend extension modules moved from `setup.py` to `setup.cfg` - DSD

2007-11-09 Applied Martin Teichmann's patch 1828813: a `QPainter` is used in `paintEvent`, which has to be destroyed using the method `end()`. If matplotlib raises an exception before the call to `end` - and it does if you feed it with bad data - this method `end()` is never called and Qt4 will start spitting error messages

2007-11-09 Moved `pyparsing` back into matplotlib namespace. Don't use `system pyparsing`, API is too variable from one release to the next - DSD

2007-11-08 Made `pylab` use straight `numpy` instead of `oldnumeric` by default - EF

2007-11-08 Added additional record array utilites to `mlab` (`rec2excel`, `rec2gtk`, `rec_join`, `rec_append_field`, `rec_drop_field`) - JDH

2007-11-08 Updated `pytz` to version 2007g - DSD

2007-11-08 Updated `pyparsing` to version 1.4.8 - DSD

2007-11-08 Moved `csv2rec` to `recutils` and added other record array utilities - JDH

2007-11-08 If available, use existing `pyparsing` installation - DSD

2007-11-07 Removed old `enthought.traits` from `lib/matplotlib`, added Gael Varoquaux's `enthought.traits-2.6b1`, which is stripped of `setuptools`. The package is installed to site-packages if not already available - DSD

2007-11-05 Added easy access to minor tick properties; slight mod of patch by Pierre G-M - EF

2007-11-02 Committed Phil Thompson's patch 1599876, fixes to `Qt4Agg` backend and `qt4 blitting` demo - DSD

2007-11-02 Committed Phil Thompson's patch 1599876, fixes to `Qt4Agg` backend and `qt4 blitting` demo - DSD

2007-10-31 Made log color scale easier to use with `contourf`; automatic level generation now works. - EF

2007-10-29 TRANSFORMS REFACTORING

The primary goal of this refactoring was to make it easier to extend matplotlib to support new kinds of projections. This is primarily an internal improvement, and the possible user-visible changes it allows are yet to come.

The transformation framework was completely rewritten in Python (with Numpy). This will make it easier to add news kinds of transformations without writing C/C++ code.

Transforms are composed into a 'transform tree', made of transforms whose value depends on other transforms (their children). When the contents of children change, their parents are automatically updated to reflect those changes. To do this an "invalidation" method is used: when children change, all of their ancestors are marked as "invalid". When the value of a transform is accessed at a later time, its value is recomputed only if it is invalid, otherwise a cached value may be used. This prevents unnecessary recomputations of transforms, and contributes to better interactive performance.

The framework can be used for both affine and non-affine transformations. However, for speed, we want use the backend renderers to perform affine transformations whenever possible. Therefore, it is possible to perform just the affine or non-affine part of a transformation on a set of data. The affine is always assumed to occur after the non-affine. For any transform:

full transform == non-affine + affine

Much of the drawing has been refactored in terms of compound paths. Therefore, many methods have been removed from the backend interface and replaced with a handful to draw compound paths. This will make updating the backends easier, since there is less to update. It also should make the backends more consistent in terms of functionality.

User visible changes:

- POLAR PLOTS: Polar plots are now interactively zoomable, and the r-axis labels can be interactively rotated. Straight line segments are now interpolated to follow the curve of the r-axis.
- Non-rectangular clipping works in more backends and with more types of objects.
- Sharing an axis across figures is now done in exactly the same way as sharing an axis between two axes in the same figure:

```
fig1 = figure()
fig2 = figure()

ax1 = fig1.add_subplot(111)
ax2 = fig2.add_subplot(111, sharex=ax1, sharey=ax1)
```

- linestyle now include steps-pre, steps-post and steps-mid. The old step still works and is equivalent to step-pre.
- Multiple line styles may be provided to a collection.

See `API_CHANGES` for more low-level information about this refactoring.

2007-10-24 Added ax kwarg to `Figure.colorbar` and `pyplot.colorbar` - EF

2007-10-19 Removed a `gsave/grestore` pair surrounding `_draw_ps`, which was causing a loss graphics state info (see "EPS output problem - scatter & edgcolors" on `mpl-dev`, 2007-10-29) - DSD

2007-10-15 Fixed a bug in `patches.Ellipse` that was broken for `aspect='auto'`. Scale free ellipses now work properly for equal and auto on Agg and PS, and they fall back on a polygonal

- approximation for nonlinear transformations until we convince ourselves that the spline approximation holds for nonlinear transformations. Added `unit/ellipse_compare.py` to compare spline with vertex approx for both aspects. JDH
- 2007-10-05 remove generator expressions from `texmanager` and `mpltraits`. generator expressions are not supported by python-2.3 - DSD
- 2007-10-01 Made `matplotlib.use()` raise an exception if called after backends has been imported. - EF
- 2007-09-30 Modified `update*` methods of `Bbox` and `Interval` so they work with reversed axes. Prior to this, trying to set the ticks on a reversed axis failed with an uninformative error message. - EF
- 2007-09-30 Applied patches to `axes3d` to fix index error problem - EF
- 2007-09-24 Applied Eike Welk's patch reported on `mpl-dev` on 2007-09-22 Fixes a bug with multiple plot windows in the qt backend, ported the changes to `backend_qt4` as well - DSD
- 2007-09-21 Changed `cbook.reversed` to yield the same result as the python reversed builtin - DSD
- 2007-09-13 The `usetex` support in the pdf backend is more usable now, so I am enabling it. - JKS
- 2007-09-12 Fixed a `Axes.bar` unit bug - JDH
- 2007-09-10 Made `skiprows=1` the default on `csv2rec` - JDH
- 2007-09-09 Split out the plotting part of `pylab` and put it in `pyplot.py`; removed `numerix` from the remaining `pylab.py`, which imports everything from `pyplot.py`. The intention is that apart from cleanups, the result of importing from `pylab` is nearly unchanged, but there is the new alternative of importing from `pyplot` to get the state-engine graphics without all the numeric functions. Numpified examples; deleted two that were obsolete; modified some to use `pyplot`. - EF
- 2007-09-08 Eliminated `gd` and `paint` backends - EF
- 2007-09-06 `.bmp` file format is now longer an alias for `.raw`
- 2007-09-07 Added clip path support to pdf backend. - JKS
- 2007-09-06 Fixed a bug in the embedding of Type 1 fonts in PDF. Now it doesn't crash `Preview.app`. - JKS
- 2007-09-06 Refactored image saving code so that all GUI backends can save most image types. See `FILETYPES` for a matrix of backends and their supported file types. Backend canvases should no longer write their own `print_figure()` method – instead they should write a `print_xxx` method for each filetype they can output and add an entry to their class-scoped `filetypes` dictionary. - MGD
- 2007-09-05 Fixed Qt version reporting in `setuptools.py` - DSD
- 2007-09-04 Embedding Type 1 fonts in PDF, and thus `usetex` support via `dviread`, sort of works. To test, enable it by renaming `_draw_tex` to `draw_tex`. - JKS
- 2007-09-03 Added ability of errorbar show limits via caret or arrowhead ends on the bars; patch by Manual Metz. - EF

2007-09-03 Created `type1font.py`, added features to `AFM` and `FT2Font` (see `API_CHANGES`), started work on embedding Type 1 fonts in pdf files. - JKS

2007-09-02 Continued work on `dviread.py`. - JKS

2007-08-16 Added a `set_extent` method to `AxesImage`, allow data extent to be modified after initial call to `imshow` - DSD

2007-08-14 Fixed a bug in `pyqt4` `subplots-adjust`. Thanks to Xavier Gnata for the report and suggested fix - DSD

2007-08-13 Use `pickle` to cache entire `fontManager`; change to using `font_manager` module-level function `findfont` wrapper for the `fontManager.findfont` method - EF

2007-08-11 Numpification and cleanup of `mlab.py` and some examples - EF

2007-08-06 Removed `mathtext2`

2007-07-31 Refactoring of `distutils` scripts.

- Will not fail on the entire build if an optional Python package (e.g., `Tkinter`) is installed but its development headers are not (e.g., `tk-devel`). Instead, it will continue to build all other extensions.
- Provide an overview at the top of the output to display what dependencies and their versions were found, and (by extension) what will be built.
- Use `pkg-config`, when available, to find `freetype2`, since this was broken on Mac OS-X when using `MacPorts` in a non-standard location.

2007-07-30 Reorganized configuration code to work with `traited config` objects. The new config system is located in the `matplotlib.config` package, but it is disabled by default. To enable it, set `NEWCONFIG=True` in `matplotlib.__init__.py`. The new configuration system will still use the old `matplotlibrc` files by default. To switch to the experimental, `traited` configuration, set `USE_TRAITED_CONFIG=True` in `config.__init__.py`.

2007-07-29 Changed default `pcolor` shading to flat; added aliases to make collection `kwargs` agree with setter names, so updating works; related minor cleanups. Removed `quiver_classic`, `scatter_classic`, `pcolor_classic`. - EF

2007-07-26 Major rewrite of `mathtext.py`, using the TeX box layout model.

There is one (known) backward incompatible change. The font commands (`cal`, `rm`, `it`, `tt`) now behave as TeX does: they are in effect until the next font change command or the end of the grouping. Therefore uses of `$cal{R}$` should be changed to `#{cal R}$`. Alternatively, you may use the new LaTeX-style font commands (`mathcal`, `mathrm`, `mathit`, `mathtt`) which do affect the following group, e.g., `$mathcal{R}$`.

Other new features include:

- Math may be interspersed with non-math text. Any text with an even number of `$`'s (non-escaped) will be sent to the `mathtext` parser for layout.
- Sub/superscripts are less likely to accidentally overlap.
- Support for sub/superscripts in either order, e.g., `x^i_j` and `x_j^i` are equivalent.

- Double sub/superscripts (e.g., x_i_j) are considered ambiguous and raise an exception. Use braces to disambiguate.
- $\frac{x}{y}$ can be used for displaying fractions.
- $\sqrt[3]{x}$ can be used to display the radical symbol with a root number and body.
- $\left(\frac{x}{y}\right)$ may be used to create parentheses and other delimiters that automatically resize to the height of their contents.
- Spacing around operators etc. is now generally more like TeX.
- Added support (and fonts) for boldface (bf) and sans-serif (sf) symbols.
- Log-like function name shortcuts are supported. For example, $\sin(x)$ may be used instead of $\{\rm sin\}(x)$
- Limited use of kerning for the easy case (same font)

Behind the scenes, the `pyparsing.py` module used for doing the math parsing was updated to the latest stable version (1.4.6). A lot of duplicate code was refactored out of the `Font` classes.

- MGD

2007-07-19 completed numpification of most trivial cases - NN

2007-07-19 converted non-numpy relicts throughout the code - NN

2007-07-19 replaced the Python code in `numerix/` by a minimal wrapper around `numpy` that explicitly mentions all symbols that need to be addressed for further numpification - NN

2007-07-18 make `usetex` respect changes to `rcParams`. `texmanager` used to only configure itself when it was created, now it reconfigures when `rcParams` are changed. Thank you Alexander Schmolck for contributing a patch - DSD

2007-07-17 added validation to setting and changing `rcParams` - DSD

2007-07-17 bugfix segfault in `transforms` module. Thanks Ben North for the patch. - ADS

2007-07-16 clean up some code in `ticker.ScalarFormatter`, use unicode to render multiplication sign in offset ticklabel - DSD

2007-07-16 fixed a formatting bug in `ticker.ScalarFormatter`'s scientific notation (10^0 was being rendered as 10 in some cases) - DSD

2007-07-13 Add `MPL_isfinite64()` and `MPL_isinf64()` for testing doubles in (the now misnamed) `MPL_isnan.h`. - ADS

2007-07-13 The `matplotlib._isnan` module removed (use `numpy.isnan`) - ADS

2007-07-13 Some minor cleanups in `_transforms.cpp` - ADS

2007-07-13 Removed the rest of the `numerix` extension code detritus, numpified `axes.py`, and cleaned up the imports in `axes.py` - JDH

2007-07-13 Added `legend.loc` as configurable option that could in future default to 'best'. - NN

- 2007-07-12 Bugfixes in mlab.py to coerce inputs into numpy arrays. -ADS
- 2007-07-11 Added linespacing kwarg to text.Text - EF
- 2007-07-11 Added code to store font paths in SVG files. - MGD
- 2007-07-10 Store subset of TTF font as a Type 3 font in PDF files. - MGD
- 2007-07-09 Store subset of TTF font as a Type 3 font in PS files. - MGD
- 2007-07-09 Applied Paul's pick restructure pick and add pickers, [sourceforge patch 1749829](#) - JDH
- 2007-07-09 Applied Allan's draw_lines agg optimization. JDH
- 2007-07-08 Applied Carl Worth's patch to fix cairo draw_arc - SC
- 2007-07-07 fixed bug 1712099: xpdf distiller on windows - DSD
- 2007-06-30 Applied patches to tkagg, gtk, and wx backends to reduce memory leakage. Patches supplied by Mike Droettboom; see tracker numbers 1745400, 1745406, 1745408. Also made unit/memleak_gui.py more flexible with command-line options. - EF
- 2007-06-30 Split defaultParams into separate file rcdefaults (together with validation code). Some heavy refactoring was necessary to do so, but the overall behavior should be the same as before. - NN
- 2007-06-27 Added MPLCONFIGDIR for the default location for mpl data and configuration. useful for some apache installs where HOME is not writable. Tried to clean up the logic in __get_config_dir to support non-writable HOME where are writable HOME/.matplotlib already exists - JDH
- 2007-06-27 Fixed locale bug reported at http://sourceforge.net/tracker/index.php?func=detail&aid=1744154&group_id=80706&atid=560720 by adding a cbook.unicode_safe function - JDH
- 2007-06-27 Applied Micheal's tk savefig bugfix described at http://sourceforge.net/tracker/index.php?func=detail&aid=1716732&group_id=80706&atid=560720 Thanks Michael!
- 2007-06-27 Patch for get_py2exe_datafiles() to work with new directory layout. (Thanks Tocer and also Werner Bruhin.) -ADS
- 2007-06-27 Added a scroll event to the mpl event handling system and implemented it for backends GTK* – other backend users/developers/maintainers, please add support for your backend. - JDH
- 2007-06-25 Changed default to clip=False in colors.Normalize; modified ColorbarBase for easier colormap display - EF
- 2007-06-13 Added maskedarray option to rc, numerix - EF
- 2007-06-11 Python 2.5 compatibility fix for mlab.py - EF
- 2007-06-10 In matplotlibrc file, use 'dashed' | 'solid' instead of a pair of floats for contour.negative_linestyle - EF
- 2007-06-08 Allow plot and fill fmt string to be any mpl string colorspec - EF

- 2007-06-08 Added `gnuplot` file `plotfile` function to `pylab` – see `examples/plotfile_demo.py` - JDH
- 2007-06-07 Disable build of `numarray` and `Numeric` extensions for internal MPL use and the `numerix` layer. - ADS
- 2007-06-07 Added `csv2rec` to `matplotlib.mlab` to support automatically converting `csv` files to record arrays using type introspection, and turned on native `datetime` support using the new units support in `matplotlib.dates`. See `examples/loadrec.py` ! JDH
- 2007-06-07 Simplified internal code of `__auto__legend__data` - NN
- 2007-06-04 Added `labeldistance` arg to `Axes.pie` to control the radial distance of the wedge labels - JDH
- 2007-06-03 Turned `mathtext` in SVG into single `<text>` with multiple `<tspan>` objects (easier to edit in `inkscape`). - NN
-
- 2007-06-02 Released 0.90.1 at revision 3352
- 2007-06-02 Display only meaningful labels when calling `legend()` without args. - NN
- 2007-06-02 Have errorbar follow the color cycle even if line is not plotted. Suppress plotting of errorbar caps for `capsize=0`. - NN
- 2007-06-02 Set markers to same alpha value as line. - NN
- 2007-06-02 Fix `mathtext` position in `svg` backend. - NN
- 2007-06-01 Deprecate `Numeric` and `numarray` for use as `numerix`. Props to Travis – job well done. - ADS
- 2007-05-18 Added LaTeX unicode support. Enable with the `'text.latex.unicode'` rcParam. This requires the `ucs` and `inputenc` LaTeX packages. - ADS
- 2007-04-23 Fixed some problems with polar – added general polygon clipping to clip the lines and grids to the polar axes. Added support for `set_rmax` to easily change the maximum radial grid. Added support for polar legend - JDH
- 2007-04-16 Added `Figure.autofmt_xdate` to handle adjusting the bottom and rotating the tick labels for date plots when the ticks often overlap - JDH
- 2007-04-09 Beginnings of `usetex` support for pdf backend. -JKS
- 2007-04-07 Fixed legend/LineCollection bug. Added label support to collections. - EF
- 2007-04-06 Removed deprecated support for a float value as a gray-scale; now it must be a string, like `'0.5'`. Added alpha kwarg to `ColorConverter.to_rgba_list`. - EF
- 2007-04-06 Fixed rotation of ellipses in pdf backend (sf bug #1690559) -JKS
- 2007-04-04 More `matshow` tweaks; documentation updates; new method `set_bounds()` for formatters and locators. - EF
- 2007-04-02 Fixed problem with `imshow` and `matshow` of integer arrays; fixed problems with changes to color autoscaling. - EF

- 2007-04-01 Made image color autoscaling work correctly with `ax.imshow()` a tracking colorbar; `norm.autoscale` now scales unconditionally, while `norm.autoscale_None` changes only None-valued `vmin`, `vmax`. - EF
- 2007-03-31 Added a qt-based subplot-adjustment dialog - DSD
- 2007-03-30 Fixed a bug in `backend_qt4`, reported on `mpl-dev` - DSD
- 2007-03-26 Removed `colorbar_classic` from `figure.py`; fixed bug in `Figure.clf()` in which `__axob-`
`servers` was not getting cleared. Modernization and cleanups. - EF
- 2007-03-26 Refactored some of the units support – units now live in the respective x and y `Axis`
instances. See also `API_CHANGES` for some alterations to the conversion interface. JDH
- 2007-03-25 Fix masked array handling in `quiver.py` for `numpy`. (`Numeric` and `numarray` support
for masked arrays is broken in other ways when using `quiver`. I didn't pursue that.) -
ADS
- 2007-03-23 Made `font_manager.py` close opened files. - JKS
- 2007-03-22 Made `imshow` default extent match `matshow` - EF
- 2007-03-22 Some more niceties for `xcorr` – a `maxlags` option, `normed` now works for `xcorr` as well
as `axcorr`, `usevlines` is supported, and a zero correlation `hline` is added. See exam-
ples/`xcorr_demo.py`. Thanks Sameer for the patch. - JDH
- 2007-03-21 `Axes.vlines` and `Axes.hlines` now create and returns a `LineCollection`, not a list of
lines. This is much faster. The `kwargs` signature has changed, so consult the docs. Modi-
fied `Axes.errorbar` which uses `vlines` and `hlines`. See `API_CHANGES`; the return signature
for these three functions is now different
- 2007-03-20 Refactored units support and added new examples - JDH
- 2007-03-19 Added Mike's units patch - JDH
- 2007-03-18 `Matshow` as an `Axes` method; test version `matshow1()` in `pylab`; added 'integer'
Boolean `kwargs` to `MaxNLocator` initializer to force ticks at integer locations. - EF
- 2007-03-17 Preliminary support for clipping to paths `agg` - JDH
- 2007-03-17 `Text.set_text()` accepts anything convertible with `'%s'` - EF
- 2007-03-14 Add masked-array support to `hist`. - EF
- 2007-03-03 Change `barh` to take a `kwargs` dict and pass it to `bar`. Fixes `sf` bug #1669506.
- 2007-03-02 Add `rc` parameter `pdf.inheritcolor`, which disables all color-setting operations in the
`pdf` backend. The idea is that you include the resulting file in another program and set
the colors (both stroke and fill color) there, so you can use the same `pdf` file for e.g., a paper
and a presentation and have them in the surrounding color. You will probably not want to
draw figure and axis frames in that case, since they would be filled in the same color. - JKS
- 2007-02-26 Prevent building `_wxagg.so` with broken Mac OS X `wxPython`. - ADS
- 2007-02-23 Require `setuptools` for Python 2.3 - ADS
- 2007-02-22 `WXAgg` accelerator updates - KM `WXAgg`'s C++ accelerator has been fixed to use
the correct `wxBitmap` constructor.

The backend has been updated to use new wxPython functionality to provide fast `blit()` animation without the C++ accelerator. This requires wxPython 2.8 or later. Previous versions of wxPython can use the C++ accelerator or the old pure Python routines.

`setup.py` no longer builds the C++ accelerator when wxPython ≥ 2.8 is present.

The `blit()` method is now faster regardless of which agg/wxPython conversion routines are used.

2007-02-21 Applied the PDF backend patch by Nicolas Grilly. This impacts several files and directories in matplotlib:

- Created the directory `lib/matplotlib/mpl-data/fonts/pdfcorefonts`, holding AFM files for the 14 PDF core fonts. These fonts are embedded in every PDF viewing application.
- `setup.py`: Added the directory `pdfcorefonts` to `package_data`.
- `lib/matplotlib/__init__.py`: Added the default parameter `'pdf.use14corefonts'`. When True, the PDF backend uses only the 14 PDF core fonts.
- `lib/matplotlib/afm.py`: Added some keywords found in recent AFM files. Added a little workaround to handle Euro symbol.
- `lib/matplotlib/fontmanager.py`: Added support for the 14 PDF core fonts. These fonts have a dedicated cache (file `pdfcorefont.cache`), not the same as for other AFM files (file `.afmfont.cache`). Also cleaned comments to conform to `CODING_GUIDE`.
- `lib/matplotlib/backends/backend_pdf.py`: Added support for 14 PDF core fonts. Fixed some issues with incorrect character widths and encodings (works only for the most common encoding, `WinAnsiEncoding`, defined by the official PDF Reference). Removed parameter `'dpi'` because it causes alignment issues.

-JKS (patch by Nicolas Grilly)

2007-02-17 Changed `ft2font.get_charmap`, and updated all the files where `get_charmap` is mentioned - ES

2007-02-13 Added barcode demo- JDH

2007-02-13 Added binary colormap to `cm` - JDH

2007-02-13 Added `twiny` to `pylab` - JDH

2007-02-12 Moved data files into `lib/matplotlib` so that `setuptools' develop` mode works. Re-organized the `mpl-data` layout so that this source structure is maintained in the installation. (i.e., the `'fonts'` and `'images'` sub-directories are maintained in `site-packages`.) Suggest removing `site-packages/matplotlib/mpl-data` and `~/.matplotlib/ttffont.cache` before installing - ADS

2007-02-07 Committed Rob Hetland's patch for qt4: remove references to `text()/latin1()`, plus some improvements to the toolbar layout - DSD

2007-02-06 Released 0.90.0 at revision 3003

- 2007-01-22 Extended the new picker API to text, patches and patch collections. Added support for user customizable pick hit testing and attribute tagging of the PickEvent - Details and examples in `examples/pick_event_demo.py` - JDH
- 2007-01-16 Begun work on a new pick API using the mpl event handling framework. Artists will define their own pick method with a configurable epsilon tolerance and return pick attrs. All artists that meet the tolerance threshold will fire a PickEvent with artist dependent attrs; e.g., a Line2D can set the indices attribute that shows the indices into the line that are within epsilon of the pick point. See `examples/pick_event_demo.py`. The implementation of pick for the remaining Artists remains to be done, but the core infrastructure at the level of event handling is in place with a proof-of-concept implementation for Line2D - JDH
- 2007-01-16 `src/_image.cpp`: update to use `Py_ssize_t` (for 64-bit systems). Use return value of `fread()` to prevent warning messages - SC
- 2007-01-15 `src/_image.cpp`: combine `buffer_argb32()` and `buffer_bgra32()` into a new method `color_conv(format)` - SC
- 2007-01-14 `backend_cairo.py`: update `draw_arc()` so that `examples/arctest.py` looks correct - SC
- 2007-01-12 `backend_cairo.py`: enable clipping. Update `draw_image()` so that `examples/contour_demo.py` looks correct - SC
- 2007-01-12 `backend_cairo.py`: fix `draw_image()` so that `examples/image_demo.py` now looks correct - SC
- 2007-01-11 Added `Axes.xcorr` and `Axes.acorr` to plot the cross correlation of x vs y or the auto-correlation of x. `pylab` wrappers also provided. See `examples/xcorr_demo.py` - JDH
- 2007-01-10 Added "Subplot.label_outer" method. It will set the visibility of the ticklabels so that yticklabels are only visible in the first column and xticklabels are only visible in the last row - JDH
- 2007-01-02 Added additional kwarg documentation - JDH
- 2006-12-28 Improved error message for nonpositive input to log transform; added log kwarg to `bar`, `barh`, and `hist`, and modified `bar` method to behave sensibly by default when the ordinate has a log scale. (This only works if the log scale is set before or by the call to `bar`, hence the utility of the log kwarg.) - EF
- 2006-12-27 `backend_cairo.py`: update `draw_image()` and `_draw_mathtext()` to work with `numpy` - SC
- 2006-12-20 Fixed `xpdf` dependency check, which was failing on windows. Removed `ps2eps` dependency check. - DSD
- 2006-12-19 Added Tim Leslie's spectral patch - JDH
- 2006-12-17 Added rc param `'axes.formatter.limits'` to control the default threshold for switching to scientific notation. Added convenience method `Axes.ticklabel_format()` for turning scientific notation on or off on either or both axes. - EF
- 2006-12-16 Added ability to turn control scientific notation in `ScalarFormatter` - EF
- 2006-12-16 Enhanced `boxplot` to handle more flexible inputs - EF

- 2006-12-13 Replaced calls to `where()` in `colors.py` with much faster `clip()` and `putmask()` calls; removed inappropriate uses of `getmaskorNone` (which should be needed only very rarely); all in response to profiling by David Cournapeau. Also fixed bugs in my 2-D array support from 12-09. - EF
- 2006-12-09 Replaced `spy` and `spy2` with the new `spy` that combines `marker` and `image` capabilities - EF
- 2006-12-09 Added support for plotting 2-D arrays with `plot`: `columns` are plotted as in Matlab - EF
- 2006-12-09 Added `linewidth` kwarg to `bar` and `barh`; fixed `arg` checking bugs - EF
- 2006-12-07 Made `pcolormesh` argument handling match `pcolor`; fixed `kwargs` handling problem noted by Pierre GM - EF
- 2006-12-06 Made `pcolor` support vector `X` and/or `Y` instead of requiring 2-D arrays - EF
- 2006-12-05 Made the default `Artist._transform` `None` (rather than invoking `identity_transform` for each artist only to have it overridden later). Use `artist.get_transform()` rather than `artist._transform`, even in derived classes, so that the default transform will be created lazily as needed - JDH
- 2006-12-03 Added `LogNorm` to `colors.py` as illustrated by `examples/pcolor_log.py`, based on suggestion by Jim McDonald. `Colorbar` modified to handle `LogNorm`. Norms have additional "inverse" method. - EF
- 2006-12-02 Changed class names in `colors.py` to match convention: `normalize` -> `Normalize`, `no_norm` -> `NoNorm`. Old names are still available. Changed `__init__.py` rc defaults to match those in `matplotlibrc` - EF
- 2006-11-22 Fixed bug in `set_*lim` that I had introduced on 11-15 - EF
- 2006-11-22 Added `examples/clippedline.py`, which shows how to clip line data based on view limits – it also changes the marker style when zoomed in - JDH
- 2006-11-21 Some `spy` bug-fixes and added `precision` arg per Robert C's suggestion - JDH
- 2006-11-19 Added semi-automatic docstring generation detailing all the `kwargs` that functions take using the artist introspection tools; e.g., `'help` text now details the scatter `kwargs` that control the `Text` properties - JDH
- 2006-11-17 Removed obsolete `scatter_classic`, leaving a stub to raise `NotImplementedError`; same for `pcolor_classic` - EF
- 2006-11-15 Removed obsolete `pcolor_classic` - EF
- 2006-11-15 Fixed 1588908 reported by Russel Owen; factored `nonsingular` method out of `ticker.py`, put it into `transforms.py` as a function, and used it in `set_xlim` and `set_ylim`. - EF
- 2006-11-14 Applied patch 1591716 by Ulf Larssen to fix a bug in `apply_aspect`. Modified and applied patch 1594894 by mdehoon to fix bugs and improve formatting in `lines.py`. Applied patch 1573008 by Greg Willden to make `psd` etc. plot full frequency range for complex inputs. - EF

- 2006-11-14 Improved the ability of the colorbar to track changes in corresponding image, pcolor, or contourf. - EF
 - 2006-11-11 Fixed bug that broke Numeric compatibility; added support for alpha to colorbar. The alpha information is taken from the mappable object, not specified as a kwarg. - EF
 - 2006-11-05 Added broken_barh function for making a sequence of horizontal bars broken by gaps – see examples/broken_barh.py
 - 2006-11-05 Removed lineprops and markerprops from the Annotation code and replaced them with an arrow configurable with kwarg arrowprops. See examples/annotation_demo.py - JDH
 - 2006-11-02 Fixed a pylab subplot bug that was causing axes to be deleted with hspace or wspace equals zero in subplots_adjust - JDH
 - 2006-10-31 Applied axes3d patch 1587359 http://sourceforge.net/tracker/index.php?func=detail&aid=1587359&group_id=80706&atid=560722 JDH
-
- 2006-10-26 Released 0.87.7 at revision 2835
 - 2006-10-25 Made "tiny" kwarg in Locator.nonsingular much smaller - EF
 - 2006-10-17 Closed sf bug 1562496 update line props dash/solid/cap/join styles - JDH
 - 2006-10-17 Complete overhaul of the annotations API and example code - See [matplotlib.text.Annotation](#) and examples/annotation_demo.py JDH
 - 2006-10-12 Committed Manuel Metz's StarPolygon code and examples/scatter_star_poly.py - JDH
 - 2006-10-11 commented out all default values in matplotlibrc.template Default values should generally be taken from defaultParam in `__init__.py` - the file matplotlib should only contain those values that the user wants to explicitly change from the default. (see thread "marker color handling" on matplotlib-devel)
 - 2006-10-10 Changed default comment character for load to '#' - JDH
 - 2006-10-10 deactivated rcfile-configurability of markerfacecolor and markeredgecolor. Both are now hardcoded to the special value 'auto' to follow the line color. Configurability at run-time (using function arguments) remains functional. - NN
 - 2006-10-07 introduced dummy argument magnification=1.0 to `FigImage.make_image` to satisfy unit test figimage_demo.py The argument is not yet handled correctly, which should only show up when using non-standard DPI settings in PS backend, introduced by patch #1562394. - NN
 - 2006-10-06 add backend-agnostic example: simple3d.py - NN
 - 2006-09-29 fix line-breaking for SVG-inline images (purely cosmetic) - NN
 - 2006-09-29 reworked `set_linestyle` and `set_marker` `markeredgecolor` and `markerfacecolor` now default to a special value "auto" that keeps the color in sync with the line color further, the intelligence of `axes.plot` is cleaned up, improved and simplified. Complete compatibility cannot

be guaranteed, but the new behavior should be much more predictable (see patch #1104615 for details) - NN

2006-09-29 changed implementation of clip-path in SVG to work around a limitation in inkscape - NN

2006-09-29 added two options to matplotlibrc: `svg.image_inline` `svg.image_noscale` see patch #1533010 for details - NN

2006-09-29 axes.py: cleaned up kwargs checking - NN

2006-09-29 setup.py: cleaned up setup logic - NN

2006-09-29 setup.py: check for required pygtk versions, fixes bug #1460783 - SC

2006-09-27 Released 0.87.6 at revision 2783

2006-09-24 Added line pointers to the Annotation code, and a pylab interface. See `matplotlib.text.Annotation`, `examples/annotation_demo.py` and `examples/annotation_demo_pylab.py` - JDH

2006-09-18 `mathtext2.py`: The SVG backend now supports the same things that the AGG backend does. Fixed some bugs with rendering, and out of bounds errors in the AGG backend - ES. Changed the return values of `math_parse_s_ft2font_svg` to support lines (fractions etc.)

2006-09-17 Added an Annotation class to facilitate annotating objects and an examples file `examples/annotation_demo.py`. I want to add dash support as in `TextWithDash`, but haven't decided yet whether inheriting from `TextWithDash` is the right base class or if another approach is needed - JDH

2006-09-05 Released 0.87.5 at revision 2761

2006-09-04 Added `nxutils` for some numeric add-on extension code – specifically a better/more efficient inside polygon tester (see `unit/inside_poly_*.py`) - JDH

2006-09-04 Made bitstream fonts the rc default - JDH

2006-08-31 Fixed alpha-handling bug in `ColorConverter`, affecting collections in general and `contour/contourf` in particular. - EF

2006-08-30 `ft2font.cpp`: Added `draw_rect_filled` method (now used by `mathtext2` to draw the fraction bar) to `FT2Font` - ES

2006-08-29 `setupext.py`: wrap calls to `tk.getvar()` with `str()`. On some systems, `getvar` returns a `Tcl_Obj` instead of a string - DSD

2006-08-28 `mathtext2.py`: Sub/superscripts can now be complex (i.e. fractions etc.). The demo is also updated - ES

2006-08-28 `font_manager.py`: Added `/usr/local/share/fonts` to list of X11 font directories - DSD

- 2006-08-28 `mathtext2.py`: Initial support for complex fractions. Also, rendering is now completely separated from parsing. The sub/superscripts now work better. Updated the `mathtext2_demo.py` - ES
- 2006-08-27 qt backends: don't create a `QApplication` when backend is imported, do it when the `FigureCanvasQt` is created. Simplifies applications where mpl is embedded in qt. Updated `embedding_in_qt*` examples - DSD
- 2006-08-27 `mathtext2.py`: Now the fonts are searched in the OS font dir and in the mpl-data dir. Also `env` is not a dict anymore. - ES
- 2006-08-26 minor changes to `__init__.py`, `mathtex2_demo.py`. Added `matplotlibrc` key "math-text.mathtext2" (removed the key "mathtext2") - ES
- 2006-08-21 `mathtext2.py`: Initial support for fractions Updated the `mathtext2_demo.py` `_mathtext_data.py`: removed "" from the unicode dicts `mathtext.py`: Minor modification (because of `_mathtext_data.py`)- ES
- 2006-08-20 Added `mathtext2.py`: Replacement for `mathtext.py`. Supports $\hat{}$, \rm , \cal etc., \sin , \cos etc., unicode, recursive nestings, inline math mode. The only backend currently supported is Agg `__init__.py`: added new rc params for `mathtext2` added `mathtext2_demo.py` example - ES
- 2006-08-19 Added `embedding_in_qt4.py` example - DSD
- 2006-08-11 Added scale free Ellipse patch for Agg - CM
- 2006-08-10 Added converters to and from julian dates to `matplotlib.dates` (`num2julian` and `julian2num`) - JDH
- 2006-08-08 Fixed widget locking so multiple widgets could share the event handling - JDH
- 2006-08-07 Added scale free Ellipse patch to SVG and PS - CM
- 2006-08-05 Re-organized imports in `numerix` for numpy 1.0b2 - TEO
- 2006-08-04 Added `draw_markers` to PDF backend. - JKS
- 2006-08-01 Fixed a bug in postscript's rendering of dashed lines - DSD
- 2006-08-01 `figure.py`: `savefig()` update docstring to add support for 'format' argument. `backend_cairo.py`: `print_figure()` add support 'format' argument. - SC
- 2006-07-31 Don't let postscript's xpdf distiller compress images - DSD
- 2006-07-31 Added `shallowcopy()` methods to all Transformations; removed `copy_bbox_transform` and `copy_bbox_transform_shallow` from `transforms.py`; added `offset_copy()` function to `transforms.py` to facilitate positioning artists with offsets. See `examples/transoffset.py`. - EF
- 2006-07-31 Don't let postscript's xpdf distiller compress images - DSD
- 2006-07-29 Fixed `numerix` polygon bug reported by Nick Fotopoulos. Added `inverse_numerix_xy()` transform method. Made `autoscale_view()` preserve axis direction (e.g., increasing down).- EF

- 2006-07-28 Added shallow bbox copy routine for transforms – mainly useful for copying transforms to apply offset to. - JDH
- 2006-07-28 Added resize method to FigureManager class for Qt and Gtk backend - CM
- 2006-07-28 Added subplots_adjust button to Qt backend - CM
- 2006-07-26 Use numerix more in collections. Quiver now handles masked arrays. - EF
- 2006-07-22 Fixed bug #1209354 - DSD
- 2006-07-22 make scatter() work with the kwarg "color". Closes bug 1285750 - DSD
- 2006-07-20 backend_cairo.py: require pycairo 1.2.0. print_figure() update to output SVG using cairo.
- 2006-07-19 Added blitting for Qt4Agg - CM
- 2006-07-19 Added lasso widget and example examples/lasso_demo.py - JDH
- 2006-07-18 Added blitting for QtAgg backend - CM
- 2006-07-17 Fixed bug #1523585: skip nans in semilog plots - DSD
- 2006-07-12 Add support to render the scientific notation label over the right-side y-axis - DSD
-
- 2006-07-11 Released 0.87.4 at revision 2558
- 2006-07-07 Fixed a usetex bug with older versions of latex - DSD
- 2006-07-07 Add compatibility for NumPy 1.0 - TEO
- 2006-06-29 Added a Qt4Agg backend. Thank you James Amundson - DSD
- 2006-06-26 Fixed a usetex bug. On windows, usetex will process postscript output in the current directory rather than in a temp directory. This is due to the use of spaces and tildes in windows paths, which cause problems with latex. The subprocess module is no longer used. - DSD
- 2006-06-22 Various changes to bar(), barh(), and hist(). Added 'edgecolor' keyword arg to bar() and barh(). The x and y args in barh() have been renamed to width and bottom respectively, and their order has been swapped to maintain a (position, value) order ala matlab. left, height, width and bottom args can now all be scalars or sequences. barh() now defaults to edge alignment instead of center alignment. Added a keyword arg 'align' to bar(), barh() and hist() that controls between edge or center bar alignment. Fixed ignoring the rcParams['patch.facecolor'] for bar color in bar() and barh(). Fixed ignoring the rcParams['lines.color'] for error bar color in bar() and barh(). Fixed a bug where patches would be cleared when error bars were plotted if rcParams['axes.hold'] was False. - MAS
- 2006-06-22 Added support for numerix 2-D arrays as alternatives to a sequence of (x,y) tuples for specifying paths in collections, quiver, contour, pcolor, transforms. Fixed contour bug involving setting limits for color mapping. Added numpy-style all() to numerix. - EF
- 2006-06-20 Added custom FigureClass hook to pylab interface - see examples/custom_figure_class.py

- 2006-06-16 Added colormaps from gist (gist_earth, gist_stern, gist_rainbow, gist_gray, gist_yarg, gist_heat, gist_ncar) - JW
 - 2006-06-16 Added a pointer to parent in figure canvas so you can access the container with fig.canvas.manager. Useful if you want to set the window title, e.g., in gtk fig.canvas.manager.window.set_title, though a GUI neutral method would be preferable JDH
 - 2006-06-16 Fixed colorbar.py to handle indexed colors (i.e., norm = no_norm()) by centering each colored region on its index. - EF
 - 2006-06-15 Added scalex and scaley to Axes.autoscale_view to support selective autoscaling just the x or y axis, and supported these command in plot so you can say plot(something, scaley=False) and just the x axis will be autoscaled. Modified axvline and axhline to support this, so for example axvline will no longer autoscale the y axis. JDH
 - 2006-06-13 Fix so numpy updates are backward compatible - TEO
 - 2006-06-12 Updated numerix to handle numpy restructuring of oldnumeric - TEO
 - 2006-06-12 Updated numerix.fft to handle numpy restructuring Added ImportError to numerix.linear_algebra for numpy -TEO
 - 2006-06-11 Added quiverkey command to pylab and Axes, using QuiverKey class in quiver.py. Changed pylab and Axes to use quiver2 if possible, but drop back to the newly-renamed quiver_classic if necessary. Modified examples/quiver_demo.py to illustrate the new quiver and quiverkey. Changed LineCollection implementation slightly to improve compatibility with PolyCollection. - EF
 - 2006-06-11 Fixed a usetex bug for windows, running latex on files with spaces in their names or paths was failing - DSD
 - 2006-06-09 Made additions to numerix, changes to quiver to make it work with all numeric flavors. - EF
 - 2006-06-09 Added quiver2 function to pylab and method to axes, with implementation via a Quiver class in quiver.py. quiver2 will replace quiver before the next release; it is placed alongside it initially to facilitate testing and transition. See also examples/quiver2_demo.py. - EF
 - 2006-06-08 Minor bug fix to make ticker.py draw proper minus signs with usetex - DSD
-
- 2006-06-06 Released 0.87.3 at revision 2432
 - 2006-05-30 More partial support for polygons with outline or fill, but not both. Made LineCollection inherit from ScalarMappable. - EF
 - 2006-05-29 Yet another revision of aspect-ratio handling. - EF
 - 2006-05-27 Committed a patch to prevent stroking zero-width lines in the svg backend - DSD
 - 2006-05-24 Fixed colorbar positioning bug identified by Helge Avlesen, and improved the algorithm; added a 'pad' kwarg to control the spacing between colorbar and parent axes. - EF

- 2006-05-23 Changed color handling so that collection initializers can take any mpl color arg or sequence of args; deprecated float as grayscale, replaced by string representation of float. - EF
- 2006-05-19 Fixed bug: plot failed if all points were masked - EF
- 2006-05-19 Added custom symbol option to scatter - JDH
- 2006-05-18 New example, multi_image.py; colorbar fixed to show offset text when the ScalarFormatter is used; FixedFormatter augmented to accept and display offset text. - EF
- 2006-05-14 New colorbar; old one is renamed to colorbar_classic. New colorbar code is in colorbar.py, with wrappers in figure.py and pylab.py. Fixed aspect-handling bug reported by Michael Mossey. Made backend_bases.draw_quad_mesh() run.- EF
- 2006-05-08 Changed handling of end ranges in contourf: replaced "clip-ends" kwarg with "extend". See docstring for details. -EF
- 2006-05-08 Added axisbelow to rc - JDH
- 2006-05-08 If using PyGTK require version 2.2+ - SC
- 2006-04-19 Added compression support to PDF backend, controlled by new pdf.compression rc setting. - JKS
- 2006-04-19 Added Jouni's PDF backend
- 2006-04-18 Fixed a bug that caused agg to not render long lines
- 2006-04-16 Masked array support for pcolormesh; made pcolormesh support the same combinations of X,Y,C dimensions as pcolor does; improved (I hope) description of grid used in pcolor, pcolormesh. - EF
- 2006-04-14 Reorganized axes.py - EF
- 2006-04-13 Fixed a bug Ryan found using usetex with sans-serif fonts and exponential tick labels - DSD
- 2006-04-11 Refactored backend_ps and backend_agg to prevent module-level texmanager imports. Now these imports only occur if text.usetex rc setting is true - DSD
- 2006-04-10 Committed changes required for building mpl on win32 platforms with visual studio. This allows wxpython blitting for fast animations. - CM
- 2006-04-10 Fixed an off-by-one bug in Axes.change_geometry.
- 2006-04-10 Fixed bug in pie charts where wedge wouldn't have label in legend. Submitted by Simon Hildebrandt. - ADS
- 2006-05-06 Usetex makes temporary latex and dvi files in a temporary directory, rather than in the user's current working directory - DSD
- 2006-04-05 Applied Ken's wx deprecation warning patch closing sf patch #1465371 - JDH
- 2006-04-05 Added support for the new API in the postscript backend. Allows values to be masked using nan's, and faster file creation - DSD
- 2006-04-05 Use python's subprocess module for usetex calls to external programs. subprocess catches when they exit abnormally so an error can be raised. - DSD

- 2006-04-03 Fixed the bug in which widgets would not respond to events. This regressed the twinx functionality, so I also updated `subplots_adjust` to update axes that share an x or y with a subplot instance. - CM
- 2006-04-02 Moved PBox class to transforms and deleted `pbox.py`; made `pylab axis` command a thin wrapper for `Axes.axis`; more tweaks to aspect-ratio handling; fixed `Axes.specgram` to account for the new `imshow` default of unit aspect ratio; made `contour` set the `Axes.dataLim`. - EF
- 2006-03-31 Fixed the Qt "Underlying C/C++ object deleted" bug. - JRE
- 2006-03-31 Applied Vasily Sulatskov's Qt Navigation Toolbar enhancement. - JRE
- 2006-03-31 Ported Norbert's rewriting of Halldor's `stineman_interp` algorithm to make it numerix compatible and added code to `matplotlib.mlab`. See `examples/interp_demo.py` - JDH
- 2006-03-30 Fixed a bug in aspect ratio handling; blocked potential crashes when panning with button 3; added `axis('image')` support. - EF
- 2006-03-28 More changes to aspect ratio handling; new PBox class in new file `pbox.py` to facilitate resizing and repositioning axes; made `PolarAxes` maintain unit aspect ratio. - EF
- 2006-03-23 Refactored `TextWithDash` class to inherit from, rather than delegate to, the `Text` class. Improves object inspection and closes bug # 1357969 - DSD
- 2006-03-22 Improved aspect ratio handling, including `pylab` interface. Interactive resizing, pan, zoom of images and plots (including panels with a shared axis) should work. Additions and possible refactoring are still likely. - EF
- 2006-03-21 Added another colorbrewer colormap (`RdYlBu`) - JSWHIT
- 2006-03-21 Fixed tickmarks for logscale plots over very large ranges. Closes bug # 1232920 - DSD
- 2006-03-21 Added Rob Knight's arrow code; see `examples/arrow_demo.py` - JDH
- 2006-03-20 Added support for masking values with nan's, using ADS's `isnan` module and the new API. Works for *Agg backends - DSD
- 2006-03-20 Added `contour.negative_linestyle` rcParam - ADS
- 2006-03-20 Added `_isnan` extension module to test for nan with Numeric
- ADS
- 2006-03-17 Added Paul and Alex's support for faceting with `quadmsh` in sf patch 1411223 - JDH
- 2006-03-17 Added Charle Twardy's pie patch to support `colors=None`. Closes sf patch 1387861 - JDH
- 2006-03-17 Applied sophana's patch to support overlapping axes with toolbar navigation by toggling activation with the 'a' key. Closes sf patch 1432252 - JDH
- 2006-03-17 Applied Aarre's linestyle patch for backend EMF; closes sf patch 1449279 - JDH
- 2006-03-17 Applied Jordan Dawe's patch to support `kwargs` properties for grid lines in the `grid` command. Closes sf patch 1451661 - JDH

2006-03-17 Center postscript output on page when using usetex - DSD

2006-03-17 subprocess module built if Python <2.4 even if subprocess can be imported from an egg - ADS

2006-03-17 Added __subprocess.c from Python upstream and hopefully enabled building (without breaking) on Windows, although not tested. - ADS

2006-03-17 Updated subprocess.py to latest Python upstream and reverted name back to subprocess.py - ADS

2006-03-16 Added John Porter's 3D handling code

2006-03-16 Released 0.87.2 at revision 2150

2006-03-15 Fixed bug in MaxNLocator revealed by daigos@infinito.it. The main change is that Locator.nonsingular now adjusts vmin and vmax if they are nearly the same, not just if they are equal. A new kwarg, "tiny", sets the threshold. - EF

2006-03-14 Added import of compatibility library for newer numpy linear_algebra - TEO

2006-03-12 Extended "load" function to support individual columns and moved "load" and "save" into matplotlib.mlab so they can be used outside of pylab - see examples/load_converter.py - JDH

2006-03-12 Added AutoDateFormatter and AutoDateLocator submitted by James Evans. Try the load_converter.py example for a demo. - ADS

2006-03-11 Added subprocess module from python-2.4 - DSD

2006-03-11 Fixed landscape orientation support with the usetex option. The backend_ps print_figure method was getting complicated, I added a __print_figure_tex method to maintain some degree of sanity - DSD

2006-03-11 Added "papertype" savefig kwarg for setting postscript papersizes. papertype and ps.papersize rc setting can also be set to "auto" to autoscale pagesizes - DSD

2006-03-09 Apply P-J's patch to make pstoeeps work on windows patch report # 1445612 - DSD

2006-03-09 Make backend rc parameter case-insensitive - DSD

2006-03-07 Fixed bug in backend_ps related to C0-C6 papersizes, which were causing problems with postscript viewers. Supported page sizes include letter, legal, ledger, A0-A10, and B0-B10 - DSD

2006-03-07 Released 0.87.1

2006-03-04 backend_cairo.py: fix get_rgb() bug reported by Keith Briggs. Require pycairo 1.0.2. Support saving png to file-like objects. - SC

2006-03-03 Fixed pcolor handling of vmin, vmax - EF

2006-03-02 improve page sizing with usetex with the latex geometry package. Closes bug # 1441629 - DSD

- 2006-03-02 Fixed dpi problem with usetex png output. Accepted a modified version of patch #1441809 - DSD
- 2006-03-01 Fixed axis('scaled') to deal with case xmax < xmin - JSWHIT
- 2006-03-01 Added reversed colormaps (with '_r' appended to name) - JSWHIT
- 2006-02-27 Improved eps bounding boxes with usetex - DSD
- 2006-02-27 Test svn commit, again!
- 2006-02-27 Fixed two dependency checking bugs related to usetex on Windows - DSD
- 2006-02-27 Made the rc deprecation warnings a little more human readable.
- 2006-02-26 Update the previous gtk.main_quit() bug fix to use gtk.main_level()
- SC
- 2006-02-24 Implemented alpha support in contour and contourf - EF
- 2006-02-22 Fixed gtk main quit bug when quit was called before mainloop. - JDH
- 2006-02-22 Small change to colors.py to workaround apparent bug in numpy masked array module - JSWHIT
- 2006-02-22 Fixed bug in ScalarMappable.to_rgba() reported by Ray Jones, and fixed incorrect fix found by Jeff Whitaker - EF
-
- 2006-02-22 Released 0.87
- 2006-02-21 Fixed portrait/landscape orientation in postscript backend - DSD
- 2006-02-21 Fix bug introduced in yesterday's bug fix - SC
- 2006-02-20 backend_gtk.py FigureCanvasGTK.draw(): fix bug reported by David Tremouilles - SC
- 2006-02-20 Remove the "pygtk.require('2.4')" error from examples/embedding_in_gtk2.py - SC
- 2006-02-18 backend_gtk.py FigureCanvasGTK.draw(): simplify to use (rather than duplicate) the expose_event() drawing code - SC
- 2006-02-12 Added stagger or waterfall plot capability to LineCollection; illustrated in examples/collections.py. - EF
- 2006-02-11 Massive cleanup of the usetex code in the postscript backend. Possibly fixed the clipping issue users were reporting with older versions of ghostscript - DSD
- 2006-02-11 Added autolim kwarg to axes.add_collection. Changed collection get_verts() methods accordingly. - EF
- 2006-02-09 added a temporary rc parameter text.dvipnghack, to allow Mac users to get nice results with the usetex option. - DSD
- 2006-02-09 Fixed a bug related to setting font sizes with the usetex option. - DSD
- 2006-02-09 Fixed a bug related to usetex's latex code. - DSD

- 2006-02-09 Modified behavior of font.size rc setting. You should define font.size in pts, which will set the "medium" or default fontsize. Special text sizes like axis labels or tick labels can be given relative font sizes like small, large, x-large, etc. and will scale accordingly. - DSD
- 2006-02-08 Added py2exe specific datapath check again. Also added new py2exe helper function get_py2exe_datafiles for use in py2exe setup.py scripts. - CM
- 2006-02-02 Added box function to pylab
- 2006-02-02 Fixed a problem in setupext.py, tk library formatted in unicode caused build problems - DSD
- 2006-02-01 Dropped TeX engine support in usetex to focus on LaTeX. - DSD
- 2006-01-29 Improved usetex option to respect the serif, sans-serif, monospace, and cursive rc settings. Removed the font.latex.package rc setting, it is no longer required - DSD
- 2006-01-29 Fixed tex's caching to include font.family rc information - DSD
- 2006-01-29 Fixed subpixel rendering bug in *Agg that was causing uneven gridlines - JDH
- 2006-01-28 Added fontcmd to backend_ps's RendererPS.draw_tex, to support other font families in eps output - DSD
- 2006-01-28 Added MaxNLocator to ticker.py, and changed contour.py to use it by default. - EF
- 2006-01-28 Added fontcmd to backend_ps's RendererPS.draw_tex, to support other font families in eps output - DSD
- 2006-01-27 Buffered reading of matplotlibrc parameters in order to allow 'verbose' settings to be processed first (allows verbose.report during rc validation process) - DSD
- 2006-01-27 Removed setuptools support from setup.py and created a separate setupegg.py file to replace it. - CM
- 2006-01-26 Replaced the ugly datapath logic with a cleaner approach from <http://wiki.python.org/moin/DistutilsInstallDataScattered>. Overrides the install_data command. - CM
- 2006-01-24 Don't use character typecodes in cntr.c — changed to use defined typenumbers instead. - TEO
- 2006-01-24 Fixed some bugs in usetex's and ps.usedistiller's dependency
- 2006-01-24 Added masked array support to scatter - EF
- 2006-01-24 Fixed some bugs in usetex's and ps.usedistiller's dependency checking - DSD
-
- 2006-01-24 Released 0.86.2
- 2006-01-20 Added a converters dict to pylab load to convert selected columns to float — especially useful for files with date strings, uses a datestr2num converter - JDH
- 2006-01-20 Added datestr2num to matplotlib dates to convert a string or sequence of strings to a matplotlib datetime
- 2006-01-18 Added quadrilateral pcolormesh patch 1409190 by Alex Mont and Paul Kienzle — this is *Agg only for now. See examples/quadmesh_demo.py - JDH

2006-01-18 Added Jouni's boxplot patch - JDH

2006-01-18 Added comma delimiter for pylab save - JDH

2006-01-12 Added Ryan's legend patch - JDH

2006-1-12 Fixed numpy / numeric to use .dtype.char to keep in SYNC with numpy SVN

2006-1-11 Released 0.86.1

2006-1-11 Fixed setup.py for win32 build and added rc template to the MANIFEST.in

2006-1-10 Added xpdf distiller option. matplotlibrc ps.usedistiller can now be none, false, ghostscript, or xpdf. Validation checks for dependencies. This needs testing, but the xpdf option should produce the highest-quality output and small file sizes - DSD

2006-01-10 For the usetex option, backend_ps now does all the LaTeX work in the os's temp directory - DSD

2006-1-10 Added checks for usetex dependencies. - DSD

2006-1-9 Released 0.86

2006-1-4 Changed to support numpy (new name for scipy_core) - TEO

2006-1-4 Added Mark's scaled axes patch for shared axis

2005-12-28 Added Chris Barker's build_wxagg patch - JDH

2005-12-27 Altered numerix/scipy to support new scipy package structure - TEO

2005-12-20 Fixed Jame's Boyles date tick reversal problem - JDH

2005-12-20 Added Jouni's rc patch to support lists of keys to set on - JDH

2005-12-12 Updated pyparsing and mathtext for some speed enhancements (Thanks Paul McGuire) and minor fixes to scipy numerix and setuptools

2005-12-12 Matplotlib data is now installed as package_data in the matplotlib module. This gets rid of checking the many possibilities in matplotlib._get_data_path() - CM

2005-12-11 Support for setuptools/pkg_resources to build and use matplotlib as an egg. Still allows matplotlib to exist using a traditional distutils install. - ADS

2005-12-03 Modified setup to build matplotlibrc based on compile time findings. It will set numerix in the order of scipy, numarray, Numeric depending on which are founds, and backend as in preference order GTKAgg, WXAgg, TkAgg, GTK, Agg, PS

2005-12-03 Modified scipy patch to support Numeric, scipy and numarray Some work remains to be done because some of the scipy imports are broken if only the core is installed. e.g., apparently we need from scipy.basic.fftpack import * rather than from scipy.fftpack import *

2005-12-03 Applied some fixes to Nicholas Young's nonuniform image patch

2005-12-01 Applied Alex Gontmakher hatch patch - PS only for now

2005-11-30 Added Rob McMullen's EMF patch

2005-11-30 Added Daishi's patch for scipy

2005-11-30 Fixed out of bounds draw markers segfault in agg

2005-11-28 Got TkAgg blitting working 100% (cross fingers) correctly. - CM

2005-11-27 Multiple changes in `cm.py`, `colors.py`, `figure.py`, `image.py`, `contour.py`, `contour_demo.py`; new `_cm.py`, `examples/image_masked.py`. 1) Separated the color table data from `cm.py` out into a new file, `_cm.py`, to make it easier to find the actual code in `cm.py` and to add new colormaps. Also added some line breaks to the color data dictionaries. Everything from `_cm.py` is imported by `cm.py`, so the split should be transparent. 2) Enabled automatic generation of a colormap from a list of colors in `contour`; see modified `examples/contour_demo.py`. 3) Support for `imshow` of a masked array, with the ability to specify colors (or no color at all) for masked regions, and for regions that are above or below the normally mapped region. See `examples/image_masked.py`. 4) In support of the above, added two new classes, `ListedColormap`, and `no_norm`, to `colors.py`, and modified the `Colormap` class to include common functionality. Added a `clip` kwarg to the `normalize` class. Reworked color handling in `contour.py`, especially in the `ContourLabeller` mixin. - EF

2005-11-25 Changed `text.py` to ensure color is hashable. EF

2005-11-16 Released 0.85

2005-11-16 Changed the default default linewidth in `rc` to 1.0

2005-11-16 Replaced `agg_to_gtk_drawable` with pure pygtk pixbuf code in `backend_gtkagg`. When the equivalent is done for blit, the agg extension code will no longer be needed

2005-11-16 Added a `maxdict` item to `cbook` to prevent caches from growing w/o bounds

2005-11-15 Fixed a `colorup/colordown` reversal bug in `finance.py` – Thanks Gilles

2005-11-15 Applied Jouni K Steppanen's boxplot patch SF patch#1349997

- JDH

2005-11-09 added `axisbelow` attr for `Axes` to determine whether ticks and such are above or below the actors

2005-11-08 Added Nicolas' irregularly spaced image patch

2005-11-08 Deprecated `HorizontalSpanSelector` and replaced with `SpanSelection` that takes a third arg, `direction`. The new `SpanSelector` supports horizontal and vertical span selection, and the appropriate min/max is returned. - CM

2005-11-08 Added `lineprops` dialog for `gtk`

2005-11-03 Added `FIFOBuffer` class to `mlab` to support real time feeds and `examples/fifo_buffer.py`

2005-11-01 Contributed Nickolas Young's patch for `afm mathtext` to support `mathtext` based upon the standard postscript Symbol font when `ps.usetex = True`.

- 2005-10-26 Added support for scatter legends - thanks John Gill
- 2005-10-20 Fixed image clipping bug that made some tex labels disappear. JDH
- 2005-10-14 Removed sqrt from dvipng 1.6 alpha channel mask.
- 2005-10-14 Added width kwarg to hist function
- 2005-10-10 Replaced all instances of os.rename with shutil.move
- 2005-10-05 Added Michael Brady's ydate patch
- 2005-10-04 Added rkern's texmanager patch
- 2005-09-25 contour.py modified to use a single ContourSet class that handles filled contours, line contours, and labels; added keyword arg (clip_ends) to contourf. Colorbar modified to work with new ContourSet object; if the ContourSet has lines rather than polygons, the colorbar will follow suit. Fixed a bug introduced in 0.84, in which contourf(...,colors=...) was broken - EF
-
- 2005-09-19 Released 0.84
- 2005-09-14 Added a new 'resize_event' which triggers a callback with a backend_bases.ResizeEvent object - JDH
- 2005-09-14 font_manager.py: removed chkfontpath from x11FontDirectory() - SC
- 2005-09-14 Factored out auto date locator/formatter factory code into matplotlib.date.date_ticker_factory; applies John Bryne's quiver patch.
- 2005-09-13 Added Mark's axes positions history patch #1286915
- 2005-09-09 Added support for auto canvas resizing with fig.set_figsize_inches(9,5,forward=True) # inches OR fig.resize(400,300) # pixels
- 2005-09-07 figure.py: update Figure.draw() to use the updated renderer.draw_image() so that examples/figimage_demo.py works again. examples/stock_demo.py: remove data_clipping (which no longer exists) - SC
- 2005-09-06 Added Eric's tick.direction patch: in or out in rc
- 2005-09-06 Added Martin's rectangle selector widget
- 2005-09-04 Fixed a logic err in text.py that was preventing rgxsuper from matching - JDH
- 2005-08-29 Committed Ken's wx blit patch #1275002
- 2005-08-26 colorbar modifications - now uses contourf instead of imshow so that colors used by contourf are displayed correctly. Added two new keyword args (cspacing and clabels) that are only relevant for ContourMappable images - JSWHIT
- 2005-08-24 Fixed a PS image bug reported by Darren - JDH
- 2005-08-23 colors.py: change hex2color() to accept unicode strings as well as normal strings. Use isinstance() instead of types.IntType etc - SC
- 2005-08-16 removed data_clipping line and rc property - JDH

2005-08-22 backend_svg.py: Remove redundant "x=0.0 y=0.0" from svg element. Increase svg version from 1.0 to 1.1. Add viewBox attribute to svg element to allow SVG documents to scale-to-fit into an arbitrary viewport - SC

2005-08-16 Added Eric's dot marker patch - JDH

2005-08-08 Added blitting/animation for TkAgg - CM

2005-08-05 Fixed duplicate tickline bug - JDH

2005-08-05 Fixed a GTK animation bug that cropped up when doing animations in gtk//gtkagg canvases that had widgets packed above them

2005-08-05 Added Clovis Goldemberg patch to the tk save dialog

2005-08-04 Removed origin kwarg from backend.draw_image. origin is handled entirely by the frontend now.

2005-07-03 Fixed a bug related to TeX commands in backend_ps

2005-08-03 Fixed SVG images to respect upper and lower origins.

2005-08-03 Added flipud method to image and removed it from to_str.

2005-07-29 Modified figure.figspect to take an array or number; modified backend_svg to write utf-8 - JDH

2005-07-30 backend_svg.py: embed png image files in svg rather than linking to a separate png file, fixes bug #1245306 (thanks to Norbert Nemec for the patch) - SC

2005-07-29 Released 0.83.2

2005-07-27 Applied SF patch 1242648: minor rounding error in IndexDateFormatter in dates.py

2005-07-27 Applied sf patch 1244732: Scale axis such that circle looks like circle - JDH

2005-07-29 Improved message reporting in texmanager and backend_ps - DSD

2005-07-28 backend_gtk.py: update FigureCanvasGTK.draw() (needed due to the recent expose_event() change) so that examples/anim.py works in the usual way - SC

2005-07-26 Added new widgets Cursor and HorizontalSpanSelector to matplotlib.widgets. See examples/widgets/cursor.py and examples/widgets/span_selector.py - JDH

2005-07-26 added draw event to mpl event hierarchy – triggered on figure.draw

2005-07-26 backend_gtk.py: allow 'f' key to toggle window fullscreen mode

2005-07-26 backend_svg.py: write "<.../>" elements all on one line and remove surplus spaces - SC

2005-07-25 backend_svg.py: simplify code by deleting GraphicsContextSVG and RendererSVG.new_gc(), and moving the gc.get_capstyle() code into RendererSVG._get_gc_props_svg() - SC

- 2005-07-24 backend_gtk.py: call FigureCanvasBase.motion_notify_event() on all motion-notify-events, not just ones where a modifier key or button has been pressed (fixes bug report from Niklas Volbers) - SC
- 2005-07-24 backend_gtk.py: modify print_figure() use own pixmap, fixing problems where print_figure() overwrites the display pixmap. return False from all button/key etc events - to allow the event to propagate further - SC
- 2005-07-23 backend_gtk.py: change expose_event from using set_back_pixmap(); clear() to draw_drawable() - SC
- 2005-07-23 backend_gtk.py: removed pygtk.require() matplotlib/___init___.py: delete 'FROZEN' and 'McPLError' which are no longer used - SC
- 2005-07-22 backend_gdk.py: removed pygtk.require() - SC
- 2005-07-21 backend_svg.py: Remove unused imports. Remove methods doc strings which just duplicate the docs from backend_bases.py. Rename draw_mathtext to _draw_mathtext. - SC
- 2005-07-17 examples/embedding_in_gtk3.py: new example demonstrating placing a FigureCanvas in a gtk.ScrolledWindow - SC
- 2005-07-14 Fixed a Windows related bug (#1238412) in texmanager - DSD
- 2005-07-11 Fixed color kwarg bug, setting color=1 or 0 caused an exception - DSD
- 2005-07-07 Added Eric's MA set_xdata Line2D fix - JDH
- 2005-07-06 Made HOME/.matplotlib the new config dir where the matplotlibrc file, the ttf.cache, and the tex.cache live. The new default filenames in .matplotlib have no leading dot and are not hidden. e.g., the new names are matplotlibrc tex.cache ttfont.cache. This is how ipython does it so it must be right. If old files are found, a warning is issued and they are moved to the new location. Also fixed texmanager to put all files, including temp files in ~/.matplotlib/tex.cache, which allows you to usetex in non-writable dirs.
- 2005-07-05 Fixed bug #1231611 in subplots adjust layout. The problem was that the text cacheing mechanism was not using the transformation affine in the key. - JDH
- 2005-07-05 Fixed default backend import problem when using API (SF bug # 1209354 - see API_CHANGES for more info - JDH
- 2005-07-04 backend_gtk.py: require PyGTK version 2.0.0 or higher - SC
- 2005-06-30 setupext.py: added numarray_inc_dirs for building against numarray when not installed in standard location - ADS
- 2005-06-27 backend_svg.py: write figure width, height as int, not float. Update to fix some of the pychecker warnings - SC
- 2005-06-23 Updated examples/agg_test.py to demonstrate curved paths and fills - JDH
- 2005-06-21 Moved some texmanager and backend_agg tex caching to class level rather than instance level - JDH
- 2005-06-20 setupext.py: fix problem where _nc_backend_gdk is installed to the wrong directory - SC

2005-06-19 Added 10.4 support for CocoaAgg. - CM

2005-06-18 Move `Figure.get_width_height()` to `FigureCanvasBase` and return `int` instead of `float`. - SC

2005-06-18 Applied Ted Drain's QtAgg patch: 1) Changed the toolbar to be a horizontal bar of push buttons instead of a `QToolbar` and updated the layout algorithms in the main window accordingly. This eliminates the ability to drag and drop the toolbar and detach it from the window. 2) Updated the resize algorithm in the main window to show the correct size for the plot widget as requested. This works almost correctly right now. It looks to me like the final size of the widget is off by the border of the main window but I haven't figured out a way to get that information yet. We could just add a small margin to the new size but that seems a little hacky. 3) Changed the x/y location label to be in the toolbar like the Tk backend instead of as a status line at the bottom of the widget. 4) Changed the toolbar pixmaps to use the ppm files instead of the png files. I noticed that the Tk backend buttons looked much nicer and it uses the ppm files so I switched them.

2005-06-17 Modified the gtk backend to not queue mouse motion events. This allows for live updates when dragging a slider. - CM

2005-06-17 Added starter CocoaAgg backend. Only works on OS 10.3 for now and requires Py-ObjC. (10.4 is high priority) - CM

2005-06-17 Upgraded pyparsing and applied Paul McGuire's suggestions for speeding things up. This more than doubles the speed of `mathtext` in my simple tests. JDH

2005-06-16 Applied David Cooke's subplot `make_key` patch

2005-06-15 0.82 released

2005-06-15 Added subplot config tool to GTK* backends – note you must now import the `NavigationToolbar2` from your backend of choice rather than from `backend_gtk` because it needs to know about the backend specific canvas – see `examples/embedding_in_gtk2.py`. Ditto for wx backend – see `examples/embedding_in_wxagg.py`

2005-06-15 `backend_cairo.py`: updated to use `pycairo` 0.5.0 - SC

2005-06-14 Wrote some GUI neutral widgets (`Button`, `Slider`, `RadioButtons`, `CheckButtons`) in `matplotlib.widgets`. See `examples/widgets/*.py` - JDH

2005-06-14 Exposed subplot parameters as rc vars and as the `fig.SubplotParams` instance `subplotspars`. See `figure.SubplotParams`, `figure.Figure.subplots_adjust` and the `pylab` method `subplots_adjust` and `examples/subplots_adjust.py`. Also added a GUI neutral widget for adjusting subplots, see `examples/subplot_toolbar.py` - JDH

2005-06-13 Exposed `cap` and `join` style for lines with new rc params and

line properties

```
lines.dash_joinstyle : miter # miter|round|bevel lines.dash_capstyle : butt #  
butt|round|projecting lines.solid_joinstyle : miter # miter|round|bevel lines.solid_capstyle  
: projecting # butt|round|projecting
```

2005-06-13 Added kwargs to Axes init

2005-06-13 Applied Baptiste's tick patch - JDH

2005-06-13 Fixed rc alias 'l' bug reported by Fernando by removing aliases for mainlevel rc options. - JDH

2005-06-10 Fixed bug #1217637 in ticker.py - DSD

2005-06-07 Fixed a bug in texmanager.py: .aux files not being removed - DSD

2005-06-08 Added Sean Richard's hist binning fix – see API_CHANGES - JDH

2005-06-07 Fixed a bug in texmanager.py: .aux files not being removed

- DSD
-

2005-06-07 matplotlib-0.81 released

2005-06-06 Added autoscale_on prop to axes

2005-06-06 Added Nick's picker "among" patch - JDH

2005-06-05 Fixed a TeX/LaTeX font discrepancy in backend_ps. - DSD

2005-06-05 Added a ps.distill option in rc settings. If True, postscript output will be distilled using ghostscript, which should trim the file size and allow it to load more quickly. Hopefully this will address the issue of large ps files due to font definitions. Tested with gnu-ghostscript-8.16. - DSD

2005-06-03 Improved support for tex handling of text in backend_ps. - DSD

2005-06-03 Added rc options to render text with tex or latex, and to select the latex font package. - DSD

2005-06-03 Fixed a bug in ticker.py causing a ZeroDivisionError

2005-06-02 backend_gtk.py remove DBL_BUFFER, add line to expose_event to try to fix pygtk 2.6 redraw problem - SC

2005-06-01 The default behavior of ScalarFormatter now renders scientific notation and large numerical offsets in a label at the end of the axis. - DSD

2005-06-01 Added Nicholas' frombyte image patch - JDH

2005-05-31 Added vertical TeX support for agg - JDH

2005-05-31 Applied Eric's cntr patch - JDH

2005-05-27 Finally found the pesky agg bug (which Maxim was kind enough to fix within hours) that was causing a segfault in the win32 cached marker drawing. Now windows users can get the enormous performance benefits of caced markers w/o those occasional pesy screenshots. - JDH

2005-05-27 Got win32 build system working again, using a more recent version of gtk and pygtk in the win32 build, gtk 2.6 from <http://www.gimp.org/~tml/gimp/win32/downloads.html> (you will also need libpng12.dll to use these). I haven't tested whether this binary build of mpl for win32 will work with older gtk runtimes, so you may need to upgrade.

- 2005-05-27 Fixed bug where 2nd wxagg could be started if using wxagg backend. - ADS
- 2005-05-26 Added Daishi text with dash patch – see `examples/dashtick.py`
- 2005-05-26 Moved `backend_latex` functionality into `backend_ps`. If `text.usetex=True`, the PostScript backend will use LaTeX to generate the .ps or .eps file. Ghostscript is required for eps output. - DSD
- 2005-05-24 Fixed alignment and color issues in latex backend. - DSD
- 2005-05-21 Fixed raster problem for small rasters with `dvipng` – looks like it was a premultiplied alpha problem - JDH
- 2005-05-20 Added `linewidth` and `faceted` kwarg to `scatter` to control `edgewidth` and `color`. Also added `autolegend` patch to inspect line segments.
- 2005-05-18 Added Orsay and JPL qt fixes - JDH
- 2005-05-17 Added a `psfrag` latex backend – some alignment issues need to be worked out. Run with `-dLaTeX` and a .tex file and *.eps file are generated. `latex` and `dvips` the generated latex file to get ps output. Note `xdvi` *does not work, you must generate ps.- JDH
- 2005-05-13 Added Florent Rougon's `Axis` `set_label1` patch
- 2005-05-17 `pcolor` optimization, fixed bug in previous `pcolor` patch - JSWHIT
- 2005-05-16 Added support for masked arrays in `pcolor` - JSWHIT
- 2005-05-12 Started work on TeX text for `antigrain` using `pngdvi` – see `examples/tex_demo.py` and the new module `matplotlib.texmanager`. Rotated text not supported and rendering small glyphs is not working right yet. BUT large font sizes and/or high dpi saved figs work great.
- 2005-05-10 New image resize options interpolation options. New values for the `interp` kwarg are `'nearest'`, `'bilinear'`, `'bicubic'`, `'spline16'`, `'spline36'`, `'hanning'`, `'hamming'`, `'hermite'`, `'kaiser'`, `'quadric'`, `'catrom'`, `'gaussian'`, `'bessel'`, `'mitchell'`, `'sinc'`, `'lanczos'`, `'blackman'`
- See `help(imshow)` for details, particularly the interpolation, `filtnorm` and `filterrad` kwargs
- 2005-05-10 Applied Eric's contour mem leak fixes - JDH
- 2005-05-10 Extended python agg wrapper and started implementing `backend_agg2`, an agg renderer based on the python wrapper. This will be more flexible and easier to extend than the current `backend_agg`. See also `examples/agg_test.py` - JDH
- 2005-05-09 Added Marcin's no legend patch to exclude lines from the `autolegend` builder
- `plot(x, y, label='nolegend')`
- 2005-05-05 Upgraded to agg23
- 2005-05-05 Added `newscalarformatter_demo.py` to `examples`. -DSD
- 2005-05-04 Added `NewScalarFormatter`. Improved formatting of tick labels, scientific notation, and the ability to plot large large numbers with small ranges, by determining a numerical offset. See `ticker.NewScalarFormatter` for more details. -DSD

- 2005-05-03 Added the option to specify a delimiter in `pylab.load` - DSD
- 2005-04-28 Added Darren's line collection example
- 2005-04-28 Fixed `aa` property in `agg` - JDH
- 2005-04-27 Set postscript page size in `.matplotlibrc` - DSD
- 2005-04-26 Added embedding in `qt` example. - JDH
- 2005-04-14 Applied Michael Brady's `qt` backend patch: 1) fix a bug where keyboard input was grabbed by the figure and not released 2) turn on cursor changes 3) clean up a typo and commented-out print statement. - JDH
- 2005-04-14 Applied Eric Firing's masked data lines patch and contour patch. Support for masked arrays has been added to the `plot` command and to the `Line2D` object. Only the valid points are plotted. A "valid_only" kwarg was added to the `get_xdata()` and `get_ydata()` methods of `Line2D`; by default it is `False`, so that the original data arrays are returned. Setting it to `True` returns the plottable points. - see `examples/masked_demo.py` - JDH
- 2005-04-13 Applied Tim Leslie's arrow key event handling patch - JDH
-
- 0.80 released
- 2005-04-11 Applied a variant of rick's `xlim/ylim/axis` patch. These functions now take kwargs to let you selectively alter only the min or max if desired. e.g., `xlim(xmin=2)` or `axis(ymax=3)`. They always return the new lim. - JDH
- 2005-04-11 Incorporated Werner's `wx` patch – `wx` backend should be compatible with `wxpython2.4` and recent versions of 2.5. Some early versions of `wxpython 2.5` will not work because there was a temporary change in the `dc` API that was rolled back to make it 2.4 compliant
- 2005-04-11 modified `tkagg` show so that new figure window pops up on call to `figure`
- 2005-04-11 fixed `wxapp` init bug
- 2005-04-02 updated `backend_ps.draw_lines`, `draw_markers` for use with the new API - DSD
- 2005-04-01 Added editable polygon example
-

- 2005-03-31 0.74 released
- 2005-03-30 Fixed and added checks for floating point inaccuracy in `ticker.Base` - DSD
- 2005-03-30 updated `/ellipse` definition in `backend_ps.py` to address bug #1122041 - DSD
- 2005-03-29 Added unicode support for `Agg` and `PS` - JDH
- 2005-03-28 Added Jarrod's `svg` patch for text - JDH
- 2005-03-28 Added Ludal's arrow and quiver patch - JDH
- 2005-03-28 Added `label` kwarg to `Axes` to facilitate forcing the creation of new `Axes` with otherwise identical attributes
-

2005-03-28 Applied boxplot and OSX font search patches

2005-03-27 Added ft2font NULL check to fix Japanese font bug - JDH

2005-03-27 Added sprint legend patch plus John Gill's tests and fix – see [examples/legend_auto.py](#) - JDH

2005-03-19 0.73.1 released

2005-03-19 Reverted wxapp handling because it crashed win32 - JDH

2005-03-18 Add .number attribute to figure objects returned by figure() - FP

2005-03-18 0.73 released

2005-03-16 Fixed labelsep bug

2005-03-16 Applied Darren's ticker fix for small ranges - JDH

2005-03-16 Fixed tick on horiz colorbar - JDH

2005-03-16 Added Japanese winreg patch - JDH

2005-03-15 backend_gtkagg.py: changed to use double buffering, this fixes the problem reported Joachim Berdal Haga - "Parts of plot lagging from previous frame in animation". Tested with anim.py and it makes no noticeable difference to performance (23.7 before, 23.6 after) - SC

2005-03-14 add src/_backend_gdk.c extension to provide a substitute function for pixbuf.get_pixels_array(). Currently pixbuf.get_pixels_array() only works with Numeric, and then only works if pygtk has been compiled with Numeric support. The change provides a function pixbuf_get_pixels_array() which works with Numeric and numarray and is always available. It means that backend_gtk should be able to display images and mathtext in all circumstances. - SC

2005-03-11 Upgraded CXX to 5.3.1

2005-03-10 remove GraphicsContextPS.set_linestyle() and GraphicsContextSVG.set_linestyle() since they do no more than the base class GraphicsContext.set_linestyle() - SC

2005-03-09 Refactored contour functionality into dedicated module

2005-03-09 Added Eric's contourf updates and Nadia's clabel functionality

2005-03-09 Moved colorbar to figure.Figure to expose it for API developers

- JDH

2005-03-09 backend_cairo.py: implemented draw_markers() - SC

2005-03-09 cbook.py: only use enumerate() (the python version) if the builtin

version is not available. Add new function 'izip' which is set to itertools.izip if available and the python equivalent if not available. - SC

2005-03-07 backend_gdk.py: remove PIXELS_PER_INCH from points_to_pixels(), but

still use it to adjust font sizes. This allows the GTK version of `line_styles.py` to more closely match `GTKAgg`, previously the markers were being drawn too large. - SC

2005-03-01 Added Eric's `contourf` routines

2005-03-01 Added start of proper `agg` SWIG wrapper. I would like to expose `agg` functionality directly at the user level and this module will serve that purpose eventually, and will hopefully take over most of the functionality of the current `_image` and `_backend_agg` modules. - JDH

2005-02-28 Fixed `polyfit` / `polyval` to convert input args to float arrays - JDH

2005-02-25 Add experimental feature to `backend_gtk.py` to enable/disable double buffering (`DBL_BUFFER=True/False`) - SC

2005-02-24 `colors.py` change `ColorConverter.to_rgb()` so it always returns `rgb` (and not `rgba`), allow `cnames` keys to be cached, change the exception raised from `RuntimeError` to `ValueError` (like `hex2color()`) `hex2color()` use a regular expression to check the color string is valid - SC

2005-02-23 Added `rc` param `ps.useafm` so backend `ps` can use native `afm` fonts or `truetype`. `afm` breaks `mathtext` but causes much smaller font sizes and may result in images that display better in some contexts (e.g., pdfs incorporated into latex docs viewed in acrobat reader). I would like to extend this approach to allow the user to use `truetype` only for `mathtext`, which should be easy.

2005-02-23 Used sequence protocol rather than tuple in `agg` collection drawing routines for greater flexibility - JDH

2005-02-22 0.72.1 released

2005-02-21 fixed `linestyles` for collections – contour now dashes for levels < 0

2005-02-21 fixed `ps` color bug - JDH

2005-02-15 fixed missing `qt` file

2005-02-15 banished `error_msg` and `report_error`. Internal backend methods like `error_msg_gtk` are preserved. backend writers, check your backends, and diff against 0.72 to make sure I did the right thing! - JDH

2005-02-14 Added `enthought` traits to `matplotlib` tree - JDH

2005-02-14 0.72 released

2005-02-14 fix bug in `cbook.alltrue()` and `onetrue()` - SC

2005-02-11 updated `qtagg` backend from Ted - JDH

2005-02-11 `matshow` fixes for figure numbering, return value and docs - FP

2005-02-09 new `zorder` example for fine control in `zorder_demo.py` - FP

2005-02-09 backend `renderer` `draw_lines` now has `transform` in backend, as in `draw_markers`; use `numerix` in `_backend_agg`, added small line optimization to `agg`

2005-02-09 `subplot` now deletes axes that it overlaps

- 2005-02-08 Added transparent support for gzipped files in load/save - Fernando Perez (FP from now on).
- 2005-02-08 Small optimizations in PS backend. They may have a big impact for large plots, otherwise they don't hurt - FP
- 2005-02-08 Added transparent support for gzipped files in load/save - Fernando Perez (FP from now on).
- 2005-02-07 Added newstyle path drawing for markers - only implemented in agg currently - JDH
- 2005-02-05 Some superscript text optimizations for ticking log plots
- 2005-02-05 Added some default key press events to pylab figures: 'g' toggles grid - JDH
- 2005-02-05 Added some support for handling log switching for lines that have nonpos data - JDH
- 2005-02-04 Added Nadia's contour patch - contour now has matlab compatible syntax; this also fixed an unequal sized contour array bug- JDH
- 2005-02-04 Modified GTK backends to allow the FigureCanvas to be resized smaller than its original size - SC
- 2005-02-02 Fixed a bug in dates mx2num - JDH
- 2005-02-02 Incorporated Fernando's matshow - JDH
- 2005-02-01 Added Fernando's figure num patch, including experimental support for pylab backend switching, LineCollection.color warns, savefig now a figure method, fixed a close(fig) bug - JDH
- 2005-01-31 updated datalim in contour - JDH
- 2005-01-30 Added backend_qtagg.py provided by Sigve Tjora - SC
- 2005-01-28 Added tk.inspect rc param to .matplotlibrc. IDLE users should set tk.pythoninspect:True and interactive:True and backend:TkAgg
- 2005-01-28 Replaced examples/interactive.py with an updated script from Fernando Perez - SC
- 2005-01-27 Added support for shared x or y axes. See examples/shared_axis_demo.py and examples/ganged_plots.py
- 2005-01-27 Added Lee's patch for missing symbols leq and LEFTbracket to __mathtext_data - JDH
- 2005-01-26 Added Baptiste's two scales patch - see help(twinx) in the pylab interface for more info. See also examples/two_scales.py
- 2005-01-24 Fixed a mathtext parser bug that prevented font changes in sub/superscripts - JDH
- 2005-01-24 Fixed contour to work w/ interactive changes in colormaps, clim, etc - JDH
-
- 2005-01-21 matplotlib-0.71 released
- 2005-01-21 Refactored numerix to solve vexing namespace issues - JDH
- 2005-01-21 Applied Nadia's contour bug fix - JDH

- 2005-01-20 Made some changes to the contour routine - particularly `region=1` seems to fix a lot of the zigzag strangeness. Added colormaps as default for contour - JDH
 - 2005-01-19 Restored builtin names which were overridden (`min`, `max`, `abs`, `round`, and `sum`) in `pylab`. This is a potentially significant change for those who were relying on an array version of those functions that previously overrode builtin function names. - ADS
 - 2005-01-18 Added accents to `mathtext`: `hat`, `breve`, `grave`, `bar`, `acute`, `tilde`, `vec`, `dot`, `ddot`. All of them have the same syntax, e.g., to make an overbar you do `bar{o}` or to make an o umlaut you do `ddot{o}`. The shortcuts are also provided, e.g., `"o 'e 'e ~n .x ^y` - JDH
 - 2005-01-18 Plugged image resize memory leaks - JDH
 - 2005-01-18 Fixed some `mathtext` parser problems relating to superscripts
 - 2005-01-17 Fixed a `yticklabel` problem for colorbars under change of `clim` - JDH
 - 2005-01-17 Cleaned up Destroy handling in `wx` reducing `memleak/fig` from approx 800k to approx 6k- JDH
 - 2005-01-17 Added `kappa` to `latex_to_bakoma` - JDH
 - 2005-01-15 Support arbitrary colorbar axes and horizontal colorbars - JDH
 - 2005-01-15 Fixed colormap number of colors bug so that the colorbar has the same discretization as the image - JDH
 - 2005-01-15 Added Nadia's x,y contour fix - JDH
 - 2005-01-15 `backend_cairo`: added PDF support which requires `pycairo 0.1.4`. Its not usable yet, but is ready for when the Cairo PDF backend matures - SC
 - 2005-01-15 Added Nadia's x,y contour fix
 - 2005-01-12 Fixed `set clip_on` bug in `artist` - JDH
 - 2005-01-11 Reverted `pythoninspect` in `tkagg` - JDH
 - 2005-01-09 Fixed a `backend_bases` event bug caused when an event is triggered when location is `None` - JDH
 - 2005-01-07 Add patch from Stephen Walton to fix bug in `pylab.load()` when the `%` character is included in a comment. - ADS
 - 2005-01-07 Added `markerscale` attribute to `Legend` class. This allows the marker size in the legend to be adjusted relative to that in the plot. - ADS
 - 2005-01-06 Add patch from Ben Vanhaeren to make the `FigureManagerGTK vbox` a public attribute - SC
-

2004-12-30 Release 0.70

2004-12-28 Added coord location to key press and added a `examples/picker_demo.py`

2004-12-28 Fixed coords notification in `wx` toolbar - JDH

2004-12-28 Moved connection and disconnection event handling to the `FigureCanvasBase`. Backends now only need to connect one time for each of the button press, button release and key press/release functions. The base class deals with callbacks and multiple connections. This fixes flakiness on some backends (tk, wx) in the presence of multiple connections and/or disconnect - JDH

2004-12-27 Fixed PS `mathtext` bug where color was not set - Jochen please verify correct - JDH

2004-12-27 Added `Shadow` class and added `shadow` kwarg to `legend` and `pie` for shadow effect - JDH

2004-12-27 Added pie charts and new example/`pie_demo.py`

2004-12-23 Fixed an agg text rotation alignment bug, fixed some text kwarg processing bugs, and added `examples/text_rotation.py` to explain and demonstrate how text rotations and alignment work in matplotlib. - JDH

2004-12-22 0.65.1 released - JDH

2004-12-22 Fixed colorbar bug which caused colorbar not to respond to changes in colormap in some instances - JDH

2004-12-22 Refactored `NavigationToolbar` in tkagg to support app embedding, `init` now takes (canvas, window) rather than (canvas, figman) - JDH

2004-12-21 Refactored axes and subplot management - removed `add_subplot` and `add_axes` from the `FigureManager`. classic toolbar updates are done via an observer pattern on the figure using `add_axobserver`. Figure now maintains the axes stack (for gca) and supports axes deletion. Ported changes to GTK, Tk, Wx, and FLTK. Please test! Added `delaxes` - JDH

2004-12-21 Lots of image optimizations - 4x performance boost over 0.65 JDH

2004-12-20 Fixed a figimage bug where the axes is shown and modified tkagg to move the destroy binding into the show method.

2004-12-18 Minor refactoring of `NavigationToolbar2` to support embedding in an application - JDH

2004-12-14 Added `linestyle` to collections (currently broken) - JDH

2004-12-14 Applied Nadia's `setupext` patch to fix `libstdc++` link problem with `contour` and `solaris` -JDH

2004-12-14 A number of pychecker inspired fixes, including removal of `True` and `False` from `cbook` which I erroneously thought was needed for python2.2 - JDH

2004-12-14 Finished porting doc strings for set introspection. Used `silent_list` for many get funcs that return lists. JDH

2004-12-13 `dates.py`: removed all `timezone()` calls, except for UTC - SC

2004-12-13 0.65 released - JDH

- 2004-12-13 colors.py: rgb2hex(), hex2color() made simpler (and faster), also rgb2hex() - added round() instead of integer truncation hex2color() - changed 256.0 divisor to 255.0, so now '#ffffff' becomes (1.0,1.0,1.0) not (0.996,0.996,0.996) - SC
- 2004-12-11 Added ion and ioff to pylab interface - JDH
- 2004-12-11 backend_template.py: delete FigureCanvasTemplate.realize() - most backends don't use it and its no longer needed
- backend_ps.py, backend_svg.py: delete show() and draw_if_interactive() - they are not needed for image backends
- backend_svg.py: write direct to file instead of StringIO - SC
- 2004-12-10 Added zorder to artists to control drawing order of lines, patches and text in axes. See examples/zorder_demo.py - JDH
- 2004-12-10 Fixed colorbar bug with scatter - JDH
- 2004-12-10 Added Nadia Dencheva <dencheva@stsci.edu> contour code - JDH
- 2004-12-10 backend_cairo.py: got mathtext working - SC
- 2004-12-09 Added Norm Peterson's svg clipping patch
- 2004-12-09 Added Matthew Newville's wx printing patch
- 2004-12-09 Migrated matlab to pylab - JDH
- 2004-12-09 backend_gtk.py: split into two parts
- backend_gdk.py - an image backend
 - backend_gtk.py - A GUI backend that uses GDK - SC
- 2004-12-08 backend_gtk.py: remove quit_after_print_xvfb(*args), show_xvfb(), Dialog_MeasureTool(gtk.Dialog) one month after sending mail to matplotlib-users asking if anyone still uses these functions - SC
- 2004-12-02 backend_bases.py, backend_template.py: updated some of the method documentation to make them consistent with each other - SC
- 2004-12-04 Fixed multiple bindings per event for TkAgg mpl_connect and mpl_disconnect. Added a "test_disconnect" command line parameter to coords_demo.py JTM
- 2004-12-04 Fixed some legend bugs JDH
- 2004-11-30 Added over command for oneoff over plots. e.g., over(plot, x, y, lw=2). Works with any plot function.
- 2004-11-30 Added bbox property to text - JDH
- 2004-11-29 Zoom to rect now respect reversed axes limits (for both linear and log axes). - GL
- 2004-11-29 Added the over command to the matlab interface. over allows you to add an overlay plot regardless of hold state. - JDH
- 2004-11-25 Added Printf to mplutils for printf style format string formatting in C++ (should help write better exceptions)

2004-11-24 `IMAGE_FORMAT`: remove from `agg` and `gtkagg` backends as its no longer used - SC

2004-11-23 Added matplotlib compatible set and get introspection. See `set_and_get.py`

2004-11-23 applied Norbert's patched and exposed legend configuration to kwargs - JDH

2004-11-23 `backend_gtk.py`: added a default exception handler - SC

2004-11-18 `backend_gtk.py`: change so that the backend knows about all image formats and does not need to use `IMAGE_FORMAT` in other backends - SC

2004-11-18 Fixed some `report_error` bugs in string interpolation as reported on SF bug tracker - JDH

2004-11-17 `backend_gtkcairo.py`: change so all `print_figure()` calls render using Cairo and get saved using `backend_gtk.print_figure()` - SC

2004-11-13 `backend_cairo.py`: Discovered the magic number (96) required for Cairo PS plots to come out the right size. Restored Cairo PS output and added support for landscape mode - SC

2004-11-13 Added `ishold` - JDH

2004-11-12 Added many new matlab colormaps - autumn bone cool copper flag gray hot hsv jet pink prism spring summer winter - PG

2004-11-11 greatly simplify the emitted postscript code - JV

2004-11-12 Added new plotting functions `spy`, `spy2` for sparse matrix visualization - JDH

2004-11-11 Added `rgrids`, `thetragrids` for customizing the grid locations and labels for polar plots - JDH

2004-11-11 make the Gtk backends build without an X-server connection - JV

2004-11-10 `matplotlib/__init__.py`: Added `FROZEN` to signal we are running under `py2exe` (or similar) - is used by `backend_gtk.py` - SC

2004-11-09 `backend_gtk.py`: Made fix suggested by maffew@cat.org.au to prevent problems when `py2exe` calls `pygtk.require()`. - SC

2004-11-09 `backend_cairo.py`: Added support for printing to a fileobject. Disabled cairo PS output which is not working correctly. - SC

2004-11-08 matplotlib-0.64 released

2004-11-04 Changed `-dbackend` processing to only use known backends, so we don't clobber other non-matplotlib uses of `-d`, like `-debug`.

2004-11-04 `backend_agg.py`: added `IMAGE_FORMAT` to list the formats that the backend can save to. `backend_gtkagg.py`: added support for saving JPG files by using the GTK backend - SC

2004-10-31 `backend_cairo.py`: now produces png and ps files (although the figure sizing needs some work). `pycairo` did not wrap all the necessary functions, so I wrapped them myself, they are included in the `backend_cairo.py` doc string. - SC

- 2004-10-31 backend_ps.py: clean up the generated PostScript code, use the PostScript stack to hold intermediate values instead of storing them in the dictionary. - JV
- 2004-10-30 backend_ps.py, ft2font.cpp, ft2font.h: fix the position of text in the PostScript output. The new FT2Font method get_descent gives the distance between the lower edge of the bounding box and the baseline of a string. In backend_ps the text is shifted upwards by this amount. - JV
- 2004-10-30 backend_ps.py: clean up the code a lot. Change the PostScript output to be more DSC compliant. All definitions for the generated PostScript are now in a PostScript dictionary 'mpldict'. Moved the long comment about drawing ellipses from the PostScript output into a Python comment. - JV
- 2004-10-30 backend_gtk.py: removed FigureCanvasGTK.realize() as its no longer needed. Merged ColorManager into GraphicsContext backend_bases.py: For set_capstyle/joinstyle() only set cap or joinstyle if there is no error. - SC
- 2004-10-30 backend_gtk.py: tidied up print_figure() and removed some of the dependency on widget events - SC
- 2004-10-28 backend_cairo.py: The renderer is complete except for mathtext, draw_image() and clipping. gtkcairo works reasonably well. cairo does not yet create any files since I can't figure how to set the 'target surface', I don't think pycairo wraps the required functions - SC
- 2004-10-28 backend_gtk.py: Improved the save dialog (GTK 2.4 only) so it presents the user with a menu of supported image formats - SC
- 2004-10-28 backend_svg.py: change print_figure() to restore original face/edge color backend_ps.py : change print_figure() to ensure original face/edge colors are restored even if there's an IOError - SC
- 2004-10-27 Applied Norbert's errorbar patch to support barsabove kwarg
- 2004-10-27 Applied Norbert's legend patch to support None handles
- 2004-10-27 Added two more backends: backend_cairo.py, backend_gtkcairo.py They are not complete yet, currently backend_gtkcairo just renders polygons, rectangles and lines - SC
- 2004-10-21 Added polar axes and plots - JDH
- 2004-10-20 Fixed corrcoef bug exposed by corrcoef(X) where X is matrix
 - JDH
- 2004-10-19 Added kwarg support to xticks and yticks to set ticklabel text properties – thanks to T. Edward Whalen for the suggestion
- 2004-10-19 Added support for PIL images in imshow(), image.py - ADS
- 2004-10-19 Re-worked exception handling in _image.py and _transforms.py to avoid masking problems with shared libraries. - JTM
- 2004-10-16 Streamlined the matlab interface wrapper, removed the noplot option to hist - just use mlab.hist instead.
- 2004-09-30 Added Andrew Dalke's strftime code to extend the range of dates supported by the DateFormatter - JDH

2004-09-30 Added barh - JDH

2004-09-30 Removed fallback to alternate array package from numerix so that ImportError are easier to debug. JTM

2004-09-30 Add GTK+ 2.4 support for the message in the toolbar. SC

2004-09-30 Made some changes to support python22 - lots of doc fixes. - JDH

2004-09-29 Added a Verbose class for reporting - JDH

2004-09-28 Released 0.63.0

2004-09-28 Added save to file object for agg - see examples/print_stdout.py

2004-09-24 Reorganized all py code to lib subdir

2004-09-24 Fixed axes resize image edge effects on interpolation - required upgrade to agg22 which fixed an agg bug related to this problem

2004-09-20 Added toolbar2 message display for backend_tkagg. JTM

2004-09-17 Added coords formatter attributes. These must be callable,
and return a string for the x or y data. These will be used to format the x and y
data for the coords box. Default is the axis major formatter. e.g.:

```
# format the coords message box
def price(x): return '$%1.2f'%x
ax.format_xdata = DateFormatter('%Y-%m-%d')
ax.format_ydata = price
```

2004-09-17 Total rewrite of dates handling to use python datetime with num2date, date2num and drange. pytz for timezone handling, dateutils for sophisticated ticking. date ranges from 0001-9999 are supported. rrules allow arbitrary date ticking. examples/date_demo*.py converted to show new usage. new example examples/date_demo_rrule.py shows how to use rrules in date plots. The date locators are much more general and almost all of them have different constructors. See matplotlib.dates for more info.

2004-09-15 Applied Fernando's backend __init__ patch to support easier backend maintenance. Added his numutils to mlab. JDH

2004-09-16 Re-designated all files in matplotlib/images as binary and w/o keyword substitution using "cvs admin -kb *.svg ...". See binary files in "info cvs" under Linux. This was messing up builds from CVS on windows since CVS was doing lf -> cr/lf and keyword substitution on the bitmaps. - JTM

2004-09-15 Modified setup to build array-package-specific extensions for those extensions which are array-aware. Setup builds extensions automatically for either Numeric, numarray, or both, depending on what you have installed. Python proxy modules for the array-aware extensions import the version optimized for numarray or Numeric determined by numerix. - JTM

2004-09-15 Moved definitions of infinity from mlab to numerix to avoid divide by zero warnings for numarray - JTM

2004-09-09 Added axhline, axvline, axhspan and axvspan

2004-08-30 matplotlib 0.62.4 released

2004-08-30 Fixed a multiple images with different extent bug, Fixed markerfacecolor as RGB tuple

2004-08-27 Mathtext now more than 5x faster. Thanks to Paul Mcguire for fixes both to pyparsing and to the matplotlib grammar! mathtext broken on python2.2

2004-08-25 Exposed Darren's and Greg's log ticking and formatting options to semilogx and friends

2004-08-23 Fixed grid w/o args to toggle grid state - JDH

2004-08-11 Added Gregory's log patches for major and minor ticking

2004-08-18 Some pixel edge effects fixes for images

2004-08-18 Fixed TTF files reads in backend_ps on win32.

2004-08-18 Added base and subs properties for logscale plots, user modifiable using set_[x,y]scale('log',base=b,subs=[mt1,mt2,...]) - GL

2004-08-18 fixed a bug exposed by trying to find the HOME dir on win32 thanks to Alan Issac for pointing to the light - JDH

2004-08-18 fixed errorbar bug in setting ecolor - JDH

2004-08-12 Added Darren Dale's exponential ticking patch

2004-08-11 Added Gregory's fltkagg backend

2004-08-09 matplotlib-0.61.0 released

2004-08-08 backend_gtk.py: get rid of the final PyGTK deprecation warning by replacing gtkOptionMenu with gtkMenu in the 2.4 version of the classic toolbar.

2004-08-06 Added Tk zoom to rect rectangle, proper idle drawing, and keybinding - JDH

2004-08-05 Updated installing.html and INSTALL - JDH

2004-08-01 backend_gtk.py: move all drawing code into the expose_event()

2004-07-28 Added Greg's toolbar2 and backend_*agg patches - JDH

2004-07-28 Added image.imread with support for loading png into numerix arrays

2004-07-28 Added key modifiers to events - implemented dynamic updates and rubber banding for interactive pan/zoom - JDH

2004-07-27 did a readthrough of SVG, replacing all the string additions with string interps for efficiency, fixed some layout problems, added font and image support (through external pngs) - JDH

2004-07-25 backend_gtk.py: modify toolbar2 to make it easier to support GTK+ 2.4. Add GTK+ 2.4 toolbar support. - SC

- 2004-07-24 backend_gtk.py: Simplified classic toolbar creation - SC
- 2004-07-24 Added images/matplotlib.svg to be used when GTK+ windows are minimised - SC
- 2004-07-22 Added right mouse click zoom for NavigationToolbar2 panning mode. - JTM
- 2004-07-22 Added NavigationToolbar2 support to backend_tkagg. Minor tweak to backend_bases. - JTM
- 2004-07-22 Incorporated Gergory's renderer cache and buffer object cache - JDH
- 2004-07-22 Backend_gtk.py: Added support for GtkFileChooser, changed FileSelection/FileChooser so that only one instance pops up, and made them both modal. - SC
- 2004-07-21 Applied backend_agg memory leak patch from hayden - jocallo@online.no. Found and fixed a leak in binary operations on transforms. Moral of the story: never incref where you meant to decref! Fixed several leaks in ft2font: moral of story: almost always return Py::asObject over Py::Object - JDH
- 2004-07-21 Fixed a to string memory allocation bug in agg and image modules - JDH
- 2004-07-21 Added mpl_connect and mpl_disconnect to matlab interface - JDH
- 2004-07-21 Added beginnings of users_guide to CVS - JDH
- 2004-07-20 ported toolbar2 to wx
- 2004-07-20 upgraded to agg21 - JDH
- 2004-07-20 Added new icons for toolbar2 - JDH
- 2004-07-19 Added vertical mathtext for *Agg and GTK - thanks Jim Benson! - JDH
- 2004-07-16 Added ps/eps/svg savefig options to wx and gtk JDH
- 2004-07-15 Fixed python framework tk finder in setupest.py - JDH
- 2004-07-14 Fixed layer images demo which was broken by the 07/12 image extent fixes - JDH
- 2004-07-13 Modified line collections to handle arbitrary length segments for each line segment. - JDH
- 2004-07-13 Fixed problems with image extent and origin - set_image_extent deprecated. Use imshow(blah, blah, extent=(xmin, xmax, ymin, ymax)) instead - JDH
- 2004-07-12 Added prototype for new nav bar with codified event handling. Use mpl_connect rather than connect for matplotlib event handling. toolbar style determined by rc toolbar param. backend status: gtk: prototype, wx: in progress, tk: not started - JDH
- 2004-07-11 backend_gtk.py: use builtin round() instead of redefining it.
- SC
- 2004-07-10 Added embedding_in_wx3 example - ADS
- 2004-07-09 Added dynamic_image_wxagg to examples - ADS
- 2004-07-09 added support for embedding TrueType fonts in PS files - PEB

2004-07-09 fixed a sfnt bug exposed if font cache is not built

2004-07-09 added default arg None to matplotlib.matlab grid command to toggle current grid state

2004-07-08 0.60.2 released

2004-07-08 fixed a mathtext bug for '6'

2004-07-08 added some numarray bug workarounds

2004-07-07 0.60 released

2004-07-07 Fixed a bug in dynamic_demo_wx

2004-07-07 backend_gtk.py: raise SystemExit immediately if 'import pygtk' fails - SC

2004-07-05 Added new mathtext commands over{sym1}{sym2} and under{sym1}{sym2}

2004-07-05 Unified image and patch collections colormapping and scaling args. Updated docstrings for all - JDH

2004-07-05 Fixed a figure legend bug and added examples/figlegend_demo.py - JDH

2004-07-01 Fixed a memory leak in image and agg to string methods

2004-06-25 Fixed fonts_demo spacing problems and added a kwargs version of the fonts_demo fonts_demo_kw.py - JDH

2004-06-25 finance.py: handle case when urlopen() fails - SC

2004-06-24 Support for multiple images on axes and figure, with blending. Support for upper and lower image origins. clim, jet and gray functions in matlab interface operate on current image - JDH

2004-06-23 ported code to Perry's new colormap and norm scheme. Added new rc attributes image.aspect, image.interpolation, image.cmap, image.lut, image.origin

2004-06-20 backend_gtk.py: replace gtk.TRUE/FALSE with True/False. simplified __make_axis_menu(). - SC

2004-06-19 anim_tk.py: Updated to use TkAgg by default (not GTK) backend_gtk.py: Added '_' in front of private widget creation functions - SC

2004-06-17 backend_gtk.py: Create a GC once in realise(), not every time draw() is called. - SC

2004-06-16 Added new py2exe FAQ entry and added frozen support in get_data_path for py2exe - JDH

2004-06-16 Removed GTKGD, which was always just a proof-of-concept backend - JDH

2004-06-16 backend_gtk.py updates to replace deprecated functions

gtk.mainquit(), gtk.mainloop(). Update NavigationToolbar to use the new GtkToolbar API - SC

2004-06-15 removed `set_default_font` from `font_manager` to unify font customization using the new function `rc`. See `API_CHANGES` for more info. The examples `fonts_demo.py` and `fonts_demo_kw.py` are ported to the new API - JDH

2004-06-15 Improved (yet again!) axis scaling to properly handle singleton plots - JDH

2004-06-15 Restored the old `FigureCanvasGTK.draw()` - SC

2004-06-11 More memory leak fixes in transforms and `ft2font` - JDH

2004-06-11 Eliminated numerix `.numerix` file and environment variable `NUMERIX`. Fixed bug which prevented command line overrides: `--numarray` or `--numeric`. - JTM

2004-06-10 Added `rc` configuration function `rc`; deferred all `rc` param setting until object creation time; added new `rc` attrs: `lines.markerfacecolor`, `lines.markeredgewidth`, `patch.linewidth`, `patch.facecolor`, `patch.edgecolor`, `patch.antialiased`; see `examples/customize_rc.py` for usage - JDH

2004-06-09 0.54.2 released

2004-06-08 Rewrote `ft2font` using `CXX` as part of general memory leak fixes; also fixed transform memory leaks - JDH

2004-06-07 Fixed several problems with log ticks and scaling - JDH

2004-06-07 Fixed width/height issues for images - JDH

2004-06-03 Fixed `draw_if_interactive` bug for `semilogx`;

2004-06-02 Fixed text clipping to clip to axes - JDH

2004-06-02 Fixed leading newline text and multiple newline text - JDH

2004-06-02 Fixed `plot_date` to return lines - JDH

2004-06-01 Fixed plot to work with `x` or `y` having shape `N,1` or `1,N` - JDH

2004-05-31 Added `renderer.markeredgewidth` attribute of `Line2D`. - ADS

2004-05-29 Fixed tick label clipping to work with navigation.

2004-05-28 Added `renderer` grouping commands to support groups in `SVG/PS`. - JDH

2004-05-28 Fixed, this time I really mean it, the singleton plot `plot([0])` scaling bug; Fixed Flavio's `shape = N,1` bug - JDH

2004-05-28 added colorbar - JDH

2004-05-28 Made some changes to the `matplotlib.colors.Colormap` to properly support `clim` - JDH

2004-05-27 0.54.1 released

2004-05-27 Lots of small bug fixes: rotated text at negative angles, `errorbar` `capsize` and `autoscaling`, right tick label position, `gtkagg` on win98, `alpha` of figure background, singleton plots - JDH

- 2004-05-26 Added Gary's errorbar stuff and made some fixes for length one plots and constant data plots - JDH
- 2004-05-25 Tweaked TkAgg backend so that `canvas.draw()` works more like the other backends. Fixed a bug resulting in 2 draws per figure manager `show()`. - JTM
-
- 2004-05-19 0.54 released
- 2004-05-18 Added newline separated text with rotations to `text.Text` layout - JDH
- 2004-05-16 Added fast pcolor using `PolyCollections`. - JDH
- 2004-05-14 Added fast polygon collections - changed scatter to use them. Added multiple symbols to scatter. 10x speedup on large scatters using `*Agg` and 5X speedup for ps. - JDH
- 2004-05-14 On second thought... created an "nx" namespace in `numerix` which maps type names onto typecodes the same way for both `numarray` and `Numeric`. This undoes my previous change immediately below. To get a typename for `Int16` useable in a `Numeric` extension: say `nx.Int16`. - JTM
- 2004-05-15 Rewrote transformation class in extension code, simplified all the artist constructors - JDH
- 2004-05-14 Modified the type definitions in the `numarray` side of `numerix` so that they are `Numeric` typecodes and can be used with `Numeric` complex extensions. The original `numarray` types were renamed to `type<old_name>`. - JTM
- 2004-05-06 Gary Ruben sent me a bevy of new plot symbols and markers. See `matplotlib.matlab.plot` - JDH
- 2004-05-06 Total rewrite of `mathtext` - factored `ft2font` stuff out of layout engine and defined abstract class for font handling to lay groundwork for ps `mathtext`. Rewrote parser and made layout engine much more precise. Fixed all the layout hacks. Added spacing commands `/` and `hspace`. Added composite chars and defined `angstrom`. - JDH
- 2004-05-05 Refactored text instances out of backend; aligned text with arbitrary rotations is now supported - JDH
- 2004-05-05 Added a Matrix capability for `numarray` to `numerix`. JTM
- 2004-05-04 Updated `whats_new.html.template` to use dictionary and template loop, added anchors for all versions and items; updated `goals.txt` to use those for links. PG
- 2004-05-04 Added `fonts_demo.py` to `backend_driver`, and AFM and TTF font caches to `font_manager.py` - PEB
- 2004-05-03 Redid `goals.html.template` to use a `goals.txt` file that has a pseudo restructured text organization. PG
- 2004-05-03 Removed the close buttons on all GUIs and added the python `#!/` bang line to the examples following Steve Chaplin's advice on matplotlib dev
- 2004-04-29 Added CXX and rewrote `backend_agg` using it; tracked down and fixed agg memory leak - JDH

2004-04-29 Added stem plot command - JDH

2004-04-28 Fixed PS scaling and centering bug - JDH

2004-04-26 Fixed errorbar autoscale problem - JDH

2004-04-22 Fixed copy tick attribute bug, fixed singular datalim ticker bug; fixed mathtext font-size interactive bug. - JDH

2004-04-21 Added calls to draw_if_interactive to axes(), legend(), and pcolor(). Deleted duplicate pcolor(). - JTM

2004-04-21 matplotlib 0.53 release

2004-04-19 Fixed vertical alignment bug in PS backend - JDH

2004-04-17 Added support for two scales on the "same axes" with tick different ticking and labeling left right or top bottom. See examples/two_scales.py - JDH

2004-04-17 Added default dirs as list rather than single dir in setupext.py - JDH

2004-04-16 Fixed wx exception swallowing bug (and there was much rejoicing!) - JDH

2004-04-16 Added new ticker locator a formatter, fixed default font return - JDH

2004-04-16 Added get_name method to FontProperties class. Fixed font lookup in GTK and WX backends. - PEB

2004-04-16 Added get- and set_fontstyle methods. - PEB

2004-04-10 Mathtext fixes: scaling with dpi, - JDH

2004-04-09 Improved font detection algorithm. - PEB

2004-04-09 Move deprecation warnings from text.py to __init__.py - PEB

2004-04-09 Added default font customization - JDH

2004-04-08 Fixed viewlim set problem on axes and axis. - JDH

2004-04-07 Added validate_comma_sep_str and font properties parameters to __init__. Removed font families and added rcParams to FontProperties __init__ arguments in font_manager. Added default font property parameters to .matplotlibrc file with descriptions. Added deprecation warnings to the get_ - and set_fontXXX methods of the Text object. - PEB

2004-04-06 Added load and save commands for ASCII data - JDH

2004-04-05 Improved font caching by not reading AFM fonts until needed. Added better documentation. Changed the behaviour of the get_family, set_family, and set_name methods of FontProperties. - PEB

2004-04-05 Added WXAagg backend - JDH

2004-04-04 Improved font caching in backend_agg with changes to font_manager - JDH

2004-03-29 Fixed fontdicts and kwargs to work with new font manager - JDH

This is the Old, stale, never used changelog

2002-12-10 - Added a TODO file and CHANGELOG. Lots to do – get

crackin'!

- Fixed y zoom tool bug
- Adopted a compromise fix for the y data clipping problem. The problem was that for solid lines, the y data clipping (as opposed to the gc clipping) caused artifactual horizontal solid lines near the ylim boundaries. I did a 5% offset hack in Axes set_ylim functions which helped, but didn't cure the problem for very high gain y zooms. So I disabled y data clipping for connected lines. If you need extensive y clipping, either plot(y,x) because x data clipping is always enabled, or change the __set_clip code to 'if 1' as indicated in the lines.py src. See __set_clip in lines.py and set_ylim in figure.py for more information.

2002-12-11 - Added a measurement dialog to the figure window to

measure axes position and the delta x delta y with a left mouse drag. These defaults can be overridden by deriving from Figure and overriding button_press_event, button_release_event, and motion_notify_event, and __dialog_measure_tool.

- fixed the navigation dialog so you can check the axes the navigation buttons apply to.

2003-04-23 Released matplotlib v0.1

2003-04-24 Added a new line style PixelLine2D which is the plots the markers as pixels (as small as possible) with format symbol ','

Added a new class Patch with derived classes Rectangle, RegularPolygon and Circle

2003-04-25 Implemented new functions errorbar, scatter and hist

Added a new line type '|' which is a vline. syntax is plot(x, Y, '|') where y.shape = len(x),2 and each row gives the ymin,ymax for the respective values of x. Previously I had implemented vlins as a list of lines, but I needed the efficiency of the numeric clipping for large numbers of vlins outside the viewport, so I wrote a dedicated class Vline2D which derives from Line2D

2003-05-01

Fixed ytick bug where grid and tick show outside axis viewport with gc clip

2003-05-14

Added new ways to specify colors 1) matlab format string 2) html-style hex string, 3) rgb tuple. See examples/color_demo.py

2003-05-28

Changed figure rendering to draw from a pixmap to reduce flicker. See examples/system_monitor.py for an example where the plot is continuously updated w/o flicker. This example is meant to simulate a system monitor that shows free CPU, RAM, etc...

2003-08-04

Added Jon Anderson's GTK shell, which doesn't require pygtk to have threading built-in and looks nice!

2003-08-25

Fixed deprecation warnings for python2.3 and pygtk-1.99.18

2003-08-26

Added figure text with new example examples/figtext.py

2003-08-27

Fixed bugs i figure text with font override dictionairies and fig text that was placed outside the window bounding box

2003-09-1 thru 2003-09-15

Added a postscript and a GD module backend

2003-09-16

Fixed font scaling and point scaling so circles, squares, etc on lines will scale with DPI as will fonts. Font scaling is not fully implemented on the gtk backend because I have not figured out how to scale fonts to arbitrary sizes with GTK

2003-09-17

Fixed figure text bug which crashed X windows on long figure text extending beyond display area. This was, I believe, due to the vestigial erase functionality that was no longer needed since I began rendering to a pixmap

2003-09-30 Added legend

2003-10-01 Fixed bug when colors are specified with rgb tuple or hex string.

2003-10-21 Andrew Straw provided some legend code which I modified and incorporated. Thanks Andrew!

2003-10-27 Fixed a bug in axis.get_view_distance that affected zoom in versus out with interactive scrolling, and a bug in the axis text reset system that prevented the text from being redrawn on a interactive gtk view lim set with the widget

Fixed a bug in that prevented the manual setting of ticklabel strings from working properly

2003-11-02 - Do a nearest neighbor color pick on GD when allocate fails

2003-11-02

- Added pcolor plot
- Added MRI example
- Fixed bug that screwed up label position if xticks or yticks were empty
- added nearest neighbor color picker when GD max colors exceeded
- fixed figure background color bug in GD backend

2003-11-10 - 2003-11-11

- major refactoring.
 - Ticks (with labels, lines and grid) handled by dedicated class
 - Artist now know bounding box and dpi
 - Bounding boxes and transforms handled by dedicated classes
 - legend in dedicated class. Does a better job of alignment and bordering. Can be initialized with specific line instances. See examples/legend_demo2.py

2003-11-14 Fixed legend positioning bug and added new position args

2003-11-16 Finsihed porting GD to new axes API

2003-11-20 - add TM for matlab on website and in docs

2003-11-20 - make a nice errorbar and scatter screenshot

2003-11-20 - auto line style cycling for multiple line types broken

2003-11-18 (using inkrect) :logical rect too big on gtk backend

2003-11-18 ticks don't reach edge of axes in gtk mode – rounding error?

2003-11-20 - port Gary's errorbar code to new API before 0.40

2003-11-20 - problem with stale `_set_font`. legend axes box doesn't resize on save in GTK backend – see htdocs legend_demo.py

2003-11-21 - make a dash-dot dict for the GC

2003-12-15 - fix install path bug

5.3.2 New in matplotlib 0.98.4

Table of Contents

- New in matplotlib 0.98.4
 - Legend enhancements
 - Fancy annotations and arrows
 - Native OS X backend
 - psd amplitude scaling
 - Fill between
 - Lots more

It's been four months since the last matplotlib release, and there are a lot of new features and bug-fixes.

Thanks to Charlie Moad for testing and preparing the source release, including binaries for OS X and Windows for python 2.4 and 2.5 (2.6 and 3.0 will not be available until numpy is available on those releases). Thanks to the many developers who contributed to this release, with contributions from Jae-Joon Lee, Michael Droettboom, Ryan May, Eric Firing, Manuel Metz, Jouni K. Seppänen, Jeff Whitaker, Darren Dale, David Kaplan, Michiel de Hoon and many others who submitted patches

Legend enhancements

Jae-Joon has rewritten the legend class, and added support for multiple columns and rows, as well as fancy box drawing. See `legend()` and `matplotlib.legend.Legend`.

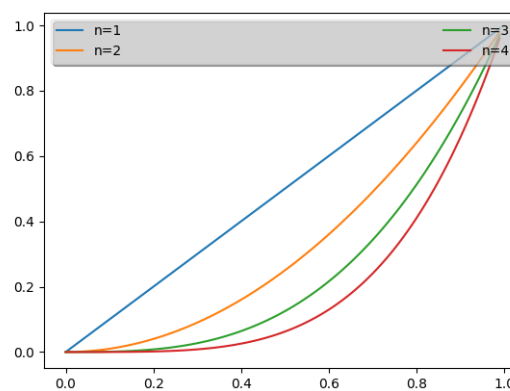


Fig. 1: Whats New 98 4 Legend

Fancy annotations and arrows

Jae-Joon has added lots of support to annotations for drawing fancy boxes and connectors in annotations. See `annotate()` and `BoxStyle`, `ArrowStyle`, and `ConnectionStyle`.

Native OS X backend

Michiel de Hoon has provided a native Mac OSX backend that is almost completely implemented in C. The backend can therefore use Quartz directly and, depending on the application, can be orders of magnitude faster than the existing backends. In addition, no third-party libraries are needed other than Python and NumPy. The backend is interactive from the usual terminal application on Mac using regular Python. It hasn't been tested with ipython yet, but in principle it should to work there as well. Set 'backend : macosx' in your matplotlibrc file, or run your script with:

```
> python myfile.py -dmacosx
```

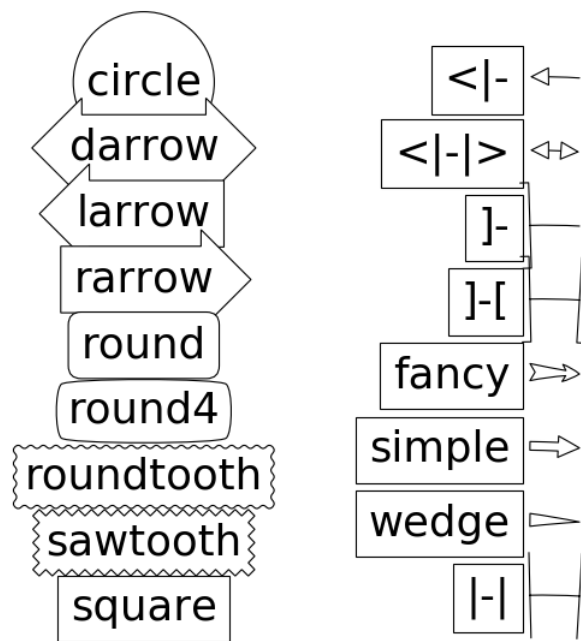


Fig. 2: Whats New 98 4 Fancy

psd amplitude scaling

Ryan May did a lot of work to rationalize the amplitude scaling of `psd()` and friends. See /gallery/lines_bars_and_markers/psd_demo. The changes should increase MATLAB compatibility and increase scaling options.

Fill between

Added a `fill_between()` function to make it easier to do shaded region plots in the presence of masked data. You can pass an `x` array and a `ylower` and `yupper` array to fill between, and an optional `where` argument which is a logical mask where you want to do the filling.

Lots more

Here are the 0.98.4 notes from the CHANGELOG:

Added mdehoon's [native macosx backend](#) from [sf patch 2179017](#) - JDH

Removed the prints in the `set_*style` commands. Return the list of pretty-printed strings instead - JDH

(continues on next page)

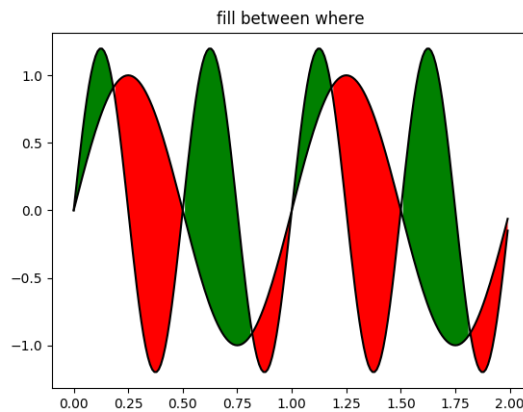


Fig. 3: Whats New 98 4 Fill Between

(continued from previous page)

Some of the changes Michael made to improve the output of the `property` tables in the rest docs broke or made difficult to use some of the interactive doc helpers, e.g., `setp` and `getp`. Having all the rest markup in the ipython shell also confused the docstrings. I added a new rc param `docstring.harcopy`, to format the docstrings differently for hardcopy and other use. The `ArtistInspector` could use a little refactoring now since there is duplication of effort between the rest output and the non-rest output - JDH

Updated spectral methods (`psd`, `csd`, etc.) to scale one-sided densities by a factor of 2 and, optionally, scale all densities by the sampling frequency. This gives better MATLAB compatibility. -RM

Fixed alignment of ticks in colorbars. -MGD

drop the deprecated "new" keyword of `np.histogram()` for numpy 1.2 or later. -JJL

Fixed a bug in svg backend that `new_figure_manager()` ignores keywords arguments such as `figsize`, etc. -JJL

Fixed a bug that the handlelength of the new legend class `set` too short when `numpoints=1` -JJL

Added support for data with units (e.g., dates) to `Axes.fill_between`. -RM

Added `fancybox` keyword to legend. Also applied some changes for

(continues on next page)

(continued from previous page)

better look, including baseline adjustment of the multiline texts so that it is center aligned. -JJL

The transmuter classes in the patches.py are reorganized as subclasses of the Style classes. A few more box and arrow styles are added. -JJL

Fixed a bug in the new legend class that didn't allowed a tuple of coordinate values as loc. -JJL

Improve checks for external dependencies, using subprocess (instead of deprecated popen*) and distutils (for version checking) - DSD

Reimplementation of the legend which supports baseline alignment, multi-column, and expand mode. - JJL

Fixed histogram autoscaling bug when bins or range are given explicitly (fixes Debian bug 503148) - MM

Added rcParam axes.unicode_minus which allows plain hyphen for minus when False - JDH

Added scatterpoints support in Legend. patch by Erik Tollerud - JJL

Fix crash in log ticking. - MGD

Added static helper method BrokenHBarCollection.span_where and Axes/pyplot method fill_between. See examples/pylab/fill_between.py - JDH

Add x_isdata and y_isdata attributes to Artist instances, and use them to determine whether either or both coordinates are used when updating dataLim. This is used to fix autoscaling problems that had been triggered by axhline, axhspan, axvline, axvspan. - EF

Update the psd(), csd(), cohere(), and specgram() methods of Axes and the csd() cohere(), and specgram() functions in mlab to be in sync with the changes to psd(). In fact, under the hood, these all call the same core to do computations. - RM

Add 'pad_to' and 'sides' parameters to mlab.psd() to allow controlling of zero padding and returning of negative frequency components, respectively. These are added in a way that does not

(continues on next page)

(continued from previous page)

change the API. - RM

Fix handling of `c` kwarg by `scatter`; generalize `is_string_like` to accept `numpy` and `numpy.ma` string array scalars. - RM and EF

Fix a possible EINTR problem in `dviread`, which might help when saving pdf files from the qt backend. - JKS

Fix bug with zoom to rectangle and twin axes - MGD

Added Jae Joon's fancy arrow, box and annotation enhancements -- see `examples/pylab_examples/annotation_demo2.py`

Autoscaling is now supported with shared axes - EF

Fixed exception in `dviread` that happened with Minion - JKS

`set_xlim`, `ylim` now return a copy of the viewlim array to avoid modify inplace surprises

Added image thumbnail generating function `matplotlib.image.thumbnail`. See `examples/misc/image_thumbnail.py` - JDH

Applied `scatleg` patch based on ideas and work by Erik Tollerud and Jae-Joon Lee. - MM

Fixed bug in pdf backend: if you pass a file object for output instead of a filename, e.g., in a web app, we now flush the object at the end. - JKS

Add path simplification support to paths with gaps. - EF

Fix problem with AFM files that don't specify the font's full name or family name. - JKS

Added '`scilimits`' kwarg to `Axes.ticklabel_format()` method, for easy access to the `set_powerlimits` method of the major `ScalarFormatter`. - EF

Experimental new kwarg `borderpad` to replace `pad` in legend, based on suggestion by Jae-Joon Lee. - EF

Allow spy to ignore zero values in sparse arrays, based on patch by Tony Yu. Also fixed plot to handle empty data arrays, and

(continues on next page)

(continued from previous page)

fixed handling of markers `in` figlegend. - EF

Introduce drawstyles `for` lines. Transparently split linestyles like `'steps--'` into drawstyle `'steps'` and linestyle `'--'`. Legends always use drawstyle `'default'`. - MM

Fixed quiver and quiverkey bugs (failure to scale properly when resizing) and added additional methods `for` determining the arrow angles - EF

Fix polar interpolation to handle negative values of theta - MGD

Reorganized `cbook` and `mlab` methods related to numerical calculations that have little to do with the goals of those two modules into a separate module `numerical_methods.py`. Also, added ability to select points and stop point selection with keyboard `in` `ginput` and manual contour labeling code. Finally, fixed contour labeling bug. - DMK

Fix backtick `in` Postscript output. - MGD

[2089958] Path simplification `for` vector output backends
Leverage the simplification code exposed through `path_to_polygons` to simplify certain well-behaved paths `in` the vector backends (PDF, PS and SVG). `"path.simplify"` must be set to `True` `in` `matplotlibrc` `for` this to work. - MGD

Add `"filled"` kwarg to `Path.intersects_path` and `Path.intersects_bbox`. - MGD

Changed full arrows slightly to avoid an xpdf rendering problem reported by Friedrich Hagedorn. - JKS

Fix conversion of quadratic to cubic Bezier curves `in` PDF and PS backends. Patch by Jae-Joon Lee. - JKS

Added 5-point star marker to plot command `q-` EF

Fix hatching `in` PS backend - MGD

Fix log with base 2 - MGD

Added support `for` bilinear interpolation `in` `NonUniformImage`; patch by Gregory Lielens. - EF

(continues on next page)

(continued from previous page)

Added support [for](#) multiple histograms [with](#) data of different length - MM

Fix step plots [with](#) log scale - MGD

Fix masked arrays [with](#) markers [in](#) non-Agg backends - MGD

Fix clip_on kwarg so it actually works correctly - MGD

Fix locale problems [in](#) SVG backend - MGD

fix quiver so masked values are [not](#) plotted - JSW

improve interactive pan/zoom [in](#) qt4 backend on windows - DSD

Fix more bugs [in](#) NaN/inf handling. In particular, path simplification (which does [not](#) handle NaNs [or](#) infs) will be turned off automatically when infs [or](#) NaNs are present. Also masked arrays are now converted to arrays [with](#) NaNs [for](#) consistent handling of masks [and](#) NaNs - MGD [and](#) EF

Added support [for](#) arbitrary rasterization resolutions to the SVG backend. - MW

5.3.3 New in matplotlib 0.99

Table of Contents

- [New in matplotlib 0.99](#)
 - [New documentation](#)
 - [mplot3d](#)
 - [axes grid toolkit](#)
 - [Axis spine placement](#)

New documentation

Jae-Joon Lee has written two new guides [Legend guide](#) and [Advanced Annotation](#). Michael Sarahan has written [Image tutorial](#). John Hunter has written two new tutorials on working with paths and transformations: [Path Tutorial](#) and [Transformations Tutorial](#).

mplot3d

Reinier Heeres has ported John Porter's mplot3d over to the new matplotlib transformations framework, and it is now available as a toolkit `mpl_toolkits.mplot3d` (which now comes standard with all mpl installs). See [mplot3d-examples-index](#) and [Getting started](#)

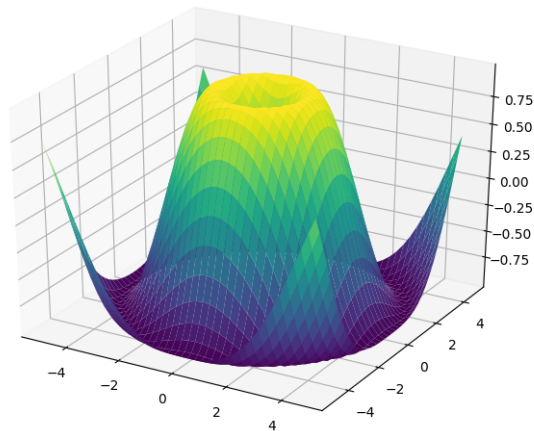


Fig. 4: Whats New 99 Mplot3d

axes grid toolkit

Jae-Joon Lee has added a new toolkit to ease displaying multiple images in matplotlib, as well as some support for curvilinear grids to support the world coordinate system. The toolkit is included standard with all new mpl installs. See [axes_grid1-examples-index](#), [axisartist-examples-index](#), [What is axes_grid1 toolkit?](#) and [axisartist](#)

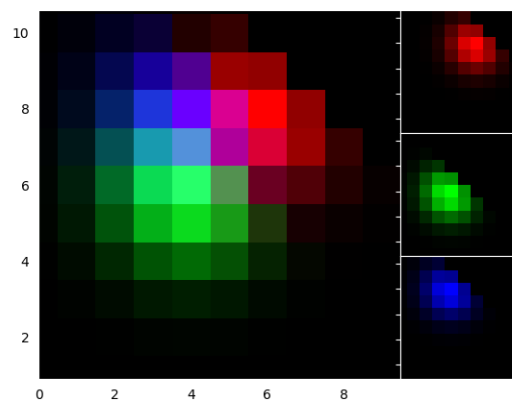


Fig. 5: Whats New 99 Axes Grid

Axis spine placement

Andrew Straw has added the ability to place "axis spines" – the lines that denote the data limits – in various arbitrary locations. No longer are your axis lines constrained to be a simple rectangle around the figure – you can turn on or off left, bottom, right and top, as well as "detach" the spine to offset it away from the data. See [/gallery/ticks_and_spines/spine_placement_demo](#) and `matplotlib.spines.Spine`.

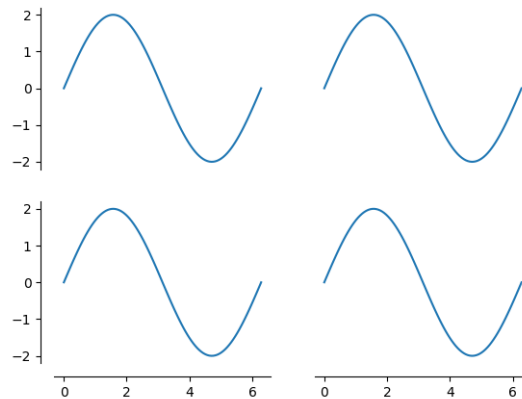


Fig. 6: Whats New 99 Spines

5.3.4 New in matplotlib 1.0

Table of Contents

- New in matplotlib 1.0
 - HTML5/Canvas backend
 - Sophisticated subplot grid layout
 - Easy pythonic subplots
 - Contour fixes and and triplot
 - multiple calls to show supported
 - mplot3d graphs can be embedded in arbitrary axes
 - tick_params
 - Lots of performance and feature enhancements
 - Much improved software carpentry
 - Bugfix marathon

HTML5/Canvas backend

Simon Ratcliffe and Ludwig Schwardt have released an [HTML5/Canvas](#) backend for matplotlib. The backend is almost feature complete, and they have done a lot of work comparing their html5 rendered images with our core renderer Agg. The backend features client/server interactive navigation of matplotlib figures in an html5 compliant browser.

Sophisticated subplot grid layout

Jae-Joon Lee has written gridspec, a new module for doing complex subplot layouts, featuring row and column spans and more. See [Customizing Figure Layouts Using GridSpec and Other Functions](#) for a tutorial overview.

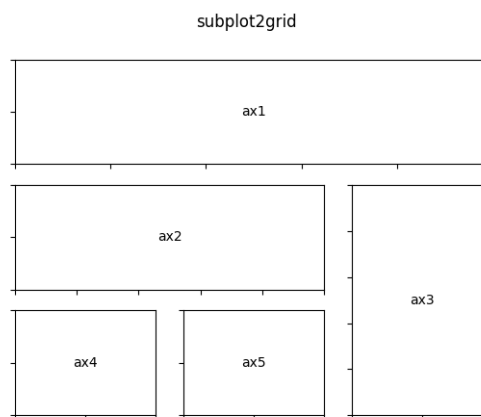


Fig. 7: Demo Gridspec01

Easy pythonic subplots

Fernando Perez got tired of all the boilerplate code needed to create a figure and multiple subplots when using the matplotlib API, and wrote a `subplots()` helper function. Basic usage allows you to create the figure and an array of subplots with numpy indexing (starts with 0). e.g.:

```
fig, axarr = plt.subplots(2, 2)
axarr[0,0].plot([1,2,3]) # upper, left
```

See [/gallery/subplots_axes_and_figures/subplot_demo](#) for several code examples.

Contour fixes and and triplot

Ian Thomas has fixed a long-standing bug that has vexed our most talented developers for years. `contourf()` now handles interior masked regions, and the boundaries of line and filled contours coincide.

Additionally, he has contributed a new module `tri` and helper function `triplot()` for creating and plotting unstructured triangular grids.

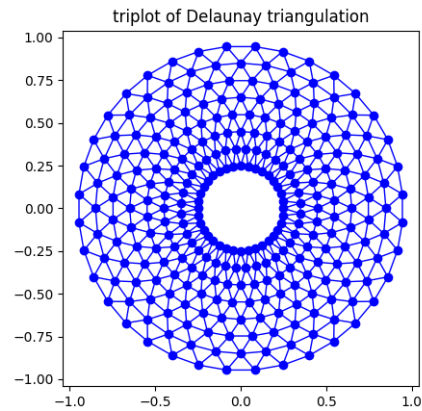


Fig. 8: Triplot Demo

multiple calls to show supported

A long standing request is to support multiple calls to `show()`. This has been difficult because it is hard to get consistent behavior across operating systems, user interface toolkits and versions. Eric Firing has done a lot of work on rationalizing show across backends, with the desired behavior to make show raise all newly created figures and block execution until they are closed. Repeated calls to show should raise newly created figures since the last call. Eric has done a lot of testing on the user interface toolkits and versions and platforms he has access to, but it is not possible to test them all, so please report problems to the [mailing list](#) and [bug tracker](#).

mplot3d graphs can be embedded in arbitrary axes

You can now place an `mplot3d` graph into an arbitrary axes location, supporting mixing of 2D and 3D graphs in the same figure, and/or multiple 3D graphs in a single figure, using the "projection" keyword argument to `add_axes` or `add_subplot`. Thanks Ben Root.

tick_params

Eric Firing wrote `tick_params`, a convenience method for changing the appearance of ticks and tick labels. See `pyplot` function `tick_params()` and associated `Axes` method `tick_params()`.

Lots of performance and feature enhancements

- Faster magnification of large images, and the ability to zoom in to a single pixel
- Local installs of documentation work better

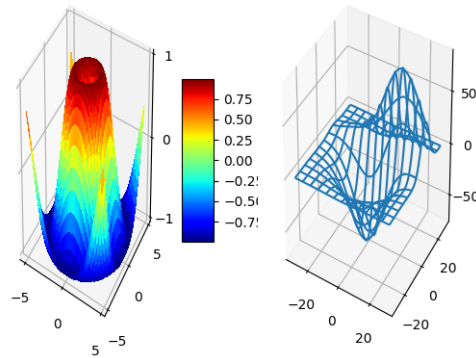


Fig. 9: Whats New 1 Subplot3d

- Improved "widgets" – mouse grabbing is supported
- More accurate snapping of lines to pixel boundaries
- More consistent handling of color, particularly the alpha channel, throughout the API

Much improved software carpentry

The matplotlib trunk is probably in as good a shape as it has ever been, thanks to improved [software carpentry](#). We now have a [buildbot](#) which runs a suite of [nose](#) regression tests on every svn commit, auto-generating a set of images and comparing them against a set of known-goods, sending emails to developers on failures with a pixel-by-pixel image comparison. Releases and release bugfixes happen in branches, allowing active new feature development to happen in the trunk while keeping the release branches stable. Thanks to Andrew Straw, Michael Droettboom and other matplotlib developers for the heavy lifting.

Bugfix marathon

Eric Firing went on a bug fixing and closing marathon, closing over 100 bugs on the [bug tracker](#) with help from Jae-Joon Lee, Michael Droettboom, Christoph Gohlke and Michiel de Hoon.

5.3.5 New in matplotlib 1.1

Table of Contents

- [New in matplotlib 1.1](#)
 - [Sankey Diagrams](#)
 - [Animation](#)

- Tight Layout
- PyQT4, PySide, and IPython
- Legend
- mplot3d
- Numerix support removed
- Markers
- Other improvements

Note: matplotlib 1.1 supports Python 2.4 to 2.7

Sankey Diagrams

Kevin Davies has extended Yannick Copin's original Sankey example into a module (sankey) and provided new examples (/gallery/api/sankey_basics, /gallery/api/sankey_links, /gallery/api/sankey_rankine).

Animation

Ryan May has written a backend-independent framework for creating animated figures. The animation module is intended to replace the backend-specific examples formerly in the examples-index listings. Examples using the new framework are in animation-examples-index; see the entrancing double pendulum [<gallery/animation/double_pendulum_sgskip.py>](/gallery/animation/double_pendulum_sgskip.py) which uses matplotlib.animation.Animation.save() to create the movie below.

This should be considered as a beta release of the framework; please try it and provide feedback.

Tight Layout

A frequent issue raised by users of matplotlib is the lack of a layout engine to nicely space out elements of the plots. While matplotlib still adheres to the philosophy of giving users complete control over the placement of plot elements, Jae-Joon Lee created the tight_layout module and introduced a new command tight_layout() to address the most common layout issues.

The usage of this functionality can be as simple as

```
plt.tight_layout()
```

and it will adjust the spacing between subplots so that the axis labels do not overlap with neighboring subplots. A [Tight Layout guide](#) has been created to show how to use this new tool.

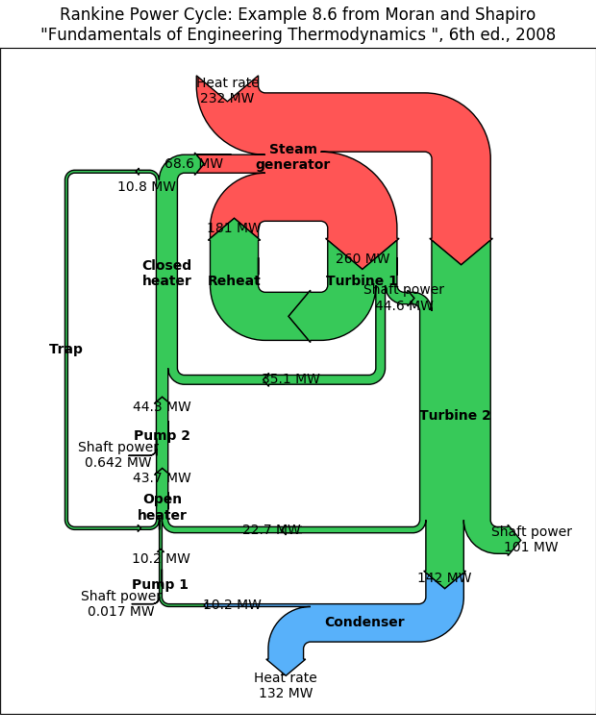
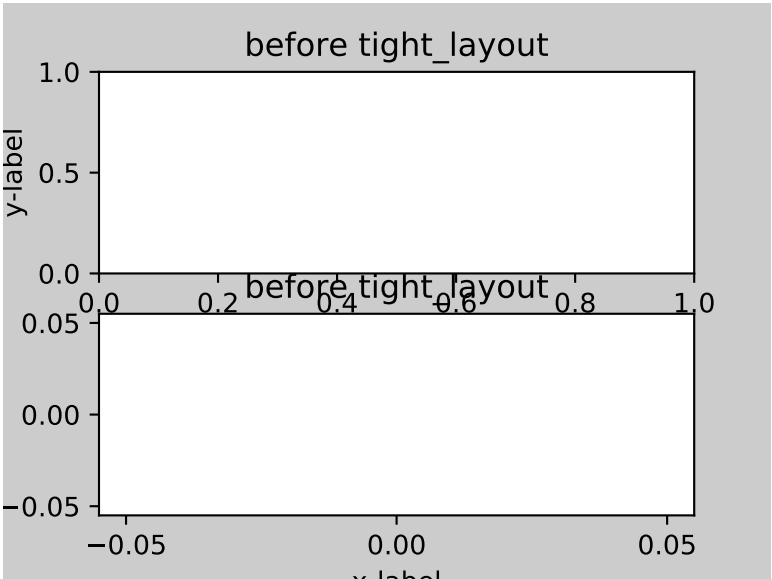
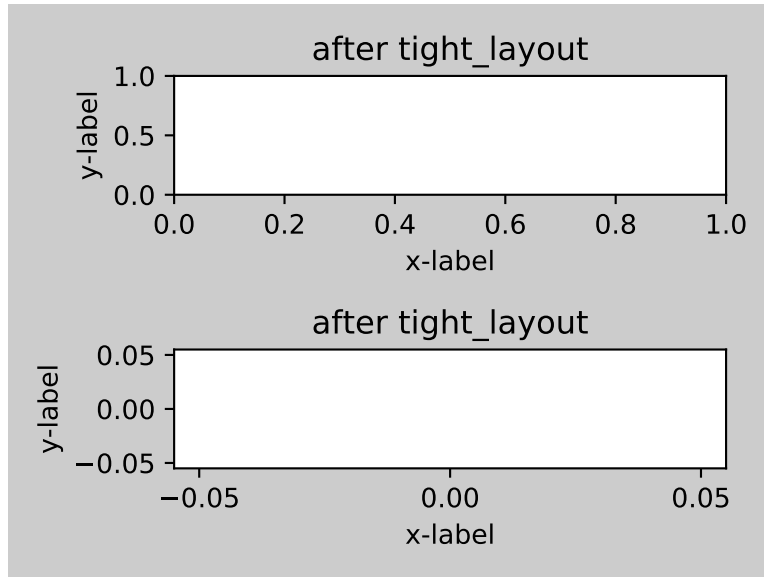


Fig. 10: Sankey Rankine





PyQT4, PySide, and IPython

Gerald Storer made the Qt4 backend compatible with PySide as well as PyQt4. At present, however, PySide does not support the `PyOS_InputHook` mechanism for handling gui events while waiting for text input, so it cannot be used with the new version 0.11 of [IPython](#). Until this feature appears in PySide, IPython users should use the PyQt4 wrapper for QT4, which remains the matplotlib default.

An rcParam entry, "backend.qt4", has been added to allow users to select PyQt4, PyQt4v2, or PySide. The latter two use the Version 2 Qt API. In most cases, users can ignore this rcParam variable; it is available to aid in testing, and to provide control for users who are embedding matplotlib in a PyQt4 or PySide app.

Legend

Jae-Joon Lee has improved plot legends. First, legends for complex plots such as `stem()` plots will now display correctly. Second, the 'best' placement of a legend has been improved in the presence of NaNs.

See the [Legend guide](#) for more detailed explanation and examples.

mplot3d

In continuing the efforts to make 3D plotting in matplotlib just as easy as 2D plotting, Ben Root has made several improvements to the mplot3d module.

- Axes3D has been improved to bring the class towards feature-parity with regular Axes objects
- Documentation for [Getting started](#) was significantly expanded
- Axis labels and orientation improved

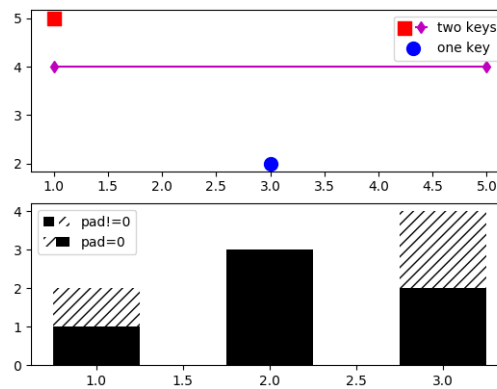


Fig. 11: Legend Demo4

- Most 3D plotting functions now support empty inputs
- Ticker offset display added:

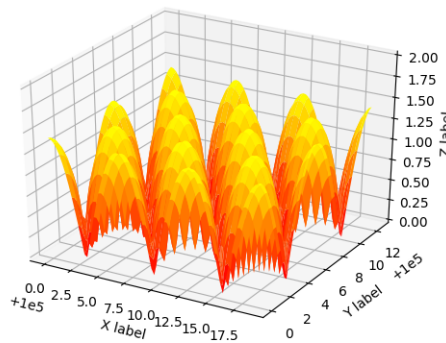


Fig. 12: Offset

- `contourf()` gains `zdir` and `offset` kwargs. You can now do this:

Numerix support removed

After more than two years of deprecation warnings, Numerix support has now been completely removed from matplotlib.

Markers

The list of available markers for `plot()` and `scatter()` has now been merged. While they were mostly similar, some markers existed for one function, but not the other. This merge did result in a conflict

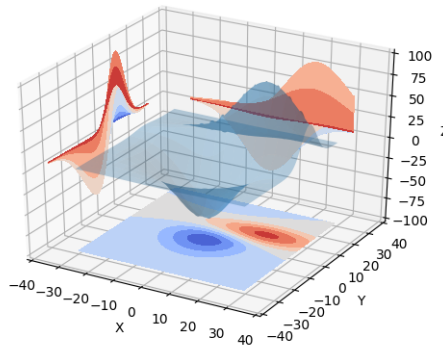


Fig. 13: Contourf3d 2

for the 'd' diamond marker. Now, 'd' will be interpreted to always mean "thin" diamond while 'D' will mean "regular" diamond.

Thanks to Michael Droettboom for this effort.

Other improvements

- Unit support for polar axes and `arrow()`
- `PolarAxes` gains getters and setters for "theta_direction", and "theta_offset" to allow for theta to go in either the clock-wise or counter-clockwise direction and to specify where zero degrees should be placed. `set_theta_zero_location()` is an added convenience function.
- Fixed error in argument handling for tri-functions such as `tripcolor()`
- `axes.labelweight` parameter added to `rcParams`.
- For `imshow()`, `interpolation='nearest'` will now always perform an interpolation. A "none" option has been added to indicate no interpolation at all.
- An error in the Hammer projection has been fixed.
- `clabel` for `contour()` now accepts a callable. Thanks to Daniel Hyams for the original patch.
- Jae-Joon Lee added the `HBox` and `VBox` classes.
- Christoph Gohlke reduced memory usage in `imshow()`.
- `scatter()` now accepts empty inputs.
- The behavior for 'symlog' scale has been fixed, but this may result in some minor changes to existing plots. This work was refined by ssyr.
- Peter Butterworth added named figure support to `figure()`.
- Michiel de Hoon has modified the MacOSX backend to make its interactive behavior consistent with the other backends.

- Pim Schellart added a new colormap called "cubehelix". Sameer Grover also added a colormap called "coolwarm". See it and all other colormaps [here](#).
- Many bug fixes and documentation improvements.

5.3.6 New in matplotlib 1.2

Table of Contents

- New in matplotlib 1.2
 - Python 3.x support
 - PGF/TikZ backend
 - Locator interface
 - Tri-Surface Plots
 - Control the lengths of colorbar extensions
 - Figures are picklable
 - Set default bounding box in matplotlibrc
 - New Boxplot Functionality
 - New RC parameter functionality
 - Streamplot
 - New hist functionality
 - Updated shipped dependencies
 - Face-centred colors in tripcolor plots
 - Hatching patterns in filled contour plots, with legends
 - Known issues in the matplotlib 1.2 release

Note: matplotlib 1.2 supports Python 2.6, 2.7, and 3.1

Python 3.x support

Matplotlib 1.2 is the first version to support Python 3.x, specifically Python 3.1 and 3.2. To make this happen in a reasonable way, we also had to drop support for Python versions earlier than 2.6.

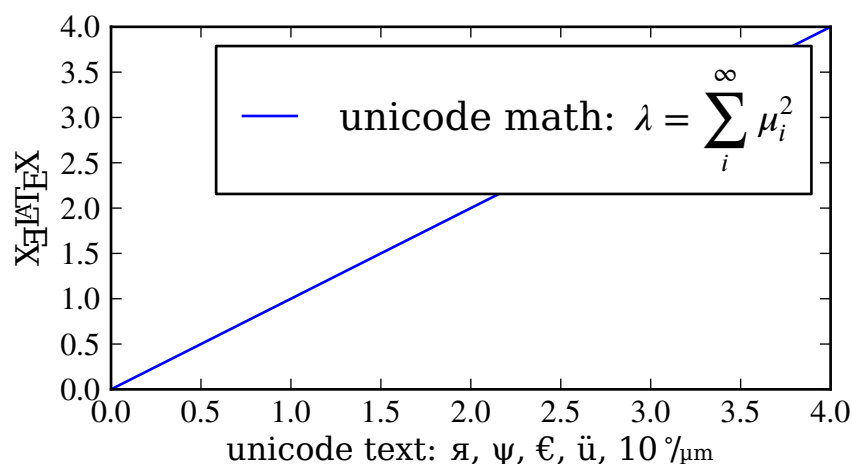
This work was done by Michael Droettboom, the Cape Town Python Users' Group, many others and supported financially in part by the SAGE project.

The following GUI backends work under Python 3.x: Gtk3Cairo, Qt4Agg, TkAgg and MacOSX. The other GUI backends do not yet have adequate bindings for Python 3.x, but continue to work on Python 2.6 and 2.7, particularly the Qt and QtAgg backends (which have been deprecated). The non-GUI backends, such as PDF, PS and SVG, work on both Python 2.x and 3.x.

Features that depend on the Python Imaging Library, such as JPEG handling, do not work, since the version of PIL for Python 3.x is not sufficiently mature.

PGF/TikZ backend

Peter Würtz wrote a backend that allows matplotlib to export figures as drawing commands for LaTeX. These can be processed by PdfLaTeX, XeLaTeX or LuaLaTeX using the PGF/TikZ package. Usage examples and documentation are found in [Typesetting With XeLaTeX/LuaLaTeX](#).



Locator interface

Philip Elson exposed the intelligence behind the tick Locator classes with a simple interface. For instance, to get no more than 5 sensible steps which span the values 10 and 19.5:

```
>>> import matplotlib.ticker as mticker
>>> locator = mticker.MaxNLocator(nbins=5)
>>> print(locator.tick_values(10, 19.5))
[ 10.  12.  14.  16.  18.  20.]
```

Tri-Surface Plots

Damon McDougall added a new plotting method for the mplot3d toolkit called `plot_trisurf()`.

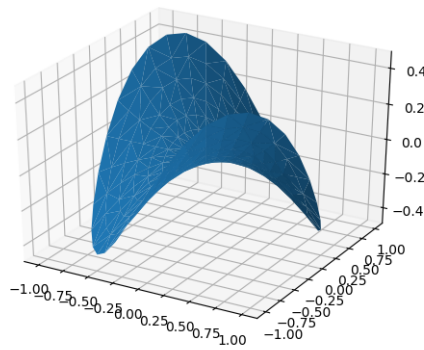


Fig. 14: Trisurf3d

Control the lengths of colorbar extensions

Andrew Dawson added a new keyword argument `extendfrac` to `colorbar()` to control the length of minimum and maximum colorbar extensions.

Figures are picklable

Philip Elson added an experimental feature to make figures picklable for quick and easy short-term storage of plots. Pickle files are not designed for long term storage, are unsupported when restoring a pickle saved in another matplotlib version and are insecure when restoring a pickle from an untrusted source. Having said this, they are useful for short term storage for later modification inside matplotlib.

Set default bounding box in matplotlibrc

Two new defaults are available in the `matplotlibrc` configuration file: `savefig.bbox`, which can be set to 'standard' or 'tight', and `savefig.pad_inches`, which controls the bounding box padding.

New Boxplot Functionality

Users can now incorporate their own methods for computing the median and its confidence intervals into the `boxplot()` method. For every column of data passed to `boxplot`, the user can specify an accompanying median and confidence interval.

New RC parameter functionality

Matthew Emmett added a function and a context manager to help manage RC parameters: `rc_file()` and `rc_context`. To load RC parameters from a file:

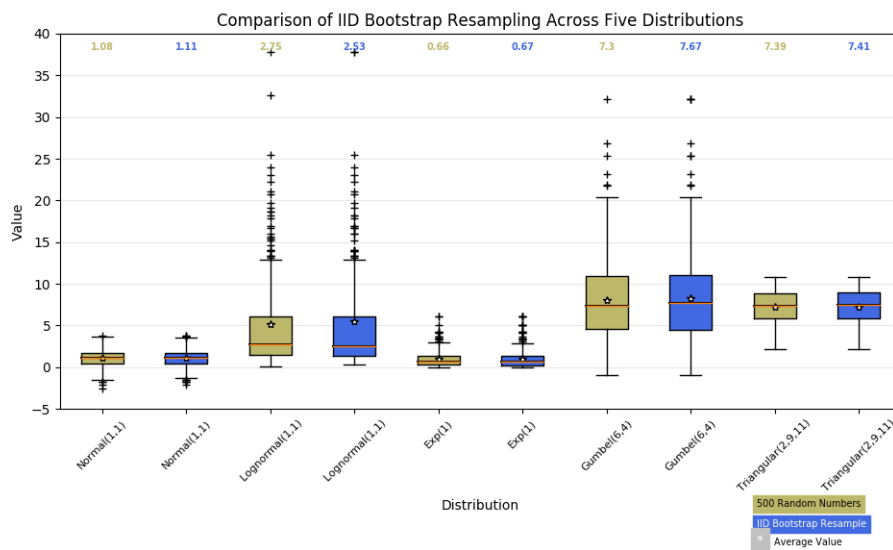
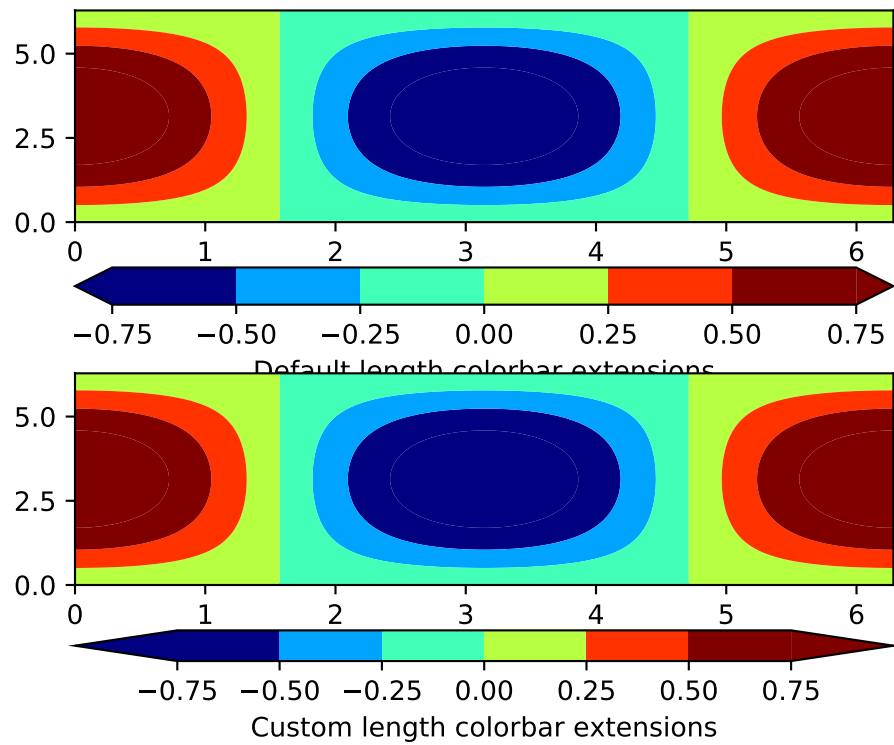


Fig. 15: Boxplot Demo3


```
>>> mpl.rc_file('mpl.rc')
```

To temporarily use RC parameters:

```
>>> with mpl.rc_context(fname='mpl.rc', rc={'text.usetex': True}):
>>> ...
```

Streamplot

Tom Flannaghan and Tony Yu have added a new `streamplot()` function to plot the streamlines of a vector field. This has been a long-requested feature and complements the existing `quiver()` function for plotting vector fields. In addition to simply plotting the streamlines of the vector field, `streamplot()` allows users to map the colors and/or line widths of the streamlines to a separate parameter, such as the speed or local intensity of the vector field.

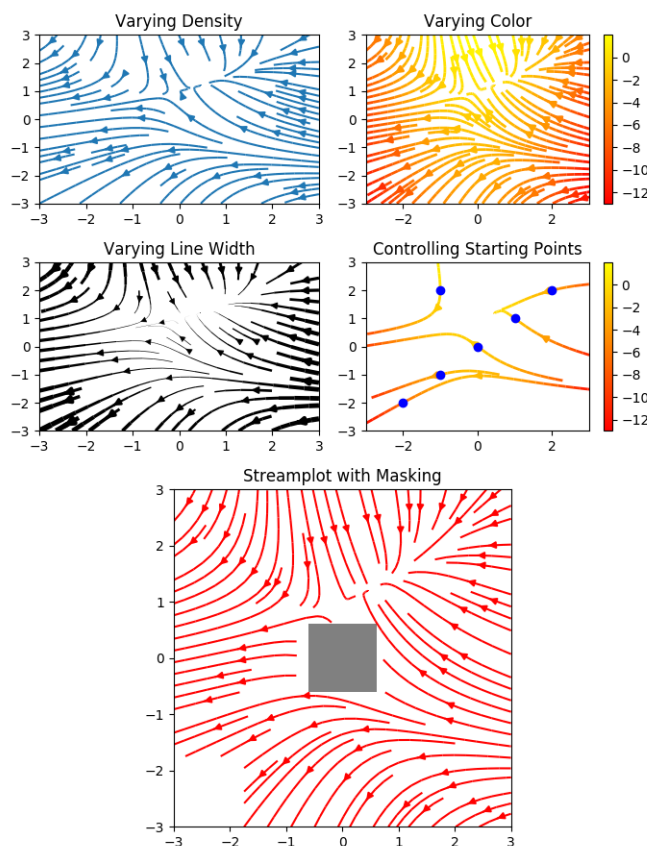


Fig. 16: Plot Streamplot

New hist functionality

Nic Eggert added a new stacked kwarg to `hist()` that allows creation of stacked histograms using any of the histogram types. Previously, this functionality was only available by using the `barstacked`

histogram type. Now, when `stacked=True` is passed to the function, any of the histogram types can be stacked. The `barstacked` histogram type retains its previous functionality for backwards compatibility.

Updated shipped dependencies

The following dependencies that ship with matplotlib and are optionally installed alongside it have been updated:

- `pytz` 2012d
- `dateutil` 1.5 on Python 2.x, and 2.1 on Python 3.x

Face-centred colors in tripcolor plots

Ian Thomas extended `tripcolor()` to allow one color value to be specified for each triangular face rather than for each point in a triangulation.

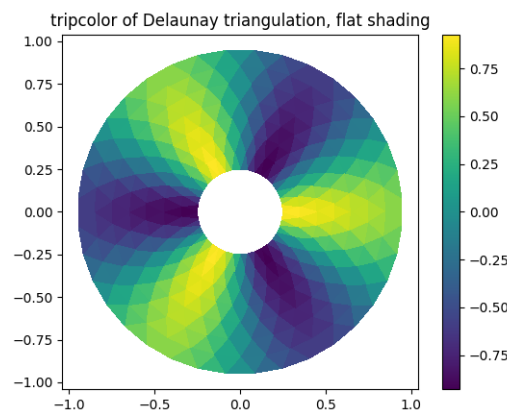


Fig. 17: Tripcolor Demo

Hatching patterns in filled contour plots, with legends

Phil Elson added support for hatching to `contourf()`, together with the ability to use a legend to identify contoured ranges.

Known issues in the matplotlib 1.2 release

- When using the `Qt4Agg` backend with `IPython` 0.11 or later, the save dialog will not display. This should be fixed in a future version of `IPython`.

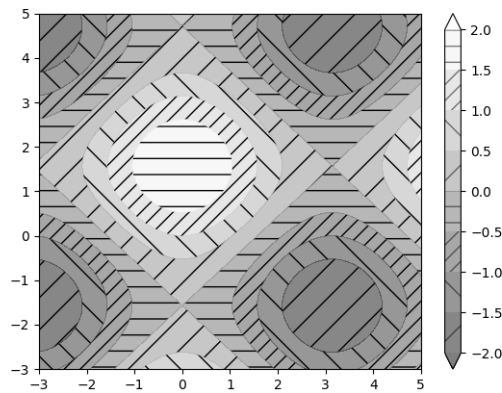


Fig. 18: Contourf Hatching

5.3.7 New in matplotlib 1.2.2

Table of Contents

- [New in matplotlib 1.2.2](#)
 - [Improved collections](#)
 - [Multiple images on same axes are correctly transparent](#)

Improved collections

The individual items of a collection may now have different alpha values and be rendered correctly. This also fixes a bug where collections were always filled in the PDF backend.

Multiple images on same axes are correctly transparent

When putting multiple images onto the same axes, the background color of the axes will now show through correctly.

5.3.8 New in matplotlib 1.3

Table of Contents

- [New in matplotlib 1.3](#)
 - [New in 1.3.1](#)

- New plotting features
- Updated Axes3D.contour methods
- Drawing
- Text
- Configuration (rcParams)
- Backends
- Documentation and examples
- Infrastructure

Note: matplotlib 1.3 supports Python 2.6, 2.7, 3.2, and 3.3

New in 1.3.1

1.3.1 is a bugfix release, primarily dealing with improved setup and handling of dependencies, and correcting and enhancing the documentation.

The following changes were made in 1.3.1 since 1.3.0.

Enhancements

- Added a context manager for creating multi-page pdfs (see `matplotlib.backends.backend_pdf.PdfPages`).
- The WebAgg backend should now have lower latency over heterogeneous Internet connections.

Bug fixes

- Histogram plots now contain the endline.
- Fixes to the Mollweide projection.
- Handling recent fonts from Microsoft and Macintosh-style fonts with non-ascii metadata is improved.
- Hatching of fill between plots now works correctly in the PDF backend.
- Tight bounding box support now works in the PGF backend.
- Transparent figures now display correctly in the Qt4Agg backend.
- Drawing lines from one subplot to another now works.
- Unit handling on masked arrays has been improved.

Setup and dependencies

- Now works with any version of `pyarsing` 1.5.6 or later, without displaying hundreds of warnings.
- Now works with 64-bit versions of Ghostscript on MS-Windows.
- When installing from source into an environment without Numpy, Numpy will first be downloaded and built and then used to build matplotlib.
- Externally installed backends are now always imported using a fully-qualified path to the module.
- Works with newer version of wxPython.
- Can now build with a PyCXX installed globally on the system from source.
- Better detection of Gtk3 dependencies.

Testing

- Tests should now work in non-English locales.
- PEP8 conformance tests now report on locations of issues.

New plotting features

xkcd-style sketch plotting

To give your plots a sense of authority that they may be missing, Michael Droettboom (inspired by the work of many others in [PR #1329](#)) has added an `xkcd-style` sketch plotting mode. To use it, simply call `matplotlib.pyplot.xkcd()` before creating your plot. For really fine control, it is also possible to modify each artist's sketch parameters individually with `matplotlib.artist.Artist.set_sketch_params()`.

Updated `Axes3D.contour` methods

Damon McDougall updated the `tricontour()` and `tricontourf()` methods to allow 3D contour plots on arbitrary unstructured user-specified triangulations.

New `eventplot` plot type

Todd Jennings added a `eventplot()` function to create multiple rows or columns of identical line segments

As part of this feature, there is a new `EventCollection` class that allows for plotting and manipulating rows or columns of identical line segments.

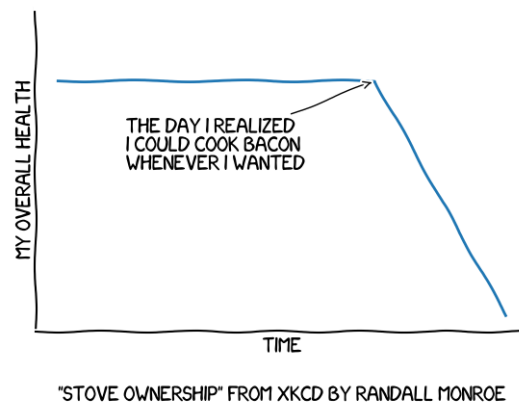


Fig. 19: Xkcd

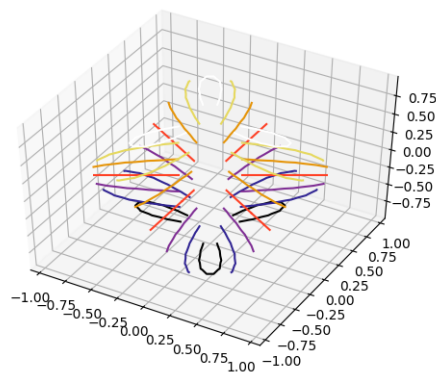


Fig. 20: Tricontour3d

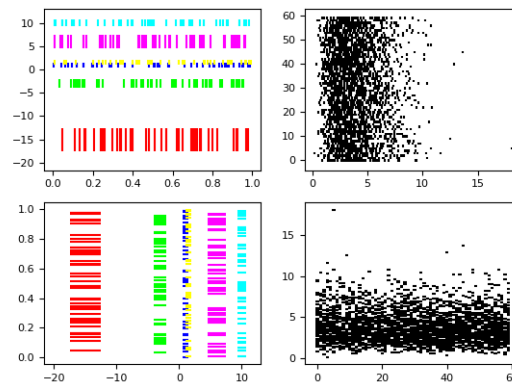


Fig. 21: Eventplot Demo

Triangular grid interpolation

Geoffroy Billotey and Ian Thomas added classes to perform interpolation within triangular grids: (LinearTriInterpolator and CubicTriInterpolator) and a utility class to find the triangles in which points lie (TrapezoidMapTriFinder). A helper class to perform mesh refinement and smooth contouring was also added (UniformTriRefiner). Finally, a class implementing some basic tools for triangular mesh improvement was added (TriAnalyzer).

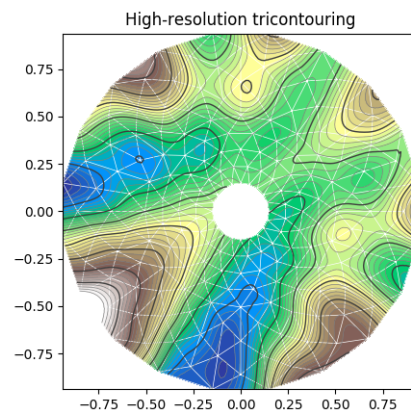


Fig. 22: Tricontour Smooth User

Baselines for stackplot

Till Stensitzki added non-zero baselines to `stackplot()`. They may be symmetric or weighted.

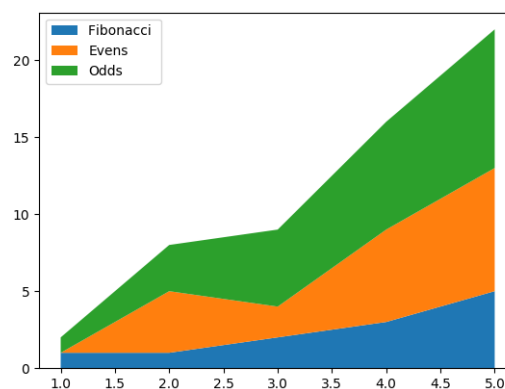


Fig. 23: Stackplot Demo2

Rectangular colorbar extensions

Andrew Dawson added a new keyword argument `extendrect` to `colorbar()` to optionally make colorbar extensions rectangular instead of triangular.

More robust boxplots

Paul Hobson provided a fix to the `boxplot()` method that prevent whiskers from being drawn inside the box for oddly distributed data sets.

Calling `subplot()` without arguments

A call to `subplot()` without any arguments now acts the same as `subplot(111)` or `subplot(1,1,1)` – it creates one axes for the whole figure. This was already the behavior for both `axes()` and `subplots()`, and now this consistency is shared with `subplot()`.

Drawing

Independent alpha values for face and edge colors

Wes Campaigne modified how Patch objects are drawn such that (for backends supporting transparency) you can set different alpha values for faces and edges, by specifying their colors in RGBA format. Note that if you set the alpha attribute for the patch object (e.g. using `set_alpha()` or the `alpha` keyword argument), that value will override the alpha components set in both the face and edge colors.

Path effects on lines

Thanks to Jae-Joon Lee, path effects now also work on plot lines.

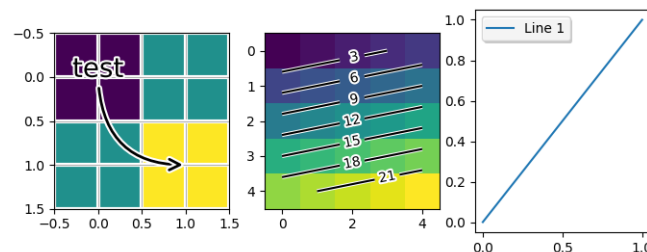


Fig. 24: Patheffect Demo

Easier creation of colormap and normalizer for levels with colors

Phil Elson added the `matplotlib.colors.from_levels_and_colors()` function to easily create a colormap and normalizer for representation of discrete colors for plot types such as `matplotlib.pyplot.pcolormesh()`, with a similar interface to that of `contourf()`.

Full control of the background color

Wes Campaigne and Phil Elson fixed the Agg backend such that PNGs are now saved with the correct background color when `fig.patch.get_alpha()` is not 1.

Improved `bbox_inches="tight"` functionality

Passing `bbox_inches="tight"` through to `plt.save()` now takes into account all artists on a figure - this was previously not the case and led to several corner cases which did not function as expected.

Initialize a rotated rectangle

Damon McDougall extended the `Rectangle` constructor to accept an `angle` kwarg, specifying the rotation of a rectangle in degrees.

Text

Anchored text support

The `svg` and `pgf` backends are now able to save text alignment information to their output formats. This allows to edit text elements in saved figures, using Inkscape for example, while preserving their intended position. For `svg` please note that you'll have to disable the default text-to-path conversion (`mpl.rc('svg', fonttype='none')`).

Better vertical text alignment and multi-line text

The vertical alignment of text is now consistent across backends. You may see small differences in text placement, particularly with rotated text.

If you are using a custom backend, note that the `draw_text` renderer method is now passed the location of the baseline, not the location of the bottom of the text bounding box.

Multi-line text will now leave enough room for the height of very tall or very low text, such as superscripts and subscripts.

Left and right side axes titles

Andrew Dawson added the ability to add axes titles flush with the left and right sides of the top of the axes using a new keyword argument `loc` to `title()`.

Improved manual contour plot label positioning

Brian Mattern modified the manual contour plot label positioning code to interpolate along line segments and find the actual closest point on a contour to the requested position. Previously, the closest path vertex was used, which, in the case of straight contours was sometimes quite distant from the requested location. Much more precise label positioning is now possible.

Configuration (rcParams)

Quickly find rcParams

Phil Elson made it easier to search for rcParameters by passing a valid regular expression to `matplotlib.RcParams.find_all()`. `matplotlib.RcParams` now also has a pretty repr and str representation so that search results are printed prettily:

```
>>> import matplotlib
>>> print(matplotlib.rcParams.find_all('\.size'))
RcParams({'font.size': 12,
          'xtick.major.size': 4,
          'xtick.minor.size': 2,
          'ytick.major.size': 4,
          'ytick.minor.size': 2})
```

axes.xmargin and axes.ymargin added to rcParams

rcParam values (`axes.xmargin` and `axes.ymargin`) were added to configure the default margins used. Previously they were hard-coded to default to 0, default value of both rcParam values is 0.

Changes to font rcParams

The `font.*` rcParams now affect only text objects created after the rcParam has been set, and will not retroactively affect already existing text objects. This brings their behavior in line with most other rcParams.

savefig.jpeg_quality added to rcParams

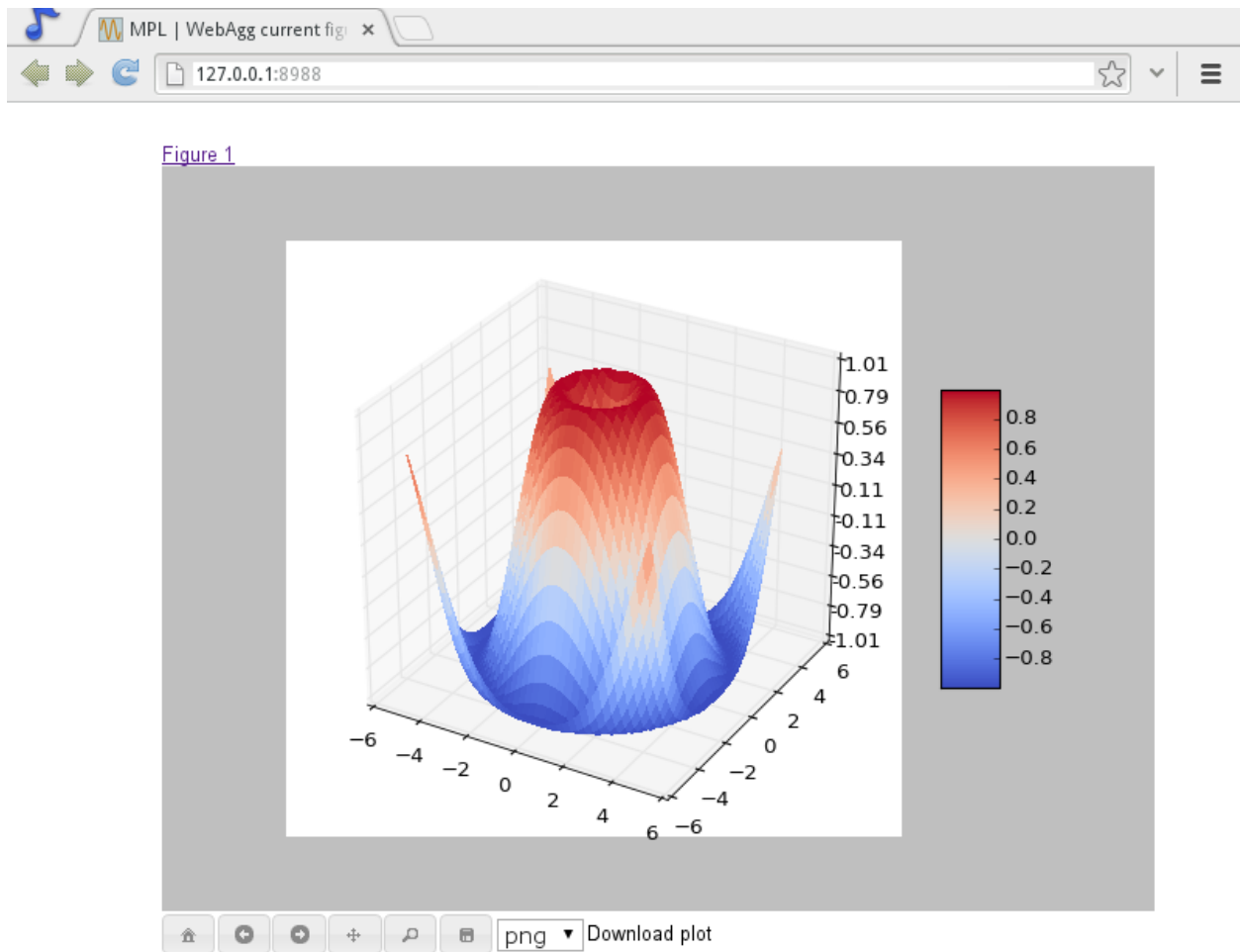
rcParam value `savefig.jpeg_quality` was added so that the user can configure the default quality used when a figure is written as a JPEG. The default quality is 95; previously, the default quality

was 75. This change minimizes the artifacting inherent in JPEG images, particularly with images that have sharp changes in color as plots often do.

Backends

WebAgg backend

Michael Droettboom, Phil Elson and others have developed a new backend, WebAgg, to display figures in a web browser. It works with animations as well as being fully interactive.



Future versions of matplotlib will integrate this backend with the IPython notebook for a fully web browser based plotting frontend.

Remember save directory

Martin Spacek made the save figure dialog remember the last directory saved to. The default is configurable with the new `savefig.directory` rcParam in matplotlibrc.

Documentation and examples

Numpydoc docstrings

Nelle Varoquaux has started an ongoing project to convert matplotlib's docstrings to numpydoc format. See [MEP10](#) for more information.

Example reorganization

Tony Yu has begun work reorganizing the examples into more meaningful categories. The new gallery page is the fruit of this ongoing work. See [MEP12](#) for more information.

Examples now use subplots()

For the sake of brevity and clarity, most of the examples now use the newer `subplots()`, which creates a figure and one (or multiple) axes object(s) in one call. The old way involved a call to `figure()`, followed by one (or multiple) `subplot()` calls.

Infrastructure

Housecleaning

A number of features that were deprecated in 1.2 or earlier, or have not been in a working state for a long time have been removed. Highlights include removing the Qt version 3 backends, and the FltkAgg and Emf backends. See [Changes in 1.3.x](#) for a complete list.

New setup script

matplotlib 1.3 includes an entirely rewritten setup script. We now ship fewer dependencies with the tarballs and installers themselves. Notably, `pytz`, `dateutil`, `pyparsing` and `six` are no longer included with matplotlib. You can either install them manually first, or let pip install them as dependencies along with matplotlib. It is now possible to not include certain subcomponents, such as the unit test data, in the install. See `setup.cfg.template` for more information.

XDG base directory support

On Linux, matplotlib now uses the XDG base directory specification to find the `matplotlibrc` configuration file. `matplotlibrc` should now be kept in `config/matplotlib`, rather than `matplotlib`. If your configuration is found in the old location, it will still be used, but a warning will be displayed.

Catch opening too many figures using pyplot

Figures created through `pyplot.figure` are retained until they are explicitly closed. It is therefore common for new users of matplotlib to run out of memory when creating a large series of figures in a loop without closing them.

matplotlib will now display a `RuntimeWarning` when too many figures have been opened at once. By default, this is displayed for 20 or more figures, but the exact number may be controlled using the `figure.max_open_warning` rcParam.

5.3.9 New in matplotlib 1.4

Thomas A. Caswell served as the release manager for the 1.4 release.

Table of Contents

- New in matplotlib 1.4
 - New colormap
 - The nbagg backend
 - New plotting features
 - Date handling
 - Configuration (rcParams)
 - style package added
 - Backends
 - Text
 - Sphinx extensions
 - Legend and PathEffects documentation
 - Widgets
 - GAE integration

Note: matplotlib 1.4 supports Python 2.6, 2.7, 3.3, and 3.4

New colormap

In heatmaps, a green-to-red spectrum is often used to indicate intensity of activity, but this can be problematic for the red/green colorblind. A new, colorblind-friendly colormap is now available at matplotlib.cm.wistia.com. This colormap maintains the red/green symbolism while achieving deuteranopic legibility through brightness variations. See [here](#) for more information.

The nbagg backend

Phil Elson added a new backend, named "nbagg", which enables interactive figures in a live IPython notebook session. The backend makes use of the infrastructure developed for the webagg backend, which itself gives standalone server backed interactive figures in the browser, however nbagg does not require a dedicated matplotlib server as all communications are handled through the IPython Comm machinery.

As with other backends nbagg can be enabled inside the IPython notebook with:

```
import matplotlib
matplotlib.use('nbagg')
```

Once figures are created and then subsequently shown, they will be placed in an interactive widget inside the notebook allowing panning and zooming in the same way as any other matplotlib backend. Because figures require a connection to the IPython notebook server for their interactivity, once the notebook is saved, each figure will be rendered as a static image - thus allowing non-interactive viewing of figures on services such as [nbviewer](#).

New plotting features

Power-law normalization

Ben Gamari added a power-law normalization method, `PowerNorm`. This class maps a range of values to the interval $[0,1]$ with power-law scaling with the exponent provided by the constructor's `gamma` argument. Power law normalization can be useful for, e.g., emphasizing small populations in a histogram.

Fully customizable boxplots

Paul Hobson overhauled the `boxplot()` method such that it is now completely customizable in terms of the styles and positions of the individual artists. Under the hood, `boxplot()` relies on a new function (`boxplot_stats()`), which accepts any data structure currently compatible with `boxplot()`, and returns a list of dictionaries containing the positions for each element of the boxplots. Then a second method, `bxp()` is called to draw the boxplots based on the stats.

The `boxplot()` function can be used as before to generate boxplots from data in one step. But now the user has the flexibility to generate the statistics independently, or to modify the output of `boxplot_stats()` prior to plotting with `bxp()`.

Lastly, each artist (e.g., the box, outliers, cap, notches) can now be toggled on or off and their styles can be passed in through individual kwargs. See the examples: [/gallery/statistics/boxplot](#) and [/gallery/statistics/bxp](#)

Added a bool kwarg, `manage_xticks`, which if `False` disables the management of the ticks and limits on the x-axis by `bxp()`.

Support for datetime axes in 2d plots

Andrew Dawson added support for datetime axes to `contour()`, `contourf()`, `pcolormesh()` and `pcolor()`.

Support for additional spectrum types

Todd Jennings added support for new types of frequency spectrum plots: `magnitude_spectrum()`, `phase_spectrum()`, and `angle_spectrum()`, as well as corresponding functions in `mlab`.

He also added these spectrum types to `specgram()`, as well as adding support for linear scaling there (in addition to the existing dB scaling). Support for additional spectrum types was also added to `specgram()`.

He also increased the performance for all of these functions and plot types.

Support for detrending and windowing 2D arrays in mlab

Todd Jennings added support for 2D arrays in the `detrend_mean()`, `detrend_none()`, and `detrend()`, as well as adding `apply_window()` which support windowing 2D arrays.

Support for strides in mlab

Todd Jennings added some functions to `mlab` to make it easier to use numpy strides to create memory-efficient 2D arrays. This includes `stride_repeat()`, which repeats an array to create a 2D array, and `stride_windows()`, which uses a moving window to create a 2D array from a 1D array.

Formatter for new-style formatting strings

Added `FormatStrFormatterNewStyle` which does the same job as `FormatStrFormatter`, but accepts new-style formatting strings instead of printf-style formatting strings

Consistent grid sizes in streamplots

`streamplot()` uses a base grid size of 30x30 for both `density=1` and `density=(1, 1)`. Previously a grid size of 30x30 was used for `density=1`, but a grid size of 25x25 was used for `density=(1, 1)`.

Get a list of all tick labels (major and minor)

Added the kwarg `'which'` to `get_xticklabels()`, `get_yticklabels()` and `get_ticklabels()`. `'which'` can be `'major'`, `'minor'`, or `'both'` select which ticks to return, like `set_ticks_position()`. If `'which'` is `None` then the old behaviour (controlled by the bool `minor`).

Separate horizontal/vertical axes padding support in ImageGrid

The kwarg `'axes_pad'` to `mpl_toolkits.axes_grid1.ImageGrid` can now be a tuple if separate horizontal/vertical padding is needed. This is supposed to be very helpful when you have a labelled legend next to every subplot and you need to make some space for legend's labels.

Support for skewed transformations

The `Affine2D` gained additional methods `skew` and `skew_deg` to create skewed transformations. Additionally, matplotlib internals were cleaned up to support using such transforms in `Axes`. This transform is important for some plot types, specifically the Skew-T used in meteorology.

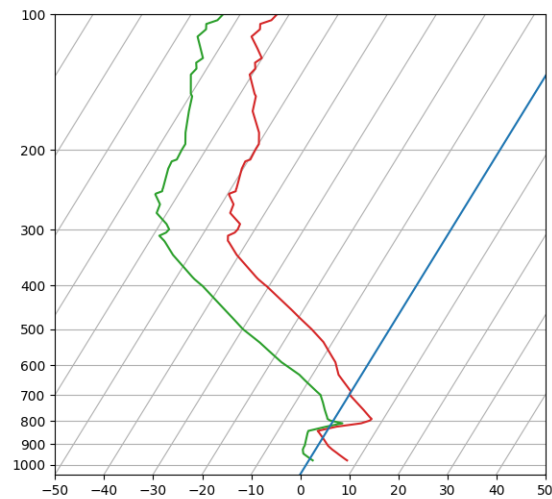


Fig. 25: Skewt

Support for specifying properties of wedge and text in pie charts.

Added the kwargs `'wedgeprops'` and `'textprops'` to `pie()` to accept properties for wedge and text objects in a pie. For example, one can specify `wedgeprops = {'linewidth':3}` to specify the width of the borders of the wedges in the pie. For more properties that the user can specify, look at the docs for the wedge and text objects.

Fixed the direction of errorbar upper/lower limits

Larry Bradley fixed the `errorbar()` method such that the upper and lower limits (`lolims`, `uplims`, `xlolims`, `xuplims`) now point in the correct direction.

More consistent add-object API for Axes

Added the Axes method `add_image` to put image handling on a par with artists, collections, containers, lines, patches, and tables.

Violin Plots

Per Parker, Gregory Kelsie, Adam Ortiz, Kevin Chan, Geoffrey Lee, Deokjae Donald Seo, and Taesu Terry Lim added a basic implementation for violin plots. Violin plots can be used to represent the distribution of sample data. They are similar to box plots, but use a kernel density estimation function to present a smooth approximation of the data sample used. The added features are:

`violin()` - Renders a violin plot from a collection of statistics. `violin_stats()` - Produces a collection of statistics suitable for rendering a violin plot. `violinplot()` - Creates a violin plot from a set of sample data. This method makes use of `violin_stats()` to process the input data, and `violin_stats()` to do the actual rendering. Users are also free to modify or replace the output of `violin_stats()` in order to customize the violin plots to their liking.

This feature was implemented for a software engineering course at the University of Toronto, Scarborough, run in Winter 2014 by Anya Taffiovich.

More `markevery` options to show only a subset of markers

Rohan Walker extended the `markevery` property in `Line2D`. You can now specify a subset of markers to show with an int, slice object, numpy fancy indexing, or float. Using a float shows markers at approximately equal display-coordinate-distances along the line.

Added size related functions to specialized Collections

Added the `get_size` and `set_size` functions to control the size of elements of specialized collections (`AsteriskPolygonCollection` `BrokenBarHCollection` `CircleCollection` `PathCollection` `PolyCollection` `RegularPolyCollection` `StarPolygonCollection`).

Fixed the mouse coordinates giving the wrong theta value in Polar graph

Added code to `transform_non_affine()` to ensure that the calculated theta value was between the range of 0 and $2 * \pi$ since the problem was that the value can become negative after applying the direction and rotation to the theta calculation.

Simple quiver plot for `mplot3d` toolkit

A team of students in an Engineering Large Software Systems course, taught by Prof. Anya Taffiovich at the University of Toronto, implemented a simple version of a quiver plot in 3D space for the `mplot3d` toolkit as one of their term project. This feature is documented in `quiver()`. The

team members are: Ryan Steve D'Souza, Victor B, xbtsw, Yang Wang, David, Caradec Bisesar and Vlad Vassilovski.

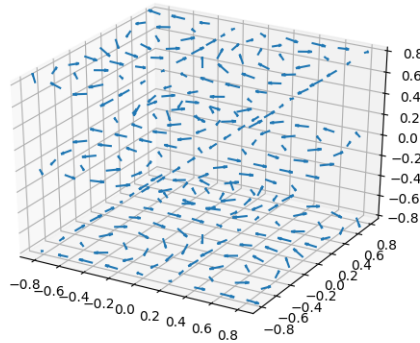


Fig. 26: Quiver3d

polar-plot r-tick locations

Added the ability to control the angular position of the r-tick labels on a polar plot via `set_rlabel_position()`.

Date handling

n-d array support for date conversion

Andrew Dawson added support for n-d array handling to `matplotlib.dates.num2date()`, `matplotlib.dates.date2num()` and `matplotlib.dates.datestr2num()`. Support is also added to the unit conversion interfaces `matplotlib.dates.DateConverter` and `matplotlib.units.Registry`.

Configuration (rcParams)

`savefig.transparent` added

Controls whether figures are saved with a transparent background by default. Previously `savefig` always defaulted to a non-transparent background.

`axes.titleweight`

Added rcParam to control the weight of the title

`axes.formatter.useoffset` added

Controls the default value of `useOffset` in `ScalarFormatter`. If `True` and the data range is much smaller than the data average, then an offset will be determined such that the tick labels are meaningful. If `False` then the full number will be formatted in all conditions.

`nbagg.transparent` added

Controls whether `nbagg` figures have a transparent background. `nbagg.transparent` is `True` by default.

XDG compliance

Matplotlib now looks for configuration files (both `rcparams` and `style`) in XDG compliant locations.

`style` package added

You can now easily switch between different styles using the new `style` package:

```
>>> from matplotlib import style
>>> style.use('dark_background')
```

Subsequent plots will use updated colors, sizes, etc. To list all available styles, use:

```
>>> print style.available
```

You can add your own custom `<style name>.mplstyle` files to `~/.matplotlib/stylelib` or call `use` with a URL pointing to a file with `matplotlibrc` settings.

Note that this is an experimental feature, and the interface may change as users test out this new feature.

Backends

Qt5 backend

Martin Fitzpatrick and Tom Badran implemented a Qt5 backend. The differences in namespace locations between Qt4 and Qt5 was dealt with by shimming Qt4 to look like Qt5, thus the Qt5 implementation is the primary implementation. Backwards compatibility for Qt4 is maintained by wrapping the Qt5 implementation.

The Qt5Agg backend currently does not work with IPython's `%matplotlib` magic.

The 1.4.0 release has a known bug where the toolbar is broken. This can be fixed by:

```
cd path/to/installed/matplotlib
wget https://github.com/matplotlib/matplotlib/pull/3322.diff
# unix2dos 3322.diff (if on windows to fix line endings)
patch -p2 < 3322.diff
```

Qt4 backend

Rudolf Höfler changed the appearance of the subplottool. All sliders are vertically arranged now, buttons for tight layout and reset were added. Furthermore, the subplottool is now implemented as a modal dialog. It was previously a QMainWindow, leaving the SPT open if one closed the plot window.

In the figure options dialog one can now choose to (re-)generate a simple automatic legend. Any explicitly set legend entries will be lost, but changes to the curves' label, linestyle, et cetera will now be updated in the legend.

Interactive performance of the Qt4 backend has been dramatically improved under windows.

The mapping of key-signals from Qt to values matplotlib understands was greatly improved (For both Qt4 and Qt5).

Cairo backends

The Cairo backends are now able to use the [cairoffi bindings](#) which are more actively maintained than the [pycairo bindings](#).

Gtk3Agg backend

The Gtk3Agg backend now works on Python 3.x, if the [cairoffi bindings](#) are installed.

PDF backend

Added context manager for saving to multi-page PDFs.

Text

Text URLs supported by SVG backend

The svg backend will now render Text objects' url as a link in output SVGs. This allows one to make clickable text in saved figures using the url kwarg of the Text class.

Anchored sizebar font

Added the fontproperties kwarg to AnchoredSizeBar to control the font properties.

Sphinx extensions

The `:context:` directive in the `plot_directive` Sphinx extension can now accept an optional `reset` setting, which will cause the context to be reset. This allows more than one distinct context to be present in documentation. To enable this option, use `:context: reset` instead of `:context:` any time you want to reset the context.

Legend and PathEffects documentation

The [Legend guide](#) and [Path effects guide](#) have both been updated to better reflect the full potential of each of these powerful features.

Widgets

Span Selector

Added an option `span_stays` to the `SpanSelector` which makes the selector rectangle stay on the axes after you release the mouse.

GAE integration

Matplotlib will now run on google app engine.

5.3.10 New in matplotlib 1.5

Table of Contents

- New in matplotlib 1.5
 - Interactive OO usage
 - Working with labeled data like pandas DataFrames
 - Added `axes.prop_cycle` key to `rcParams`
 - New Colormaps
 - Styles
 - Backends
 - Configuration (`rcParams`)

- Widgets
- New plotting features
- ToolManager
- `cbook.is_sequence_of_strings` recognizes string objects
- New `close-figs` argument for plot directive
- Support for URL string arguments to `imread`
- Display hook for animations in the IPython notebook
- Prefixed `pkg-config` for building

Note: matplotlib 1.5 supports Python 2.7, 3.4, and 3.5

Interactive OO usage

All Artists now keep track of if their internal state has been changed but not reflected in the display ('stale') by a call to `draw`. It is thus possible to pragmatically determine if a given Figure needs to be re-drawn in an interactive session.

To facilitate interactive usage a `draw_all` method has been added to pyplot which will redraw all of the figures which are 'stale'.

To make this convenient for interactive use matplotlib now registers a function either with IPython's 'post_execute' event or with the displayhook in the standard python REPL to automatically call `plt.draw_all` just before control is returned to the REPL. This ensures that the draw command is deferred and only called once.

The upshot of this is that for interactive backends (including `%matplotlib notebook`) in interactive mode (with `plt.ion()`)

```
In [1]: import matplotlib.pyplot as plt
```

```
In [2]: fig, ax = plt.subplots()
```

```
In [3]: ln, = ax.plot([0, 1, 4, 9, 16])
```

```
In [4]: plt.show()
```

```
In [5]: ln.set_color('g')
```

will automatically update the plot to be green. Any subsequent modifications to the Artist objects will do likewise.

This is the first step of a larger consolidation and simplification of the pyplot internals.

Working with labeled data like pandas DataFrames

Plot methods which take arrays as inputs can now also work with labeled data and unpack such data.

This means that the following two examples produce the same plot:

Example

```
df = pandas.DataFrame({"var1": [1, 2, 3, 4, 5, 6], "var2": [1, 2, 3, 4, 5, 6]})
plt.plot(df["var1"], df["var2"])
```

Example

```
plt.plot("var1", "var2", data=df)
```

This works for most plotting methods, which expect arrays/sequences as inputs. `data` can be anything which supports `__getitem__` (dict, `pandas.DataFrame`, `h5py`, ...) to access array like values with string keys.

In addition to this, some other changes were made, which makes working with labeled data (ex `pandas.Series`) easier:

- For plotting methods with label keyword argument, one of the data inputs is designated as the label source. If the user does not supply a label that value object will be introspected for a label, currently by looking for a `name` attribute. If the value object does not have a `name` attribute but was specified by as a key into the data kwarg, then the key is used. In the above examples, this results in an implicit `label="var2"` for both cases.
- `plot()` now uses the index of a `Series` instead of `np.arange(len(y))`, if no `x` argument is supplied.

Added `axes.prop_cycle` key to `rcParams`

This is a more generic form of the now-deprecated `axes.color_cycle` param. Now, we can cycle more than just colors, but also `linestyles`, `hatches`, and just about any other artist property. `Cycler` notation is used for defining property cycles. Adding cyclers together will be like you are `zip()`-ing together two or more property cycles together:

```
axes.prop_cycle: cycler('color', 'rgb') + cycler('lw', [1, 2, 3])
```

You can even multiply cyclers, which is like using `itertools.product()` on two or more property cycles. Remember to use parentheses if writing a multi-line `prop_cycle` parameter.

New Colormaps

All four of the colormaps proposed as the new default are available as `'viridis'` (the new default in 2.0), `'magma'`, `'plasma'`, and `'inferno'`

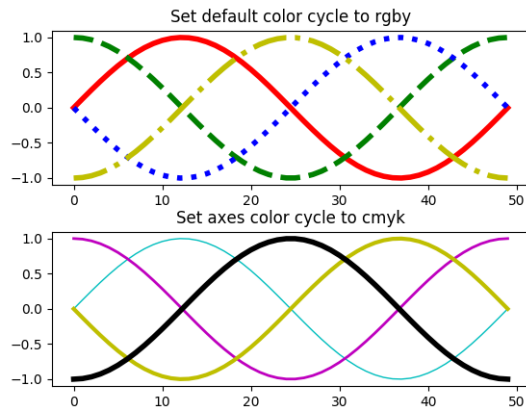
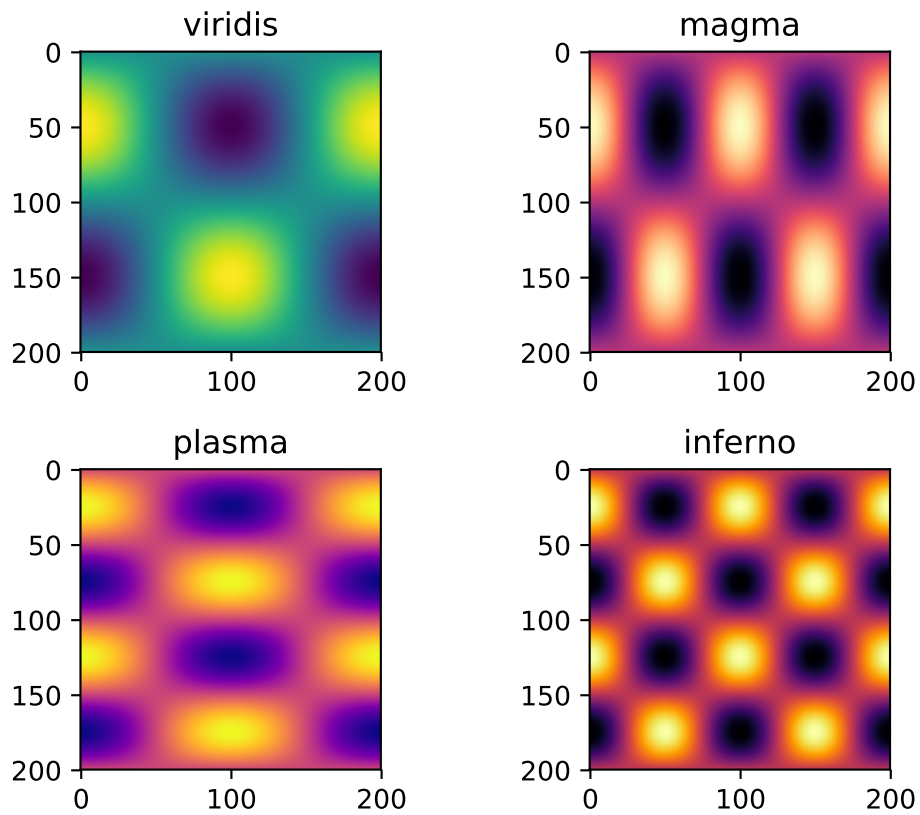


Fig. 27: Color Cycle



Styles

Several new styles have been added, including many styles from the Seaborn project. Additionally, in order to prep for the upcoming 2.0 style-change release, a 'classic' and 'default' style has been added. For this release, the 'default' and 'classic' styles are identical. By using them now in your scripts, you can help ensure a smooth transition during future upgrades of matplotlib, so that you can upgrade to the snazzy new defaults when you are ready!

```
import matplotlib.style
matplotlib.style.use('classic')
```

The 'default' style will give you matplotlib's latest plotting styles:

```
matplotlib.style.use('default')
```

Backends

New backend selection

The environment variable MPLBACKEND can now be used to set the matplotlib backend.

wx backend has been updated

The wx backend can now be used with both wxPython classic and [Phoenix](#).

wxPython classic has to be at least version 2.8.12 and works on Python 2.x. As of May 2015 no official release of wxPython Phoenix is available but a current snapshot will work on Python 2.7+ and 3.4+.

If you have multiple versions of wxPython installed, then the user code is responsible setting the wxPython version. How to do this is explained in the comment at the beginning of the example `examples/user_interfaces/embedding_in_wx2.py`.

Configuration (rcParams)

Some parameters have been added, others have been improved.

Parameter	Description
{x,y}axis. labelpad	mplot3d now respects these parameters
axes. labelpad	Default space between the axis and the label
errorbar. capsize	Default length of end caps on error bars
{x,y}tick. minor. visible	Default visibility of minor x/y ticks
legend. framealpha	Default transparency of the legend frame box
legend. facecolor	Default facecolor of legend frame box (or 'inherit' from axes.facecolor)
legend. edgecolor	Default edgecolor of legend frame box (or 'inherit' from axes.edgecolor)
figure. titlesize	Default font size for figure subtitles
figure. titleweight	Default font weight for figure subtitles
image. composite_image	Whether a vector graphics backend should composite several images into a single image or not when saving. Useful when needing to edit the files further in Inkscape or other programs.
markers. fillstyle	Default fillstyle of markers. Possible values are 'full' (the default), 'left', 'right', 'bottom', 'top' and 'none'
toolbar	Added 'toolmanager' as a valid value, enabling the experimental ToolManager feature.

Widgets

Active state of Selectors

All selectors now implement `set_active` and `get_active` methods (also called when accessing the active property) to properly update and query whether they are active.

Moved `ignore`, `set_active`, and `get_active` methods to base class `Widget`

Pushes up duplicate methods in child class to parent class to avoid duplication of code.

Adds enable/disable feature to MultiCursor

A `MultiCursor` object can be disabled (and enabled) after it has been created without destroying the object. Example:

`multi_cursor.active = False`

Improved RectangleSelector and new EllipseSelector Widget

Adds an interactive keyword which enables visible handles for manipulating the shape after it has been drawn.

Adds keyboard modifiers for:

- Moving the existing shape (default key = 'space')
- Making the shape square (default 'shift')
- Make the initial point the center of the shape (default 'control')
- Square and center can be combined

Allow Artists to Display Pixel Data in Cursor

Adds `get_pixel_data` and `format_pixel_data` methods to artists which can be used to add zdata to the cursor display in the status bar. Also adds an implementation for Images.

New plotting features

Auto-wrapping Text

Added the keyword argument "wrap" to Text, which automatically breaks long lines of text when being drawn. Works for any rotated text, different modes of alignment, and for text that are either labels or titles. This breaks at the Figure, not Axes edge.

Contour plot corner masking

Ian Thomas rewrote the C++ code that calculates contours to add support for corner masking. This is controlled by a new keyword argument `corner_mask` in the functions `contour()` and `contourf()`. The previous behaviour, which is now obtained using `corner_mask=False`, was for a single masked point to completely mask out all four quads touching that point. The new behaviour, obtained using `corner_mask=True`, only masks the corners of those quads touching the point; any triangular corners comprising three unmasked points are contoured as usual. If the `corner_mask` keyword argument is not specified, the default value is taken from `rcParams`.

Mostly unified linestyles for Line2D, Patch and Collection

The handling of linestyles for Lines, Patches and Collections has been unified. Now they all support defining linestyles with short symbols, like "--", as well as with full names, like "dashed". Also the

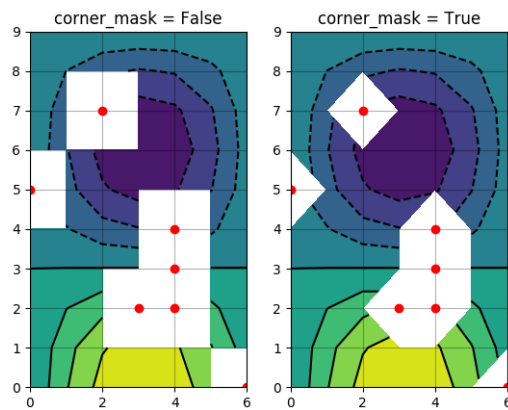
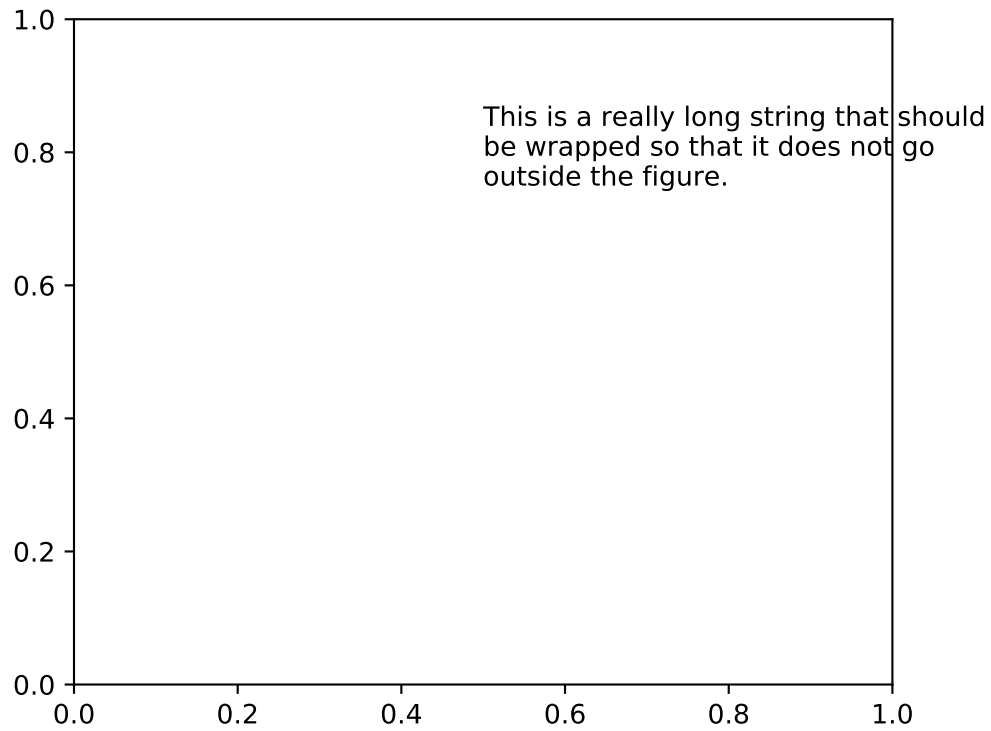


Fig. 28: Contour Corner Mask

definition using a dash pattern `((0., [3., 3.]`) is supported for all methods using `Line2D`, `Patch` or `Collection`.

Legend marker order

Added ability to place the label before the marker in a legend box with `markerfirst` keyword

Support for legend for `PolyCollection` and `stackplot`

Added a `legend_handler` for `PolyCollection` as well as a `labels` argument to `stackplot()`.

Support for alternate pivots in `mplot3d` quiver plot

Added a pivot kwarg to `quiver()` that controls the pivot point around which the quiver line rotates. This also determines the placement of the arrow head along the quiver line.

Logit Scale

Added support for the 'logit' axis scale, a nonlinear transformation

$$x \rightarrow \log 10(x/(1 - x))$$

for data between 0 and 1 excluded.

Add step kwargs to `fill_between`

Added step kwarg to `Axes.fill_between` to allow to fill between lines drawn using the 'step' draw style. The values of step match those of the where kwarg of `Axes.step`. The asymmetry of the kwargs names is not ideal, but `Axes.fill_between` already has a where kwarg.

This is particularly useful for plotting pre-binned histograms.

Square Plot

Implemented square plots feature as a new parameter in the axis function. When argument 'square' is specified, equal scaling is set, and the limits are set such that `xmax-xmin == ymax-ymin`.

Updated `figimage` to take optional `resize` parameter

Added the ability to plot simple 2D-Array using `plt.figimage(X, resize=True)`. This is useful for plotting simple 2D-Array without the Axes or whitespacing around the image.

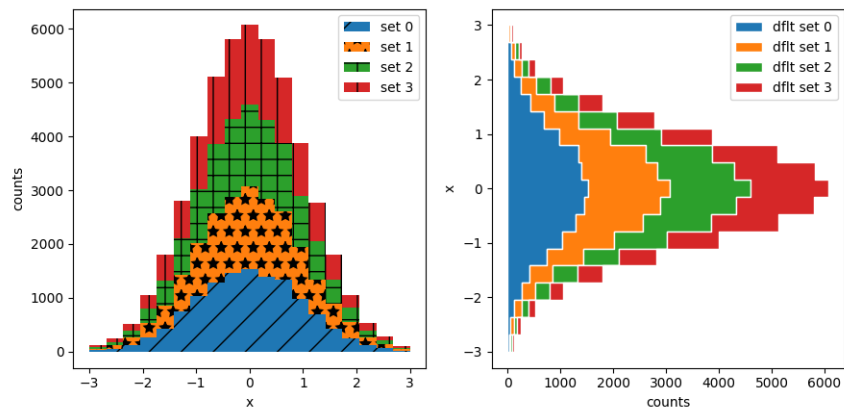
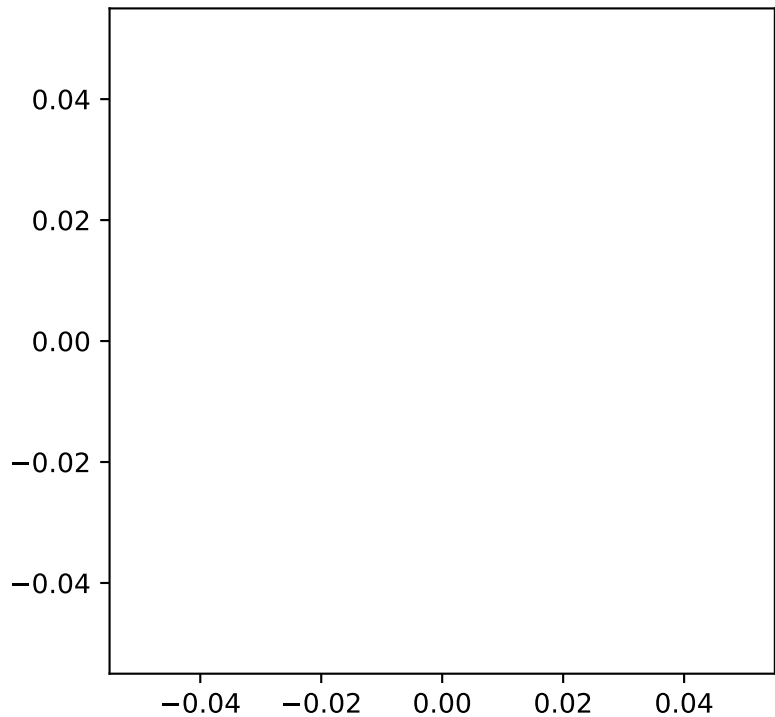
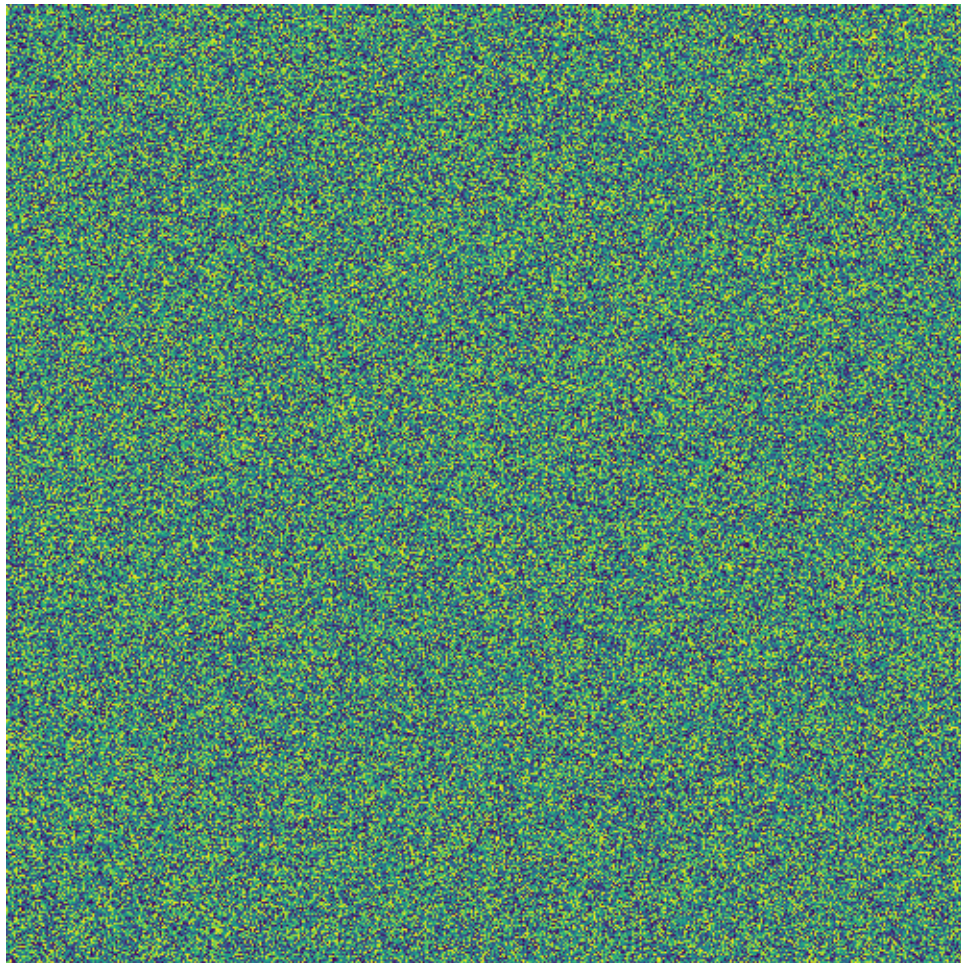


Fig. 29: Filled Step





Updated `Figure.savefig()` can now use figure's dpi

Added support to save the figure with the same dpi as the figure on the screen using `dpi='figure'`.

Example:

```
f = plt.figure(dpi=25)           # dpi set to 25
S = plt.scatter([1,2,3],[4,5,6])
f.savefig('output.png', dpi='figure') # output savefig dpi set to 25 (same as figure)
```

Updated `Table` to control edge visibility

Added the ability to toggle the visibility of lines in `Tables`. Functionality added to the `pyplot.table()` factory function under the keyword argument `"edges"`. Values can be the strings `"open"`, `"closed"`, `"horizontal"`, `"vertical"` or combinations of the letters `"L"`, `"R"`, `"T"`, `"B"` which represent left, right, top, and bottom respectively.

Example:

```
table(..., edges="open") # No line visible
table(..., edges="closed") # All lines visible
table(..., edges="horizontal") # Only top and bottom lines visible
table(..., edges="LT") # Only left and top lines visible.
```

Zero `r/cstride` support in `plot_wireframe`

Adam Hughes added support to `mplot3d`'s `plot_wireframe` to draw only row or column line plots.

Plot `bar` and `barh` with labels

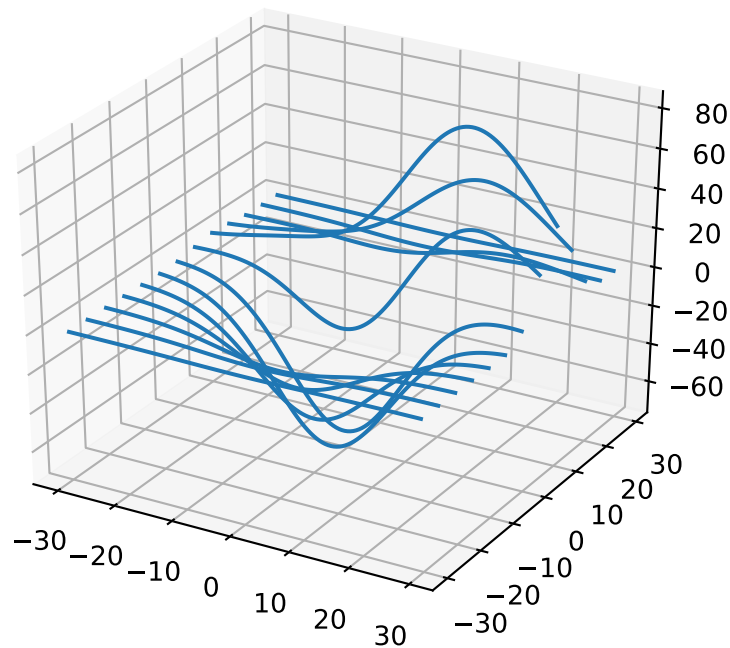
Added kwarg `"tick_label"` to `bar` and `barh` to support plotting bar graphs with a text label for each bar.

Added `center` and `frame` kwargs to `pie`

These control where the center of the pie graph are and if the Axes frame is shown.

Fixed 3D filled contour plot polygon rendering

Certain cases of 3D filled contour plots that produce polygons with multiple holes produced improper rendering due to a loss of path information between `PolyCollection` and `Poly3DCollection`. A function `set_verts_and_codes()` was added to allow path information to be retained for proper rendering.



Dense colorbars are rasterized

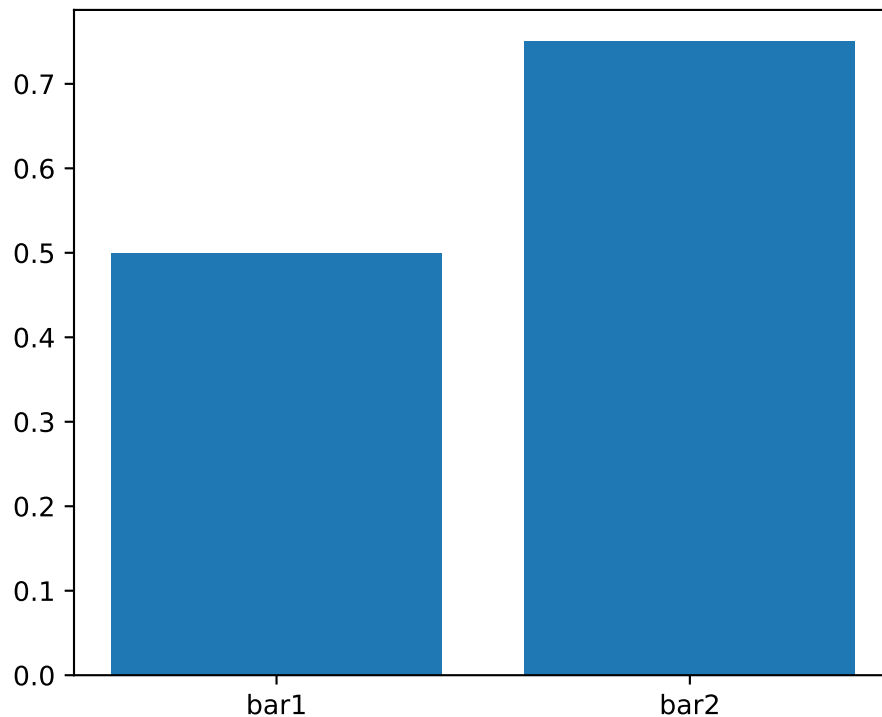
Vector file formats (pdf, ps, svg) are efficient for many types of plot element, but for some they can yield excessive file size and even rendering artifacts, depending on the renderer used for screen display. This is a problem for colorbars that show a large number of shades, as is most commonly the case. Now, if a colorbar is showing 50 or more colors, it will be rasterized in vector backends.

DateFormatter strftime

strftime method will format a `datetime.datetime` object with the format string passed to the formatter's constructor. This method accepts datetimes with years before 1900, unlike `datetime.datetime.strftime()`.

Artist-level `{get,set}_usetex` for text

Add `{get,set}_usetex` methods to Text objects which allow artist-level control of LaTeX rendering vs the internal mathtex rendering.



`ax.remove()` works as expected

As with artists added to an Axes, Axes objects can be removed from their figure via `remove()`.

API Consistency fix within Locators `set_params()` function

`set_params()` function, which sets parameters within a Locator type instance, is now available to all Locator types. The implementation also prevents unsafe usage by strictly defining the parameters that a user can set.

To use, call `set_params()` on a Locator instance with desired arguments:

```
loc = matplotlib.ticker.LogLocator()
# Set given attributes for loc.
loc.set_params(numticks=8, numdecs=8, subs=[2.0], base=8)
# The below will error, as there is no such parameter for LogLocator
# named foo
# loc.set_params(foo='bar')
```

Date Locators

Date Locators (derived from `DateLocator`) now implement the `tick_values()` method. This is expected of all Locators derived from `Locator`.

The Date Locators can now be used easily without creating axes

```
from datetime import datetime
from matplotlib.dates import YearLocator
t0 = datetime(2002, 10, 9, 12, 10)
tf = datetime(2005, 10, 9, 12, 15)
loc = YearLocator()
values = loc.tick_values(t0, tf)
```

OffsetBoxes now support clipping

Artists draw onto objects of type `OffsetBox` through `DrawingArea` and `TextArea`. The `TextArea` calculates the required space for the text and so the text is always within the bounds, for this nothing has changed.

However, `DrawingArea` acts as a parent for zero or more Artists that draw on it and may do so beyond the bounds. Now child Artists can be clipped to the bounds of the `DrawingArea`.

OffsetBoxes now considered by `tight_layout`

When `tight_layout()` or `Figure.tight_layout()` or `GridSpec.tight_layout()` is called, `OffsetBoxes` that are anchored outside the axes will not get chopped out. The `OffsetBoxes` will also not get overlapped by other axes in case of multiple subplots.

Per-page pdf notes in multi-page pdfs (`PdfPages`)

Add a new method `attach_note()` to the `PdfPages` class, allowing the attachment of simple text notes to pages in a multi-page pdf of figures. The new note is visible in the list of pdf annotations in a viewer that has this facility (Adobe Reader, OSX Preview, Skim, etc.). Per default the note itself is kept off-page to prevent it to appear in print-outs.

`PdfPages.attach_note` needs to be called before `savefig()` in order to be added to the correct figure.

Updated `fignum_exists` to take figure name

Added the ability to check the existence of a figure using its name instead of just the figure number. Example:

```
figure('figure')
fignum_exists('figure') #true
```

ToolManager

Federico Ariza wrote the new ToolManager that comes as replacement for NavigationToolbar2

ToolManager offers a new way of looking at the user interactions with the figures. Before we had the NavigationToolbar2 with its own tools like zoom/pan/home/save/... and also we had the shortcuts like yscale/grid/quit/.... Toolmanager relocate all those actions as Tools (located in backend_tools), and defines a way to access/trigger/reconfigure them.

The Toolbars are replaced for ToolContainers that are just GUI interfaces to trigger the tools. But don't worry the default backends include a ToolContainer called toolbar

Note: At the moment, we release this primarily for feedback purposes and should be treated as experimental until further notice as API changes will occur. For the moment the ToolManager works only with the GTK3 and Tk backends. Make sure you use one of those. Port for the rest of the backends is coming soon.

To activate the ToolManager include the following at the top of your file

```
>>> matplotlib.rcParams['toolbar'] = 'toolmanager'
```

Interact with the ToolContainer

The most important feature is the ability to easily reconfigure the ToolContainer (aka toolbar). For example, if we want to remove the "forward" button we would just do.

```
>>> fig.canvas.manager.toolmanager.remove_tool('forward')
```

Now if you want to programmatically trigger the "home" button

```
>>> fig.canvas.manager.toolmanager.trigger_tool('home')
```

New Tools for ToolManager

It is possible to add new tools to the ToolManager

A very simple tool that prints "You're awesome" would be:

```
from matplotlib.backend_tools import ToolBase
class AwesomeTool(ToolBase):
    def trigger(self, *args, **kwargs):
        print("You're awesome")
```

To add this tool to ToolManager

```
>>> fig.canvas.manager.toolmanager.add_tool('Awesome', AwesomeTool)
```

If we want to add a shortcut ("d") for the tool

```
>>> fig.canvas.manager.toolmanager.update_keymap('Awesome', 'd')
```

To add it to the toolbar inside the group 'foo'

```
>>> fig.canvas.manager.toolbar.add_tool('Awesome', 'foo')
```

There is a second class of tools, "Toggleable Tools", this are almost the same as our basic tools, just that belong to a group, and are mutually exclusive inside that group. For tools derived from `ToolToggleBase` there are two basic methods `enable` and `disable` that are called automatically whenever it is toggled.

A full example is located in `/gallery/user_interfaces/toolmanager_sgskip`

`cbook.is_sequence_of_strings` recognizes string objects

This is primarily how pandas stores a sequence of strings

```
import pandas as pd
import matplotlib.cbook as cbook

a = np.array(['a', 'b', 'c'])
print(cbook.is_sequence_of_strings(a)) # True

a = np.array(['a', 'b', 'c'], dtype=object)
print(cbook.is_sequence_of_strings(a)) # True

s = pd.Series(['a', 'b', 'c'])
print(cbook.is_sequence_of_strings(s)) # True
```

Previously, the last two prints returned false.

New `close-figs` argument for `plot` directive

Matplotlib has a sphinx extension `plot_directive` that creates plots for inclusion in sphinx documents. Matplotlib 1.5 adds a new option to the `plot` directive - `close-figs` - that closes any previous figure windows before creating the plots. This can help avoid some surprising duplicates of plots when using `plot_directive`.

Support for URL string arguments to `imread`

The `imread()` function now accepts URL strings that point to remote PNG files. This circumvents the generation of a `HTTPResponse` object directly.

Display hook for animations in the IPython notebook

Animation instances gained a `__repr_html__` method to support inline display of animations in the notebook. The method used to display is controlled by the `animation.html` rc parameter, which currently supports values of `none` and `html5`. `none` is the default, performing no display. `html5` converts the animation to an h264 encoded video, which is embedded directly in the notebook.

Users not wishing to use the `__repr_html__` display hook can also manually call the `to_html5_video` method to get the HTML and display using IPython's HTML display class:

```
from IPython.display import HTML
HTML(anim.to_html5_video())
```

Prefixed pkg-config for building

Handling of pkg-config has been fixed in so far as it is now possible to set it using the environment variable `PKG_CONFIG`. This is important if your toolchain is prefixed. This is done in a similar way as setting `CC` or `CXX` before building. An example follows.

```
export PKG_CONFIG=x86_64-pc-linux-gnu-pkg-config
```

5.3.11 New in matplotlib 2.0

Note: matplotlib 2.0 supports Python 2.7, and 3.4+

Default style changes

The major changes in v2.0 are related to overhauling the default styles.

Changes to the default style

The most important changes in matplotlib 2.0 are the changes to the default style.

While it is impossible to select the best default for all cases, these are designed to work well in the most common cases.

A 'classic' style sheet is provided so reverting to the 1.x default values is a single line of python

```
import matplotlib.style
import matplotlib as mpl
mpl.style.use('classic')
```

See [The matplotlibrc file](#) for details about how to persistently and selectively revert many of these changes.

Table of Contents

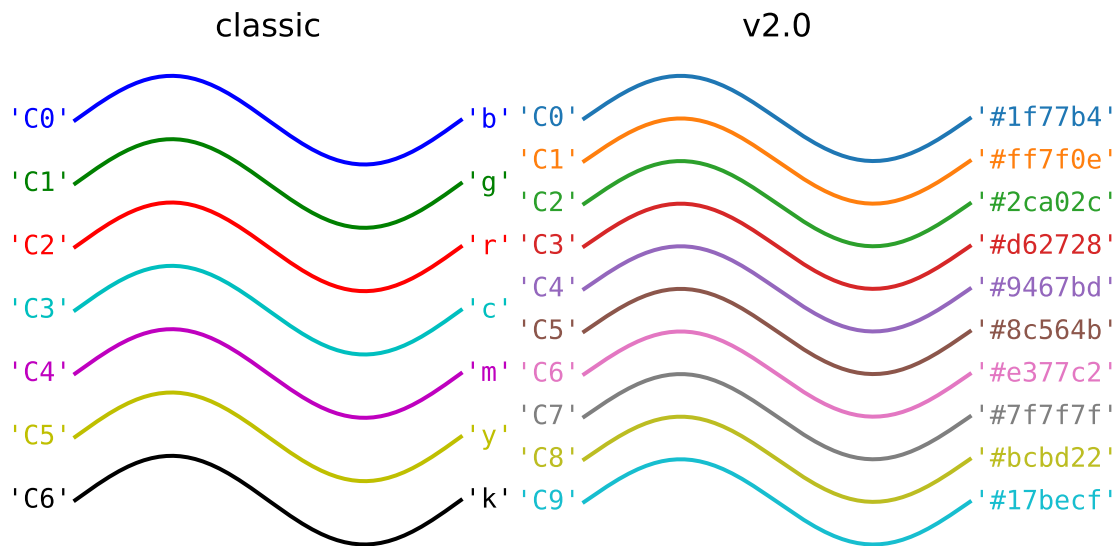
- Colors, color cycles, and color maps
 - Colors in default property cycle
 - Colormap
 - Interactive figures
 - Grid lines
- Figure size, font size, and screen dpi
- Plotting functions
 - scatter
 - plot
 - errorbar
 - boxplot
 - fill_between and fill_betweenx
 - Patch edges and color
 - hexbin
 - bar and barh
- Hatching
- Fonts
 - Normal text
 - Math text
- Legends
- Image
 - Interpolation
 - Colormapping pipeline
 - Shading
- Plot layout
 - Auto limits
 - Z-order
 - Ticks
 - Tick label formatting

- mplot3d

Colors, color cycles, and color maps

Colors in default property cycle

The colors in the default property cycle have been changed from ['b', 'g', 'r', 'c', 'm', 'y', 'k'] to the category10 color palette used by Vega and d3 originally developed at Tableau.



In addition to changing the colors, an additional method to specify colors was added. Previously, the default colors were the single character short-hand notations for red, green, blue, cyan, magenta, yellow, and black. This made them easy to type and usable in the abbreviated style string in plot, however the new default colors are only specified via hex values. To access these colors outside of the property cycling the notation for colors 'CN', where N takes values 0-9, was added to denote the first 10 colors in `mpl.rcParams['axes.prop_cycle']`. See [Specifying Colors](#) for more details.

To restore the old color cycle use

```
fromycler import cycler
mpl.rcParams['axes.prop_cycle'] = cycler(color='bgrcmyk')
```

or set

```
axes.prop_cycle : cycler('color', 'bgrcmyk')
```

in your matplotlibrc file.

Colormap

The new default color map used by `matplotlib.cm.ScalarMappable` instances is `'viridis'` (aka `option D`).

For an introduction to color theory and how `'viridis'` was generated watch Nathaniel Smith and Stéfan van der Walt's talk from SciPy2015. See [here for many more details](#) about the other alternatives and the tools used to create the color map. For details on all of the color maps available in matplotlib see [Colormaps in Matplotlib](#).

The previous default can be restored using

```
mpl.rcParams['image.cmap'] = 'jet'
```

or setting

```
image.cmap : 'jet'
```

in your `matplotlibrc` file; however this is strongly discouraged.

Interactive figures

The default interactive figure background color has changed from grey to white, which matches the default background color used when saving.

The previous defaults can be restored by

```
mpl.rcParams['figure.facecolor'] = '0.75'
```

or by setting

```
figure.facecolor : '0.75'
```

in your `matplotlibrc` file.

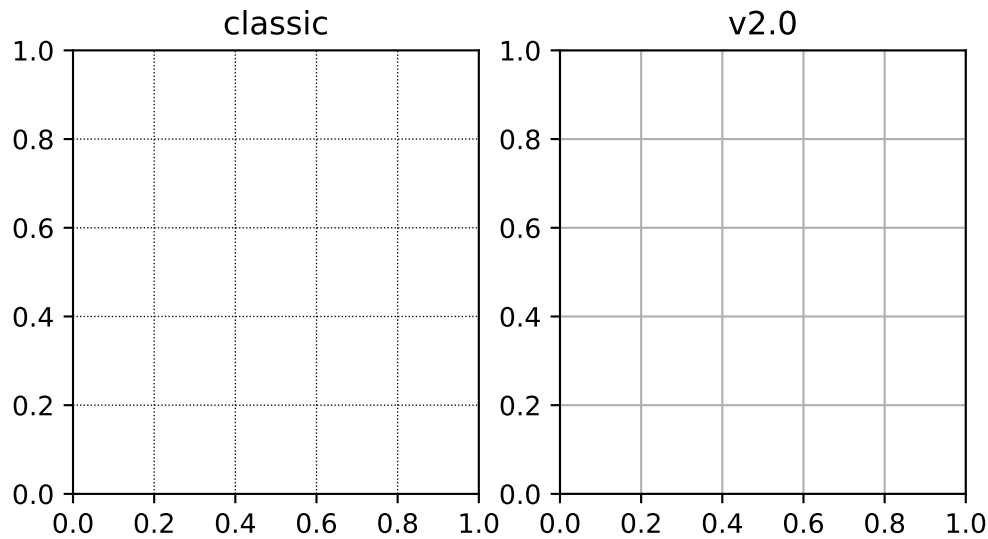
Grid lines

The default style of grid lines was changed from black dashed lines to thicker solid light grey lines.

The previous default can be restored by using:

```
mpl.rcParams['grid.color'] = 'k'  
mpl.rcParams['grid.linestyle'] = ':'  
mpl.rcParams['grid.linewidth'] = 0.5
```

or by setting:



```

grid.color      : k      # grid color
grid.linestyle  : :      # dotted
grid.linewidth  : 0.5    # in points

```

in your matplotlibrc file.

Figure size, font size, and screen dpi

The default dpi used for on-screen display was changed from 80 dpi to 100 dpi, the same as the default dpi for saving files. Due to this change, the on-screen display is now more what-you-see-is-what-you-get for saved files. To keep the figure the same size in terms of pixels, in order to maintain approximately the same size on the screen, the default figure size was reduced from 8x6 inches to 6.4x4.8 inches. As a consequence of this the default font sizes used for the title, tick labels, and axes labels were reduced to maintain their size relative to the overall size of the figure. By default the dpi of the saved image is now the dpi of the Figure instance being saved.

This will have consequences if you are trying to match text in a figure directly with external text.

The previous defaults can be restored by

```

mpl.rcParams['figure.figsize'] = [8.0, 6.0]
mpl.rcParams['figure.dpi'] = 80
mpl.rcParams['savefig.dpi'] = 100

mpl.rcParams['font.size'] = 12
mpl.rcParams['legend.fontsize'] = 'large'
mpl.rcParams['figure.titlesize'] = 'medium'

```

or by setting:

```
figure.figsize : [8.0, 6.0]
figure.dpi      : 80
savefig.dpi     : 100

font.size       : 12.0
legend.fontsize : 'large'
figure.titlesize : 'medium'
```

In your `matplotlibrc` file.

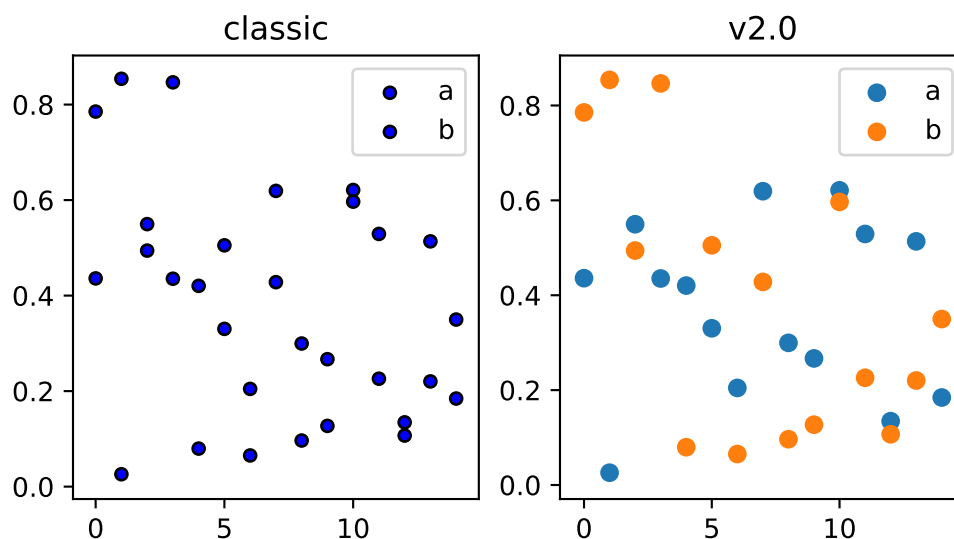
In addition, the forward kwarg to `set_size_inches` now defaults to `True` to improve the interactive experience. Backend canvases that adjust the size of their bound `matplotlib.figure.Figure` must pass `forward=False` to avoid circular behavior. This default is not configurable.

Plotting functions

scatter

The following changes were made to the default behavior of `scatter`

- The default size of the elements in a scatter plot is now based on the rcParam `lines.markersize` so it is consistent with `plot(X, Y, 'o')`. The old value was 20, and the new value is 36 (6^2).
- `scatter` markers no longer have a black edge.
- if the color of the markers is not specified it will follow the property cycle, pulling from the 'patches' cycle on the Axes.



The classic default behavior of `scatter` can only be recovered through `mpl.style.use('classic')`. The marker size can be recovered via

```
mpl.rcParams['lines.markersize'] = np.sqrt(20)
```

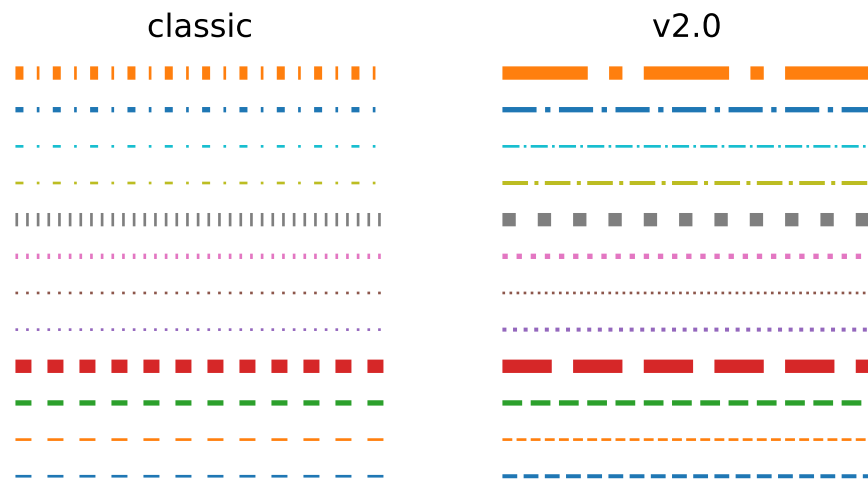
however, this will also affect the default marker size of plot. To recover the classic behavior on a per-call basis pass the following kwargs:

```
classic_kwargs = {'s': 20, 'edgecolors': 'k', 'c': 'b'}
```

plot

The following changes were made to the default behavior of plot

- the default linewidth increased from 1 to 1.5
- the dash patterns associated with '--', ':', and '-.' have changed
- the dash patterns now scale with line width



The previous defaults can be restored by setting:

```
mpl.rcParams['lines.linewidth'] = 1.0
mpl.rcParams['lines.dashed_pattern'] = [6, 6]
mpl.rcParams['lines.dashdot_pattern'] = [3, 5, 1, 5]
mpl.rcParams['lines.dotted_pattern'] = [1, 3]
mpl.rcParams['lines.scale_dashes'] = False
```

or by setting:

```
lines.linewidth : 1.0
lines.dashed_pattern : 6, 6
lines.dashdot_pattern : 3, 5, 1, 5
```

(continues on next page)

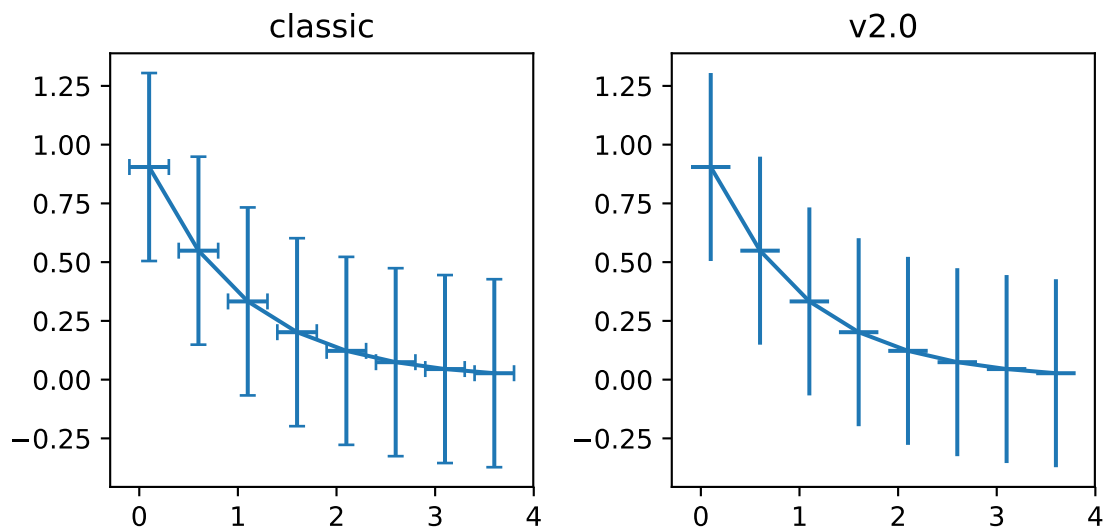
(continued from previous page)

```
lines.dotted_pattern : 1, 3
lines.scale_dashes: False
```

in your matplotlibrc file.

errorbar

By default, caps on the ends of errorbars are not present.



This also changes the return value of `errorbar()` as the list of 'caplines' will be empty by default.

The previous defaults can be restored by setting:

```
mpl.rcParams['errorbar.capsize'] = 3
```

or by setting

```
errorbar.capsize : 3
```

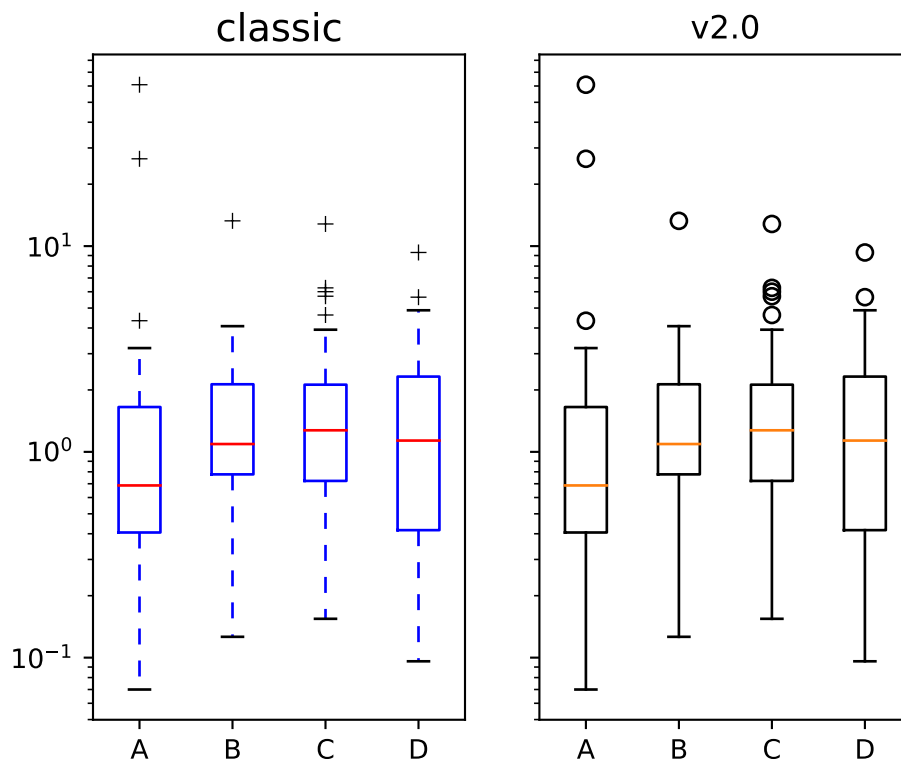
in your matplotlibrc file.

boxplot

Previously, boxplots were composed of a mish-mash of styles that were, for better for worse, inherited from Matlab. Most of the elements were blue, but the medians were red. The fliers (outliers) were black plus-symbols (+) and the whiskers were dashed lines, which created ambiguity if the (solid and black) caps were not drawn.

For the new defaults, everything is black except for the median and mean lines (if drawn), which are set to the first two elements of the current color cycle. Also, the default flier markers are now

hollow circles, which maintain the ability of the plus-symbols to overlap without obscuring data too much.



The previous defaults can be restored by setting:

```
mpl.rcParams['boxplot.flierprops.color'] = 'k'
mpl.rcParams['boxplot.flierprops.marker'] = '+'
mpl.rcParams['boxplot.flierprops.markerfacecolor'] = 'none'
mpl.rcParams['boxplot.flierprops.markeredgecolor'] = 'k'
mpl.rcParams['boxplot.boxprops.color'] = 'b'
mpl.rcParams['boxplot.whiskerprops.color'] = 'b'
mpl.rcParams['boxplot.whiskerprops.linestyle'] = '--'
mpl.rcParams['boxplot.medianprops.color'] = 'r'
mpl.rcParams['boxplot.meanprops.color'] = 'r'
mpl.rcParams['boxplot.meanprops.marker'] = '^'
mpl.rcParams['boxplot.meanprops.markerfacecolor'] = 'r'
mpl.rcParams['boxplot.meanprops.markeredgecolor'] = 'k'
mpl.rcParams['boxplot.meanprops.markersize'] = 6
mpl.rcParams['boxplot.meanprops.linestyle'] = '--'
mpl.rcParams['boxplot.meanprops.linewidth'] = 1.0
```

or by setting:

```

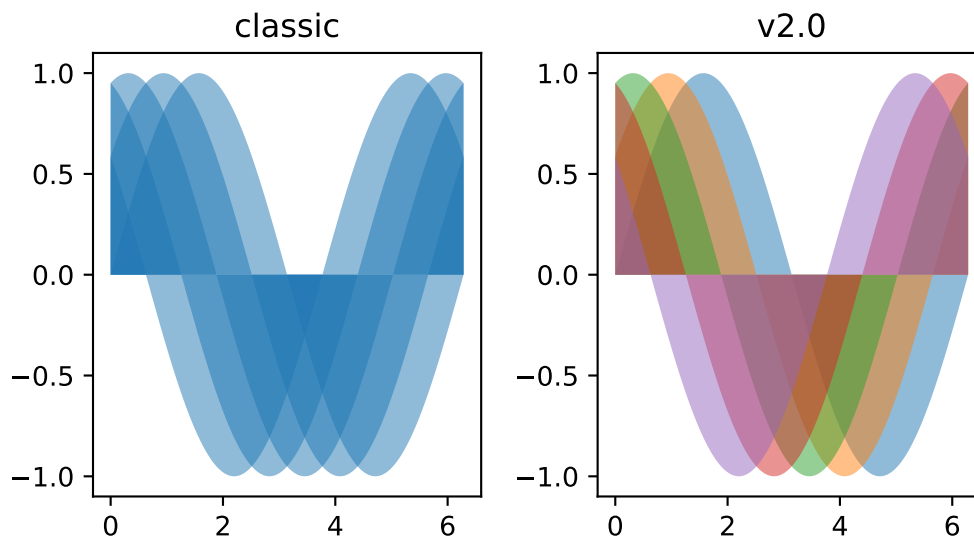
boxplot.flierprops.color:      'k'
boxplot.flierprops.marker:     '+'
boxplot.flierprops.markerfacecolor: 'none'
boxplot.flierprops.markeredgcolor: 'k'
boxplot.boxprops.color:       'b'
boxplot.whiskerprops.color:    'b'
boxplot.whiskerprops.linestyle: '--'
boxplot.medianprops.color:     'r'
boxplot.meanprops.color:       'r'
boxplot.meanprops.marker:      '^'
boxplot.meanprops.markerfacecolor: 'r'
boxplot.meanprops.markeredgcolor: 'k'
boxplot.meanprops.markersize:   6
boxplot.meanprops.linestyle:    '--'
boxplot.meanprops.linewidth:    1.0

```

in your matplotlibrc file.

`fill_between` and `fill_betweenx`

`fill_between` and `fill_betweenx` both follow the patch color cycle.

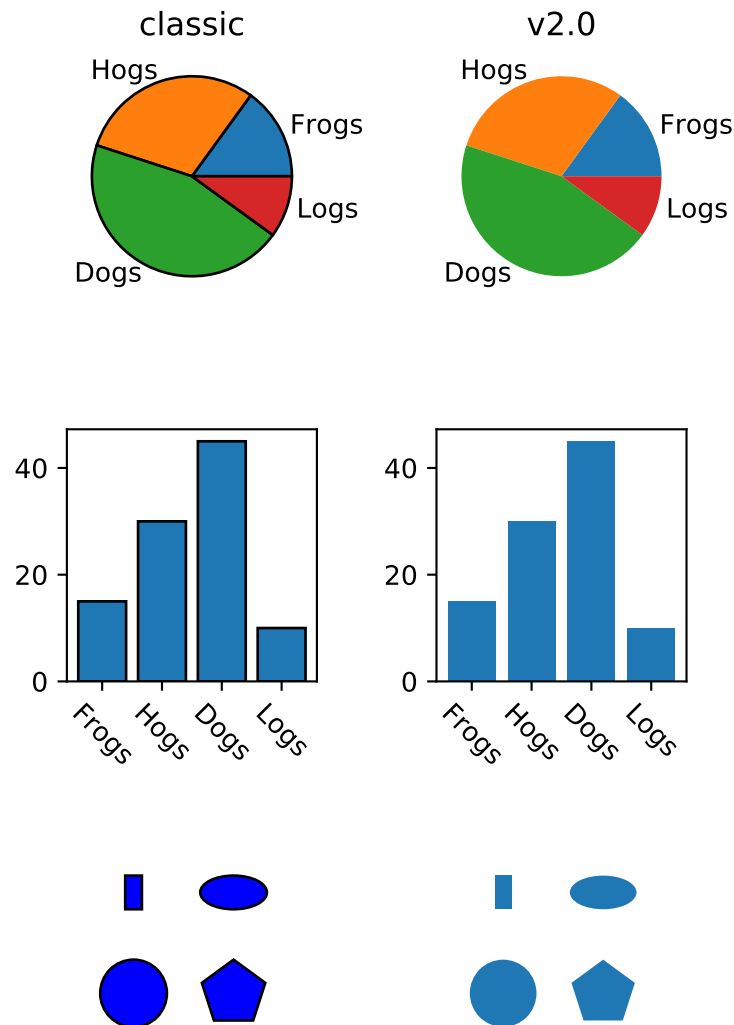


If the facecolor is set via the facecolors or color keyword argument, then the color is not cycled.

To restore the previous behavior, explicitly pass the keyword argument `facecolors='C0'` to the method call.

Patch edges and color

Most artists drawn with a patch (`~matplotlib.axes.Axes.bar`, `~matplotlib.axes.Axes.pie`, etc) no longer have a black edge by default. The default face color is now 'C0' instead of 'b'.



The previous defaults can be restored by setting:

```
mpl.rcParams['patch.force_edgecolor'] = True
mpl.rcParams['patch.facecolor'] = 'b'
```

or by setting:

```
patch.facecolor      : b
patch.force_edgecolor : True
```

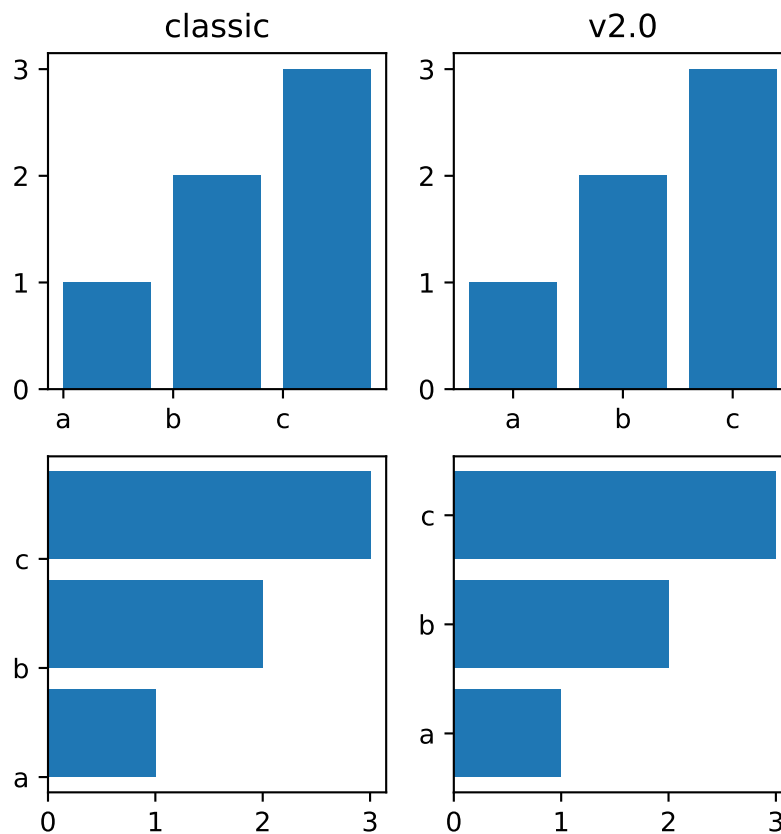

in your matplotlibrc file.

hexbin

The default value of the `linecolor` kwarg for `hexbin` has changed from `'none'` to `'face'`. If `'none'` is now supplied, no line edges are drawn around the hexagons.

bar and barh

The default value of the `align` kwarg for both `bar` and `barh` is changed from `'edge'` to `'center'`.



To restore the previous behavior explicitly pass the keyword argument `align='edge'` to the method call.

Hatching

The color of the lines in the hatch is now determined by

- If an edge color is explicitly set, use that for the hatch color
- If the edge color is not explicitly set, use `rcParam['hatch.color']` which is looked up at artist creation time.

The width of the lines in a hatch pattern is now configurable by the rcParams `hatch.linewidth`, which defaults to 1 point. The old behavior for the line width was different depending on backend:

- PDF: 0.1 pt
- SVG: 1.0 pt
- PS: 1 px
- Agg: 1 px

The old line width behavior can not be restored across all backends simultaneously, but can be restored for a single backend by setting:

```
mpl.rcParams['hatch.linewidth'] = 0.1 # previous pdf hatch linewidth
mpl.rcParams['hatch.linewidth'] = 1.0 # previous svg hatch linewidth
```

The behavior of the PS and Agg backends was DPI dependent, thus:

```
mpl.rcParams['figure.dpi'] = dpi
mpl.rcParams['savefig.dpi'] = dpi # or leave as default 'figure'
mpl.rcParams['hatch.linewidth'] = 1.0 / dpi # previous ps and Agg hatch linewidth
```

There is no direct API level control of the hatch color or linewidth.

Hatching patterns are now rendered at a consistent density, regardless of DPI. Formerly, high DPI figures would be more dense than the default, and low DPI figures would be less dense. This old behavior cannot be directly restored, but the density may be increased by repeating the hatch specifier.

Fonts

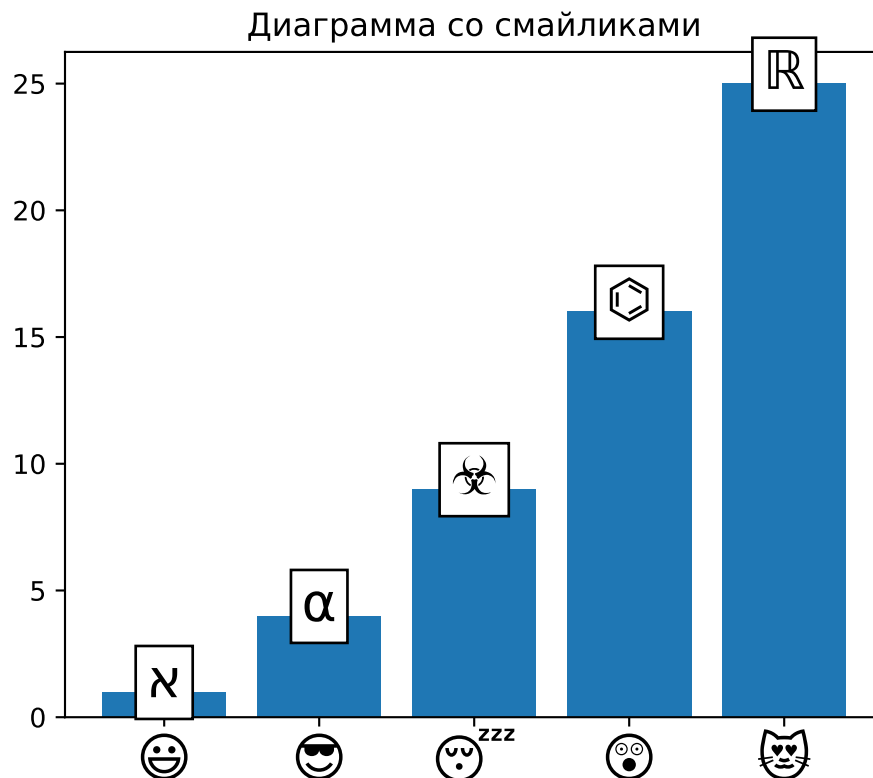
Normal text

The default font has changed from "Bitstream Vera Sans" to "DejaVu Sans". DejaVu Sans has additional international and math characters, but otherwise has the same appearance as Bitstream Vera Sans. Latin, Greek, Cyrillic, Armenian, Georgian, Hebrew, and Arabic are [all supported](#) (but right-to-left rendering is still not handled by matplotlib). In addition, DejaVu contains a sub-set of emoji symbols.

See the [DejaVu Sans PDF sample for full coverage](#).

Math text

The default math font when using the built-in math rendering engine (`mathtext`) has changed from "Computer Modern" (i.e. LaTeX-like) to "DejaVu Sans". This change has no effect if the TeX



backend is used (i.e. `text.usetex` is `True`).

To revert to the old behavior set the:

```
mpl.rcParams['mathtext.fontset'] = 'cm'
mpl.rcParams['mathtext.rm'] = 'serif'
```

or set:

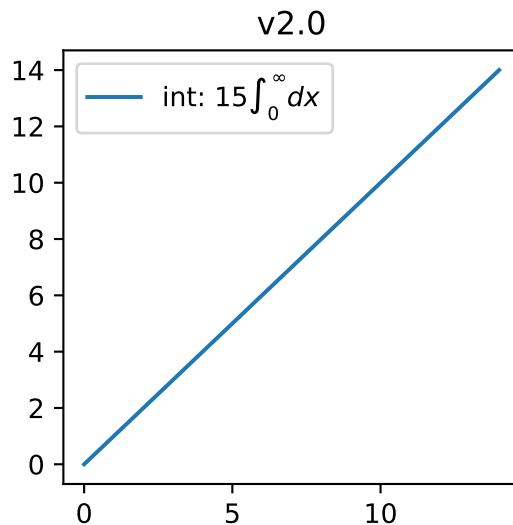
```
mathtext.fontset: cm
mathtext.rm : serif
```

in your `matplotlibrc` file.

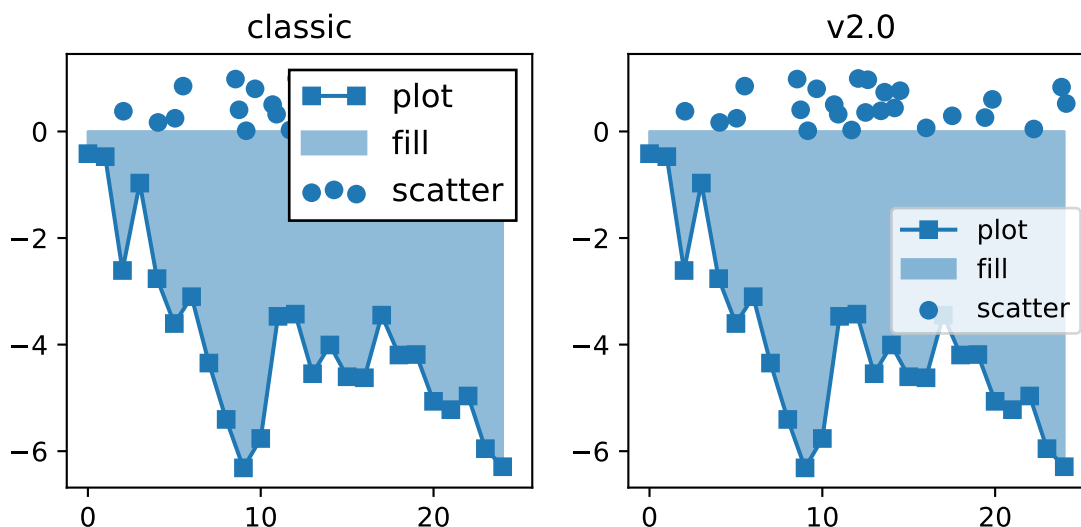
This rcParam is consulted when the text is drawn, not when the artist is created. Thus all `mathtext` on a given canvas will use the same fontset.

Legends

- By default, the number of points displayed in a legend is now 1.



- The default legend location is 'best', so the legend will be automatically placed in a location to minimize overlap with data.
- The legend defaults now include rounded corners, a lighter boundary, and partially transparent boundary and background.



The previous defaults can be restored by setting:

```
mpl.rcParams['legend.fancybox'] = False
mpl.rcParams['legend.loc'] = 'upper right'
mpl.rcParams['legend.numpoints'] = 2
mpl.rcParams['legend.fontsize'] = 'large'
mpl.rcParams['legend.framealpha'] = None
mpl.rcParams['legend.scatterpoints'] = 3
```

(continues on next page)

(continued from previous page)

```
mpl.rcParams['legend.edgecolor'] = 'inherit'
```

or by setting:

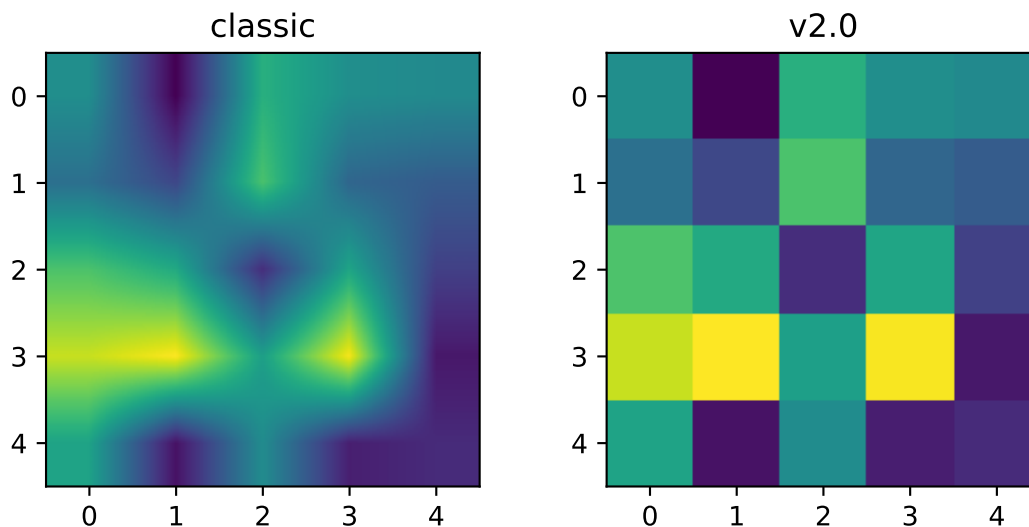
```
legend.fancybox      : False
legend.loc           : upper right
legend.numpoints     : 2      # the number of points in the legend line
legend.fontsize      : large
legend.framealpha    : None   # opacity of legend frame
legend.scatterpoints : 3      # number of scatter points
legend.edgecolor     : inherit # legend edge color ('inherit'
                               # means it uses axes.edgecolor)
```

in your matplotlibrc file.

Image

Interpolation

The default interpolation method for `imshow` is now 'nearest' and by default it resamples the data (both up and down sampling) before color mapping.



To restore the previous behavior set:

```
mpl.rcParams['image.interpolation'] = 'bilinear'
mpl.rcParams['image.resample'] = False
```

or set:

```
image.interpolation : bilinear # see help(imshow) for options
image.resample : False
```

in your matplotlibrc file.

Colormapping pipeline

Previously, the input data was normalized, then color mapped, and then resampled to the resolution required for the screen. This meant that the final resampling was being done in color space. Because the color maps are not generally linear in RGB space, colors not in the color map may appear in the final image. This bug was addressed by an almost complete overhaul of the image handling code.

The input data is now normalized, then resampled to the correct resolution (in normalized dataspace), and then color mapped to RGB space. This ensures that only colors from the color map appear in the final image. (If your viewer subsequently resamples the image, the artifact may reappear.)

The previous behavior cannot be restored.

Shading

- The default shading mode for light source shading, in `matplotlib.colors.LightSource.shade`, is now `overlay`. Formerly, it was `hsv`.

Plot layout

Auto limits

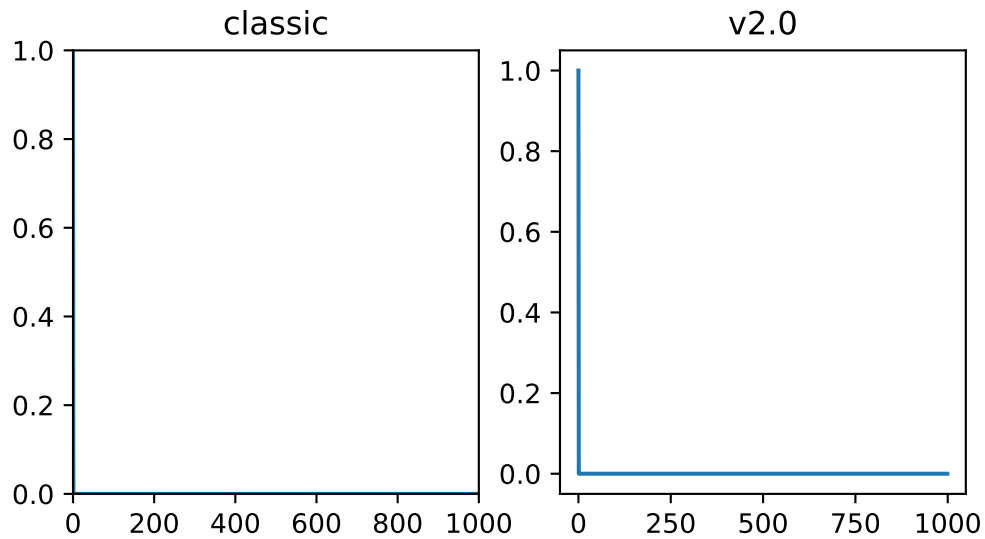
The previous auto-scaling behavior was to find 'nice' round numbers as view limits that enclosed the data limits, but this could produce bad plots if the data happened to fall on a vertical or horizontal line near the chosen 'round number' limit. The new default sets the view limits to 5% wider than the data range.

The size of the padding in the x and y directions is controlled by the `'axes.xmargin'` and `'axes.ymargin'` rcParams respectively. Whether the view limits should be 'round numbers' is controlled by the `'axes.autolimit_mode'` rcParam. In the original `'round_number'` mode, the view limits coincide with ticks.

The previous default can be restored by using:

```
mpl.rcParams['axes.autolimit_mode'] = 'round_numbers'
mpl.rcParams['axes.xmargin'] = 0
mpl.rcParams['axes.ymargin'] = 0
```

or setting:



```
axes.autolimit_mode: round_numbers
axes.xmargin: 0
axes.ymargin: 0
```

in your matplotlibrc file.

Z-order

- Ticks and grids are now plotted above solid elements such as filled contours, but below lines. To return to the previous behavior of plotting ticks and grids above lines, set `rcParams['axes.axisbelow'] = False`.

Ticks

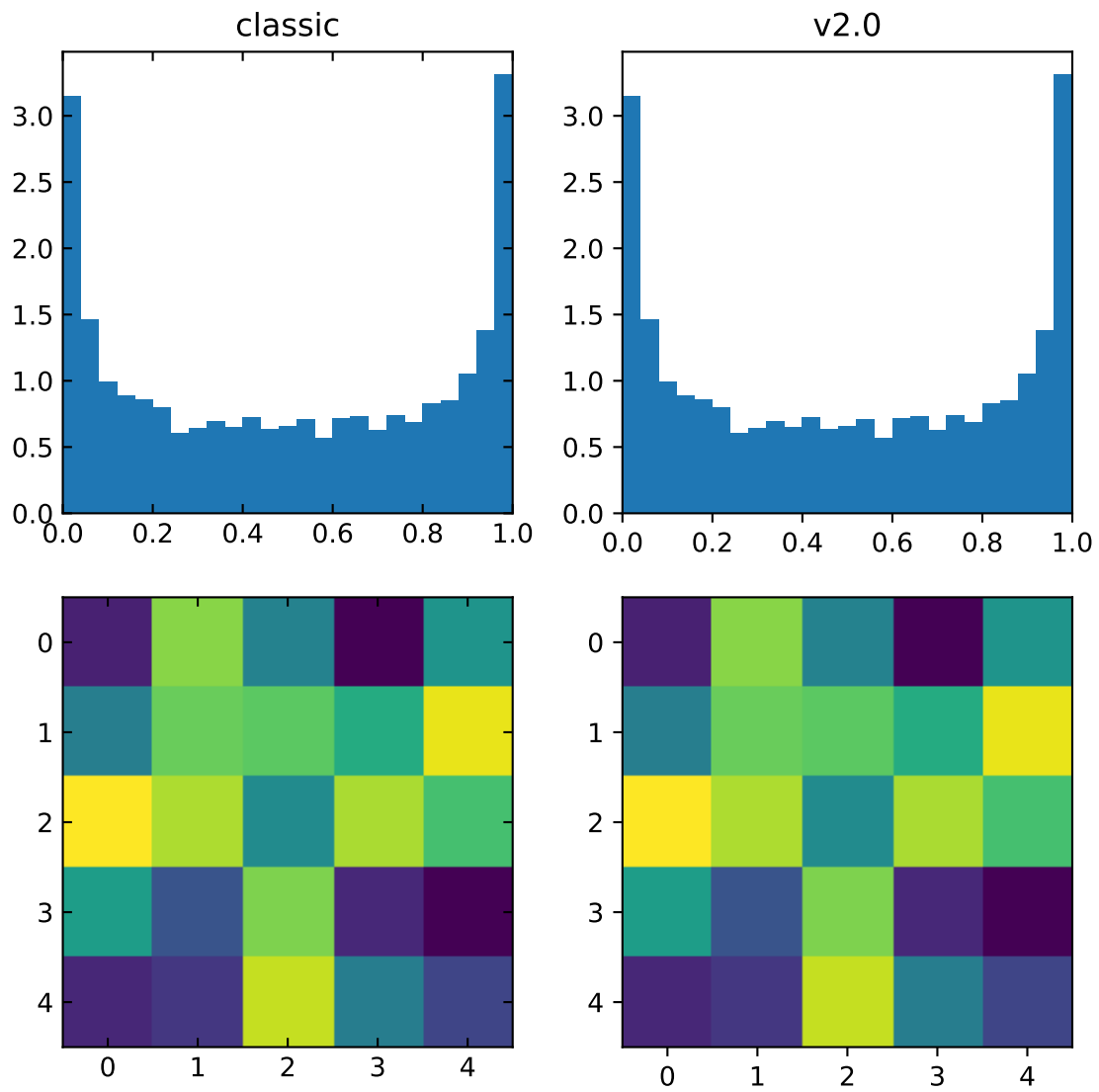
Direction

To reduce the collision of tick marks with data, the default ticks now point outward by default. In addition, ticks are now drawn only on the bottom and left spines to prevent a porcupine appearance, and for a cleaner separation between subplots.

To restore the previous behavior set:

```
mpl.rcParams['xtick.direction'] = 'in'
mpl.rcParams['ytick.direction'] = 'in'
mpl.rcParams['xtick.top'] = True
mpl.rcParams['ytick.right'] = True
```

or set:



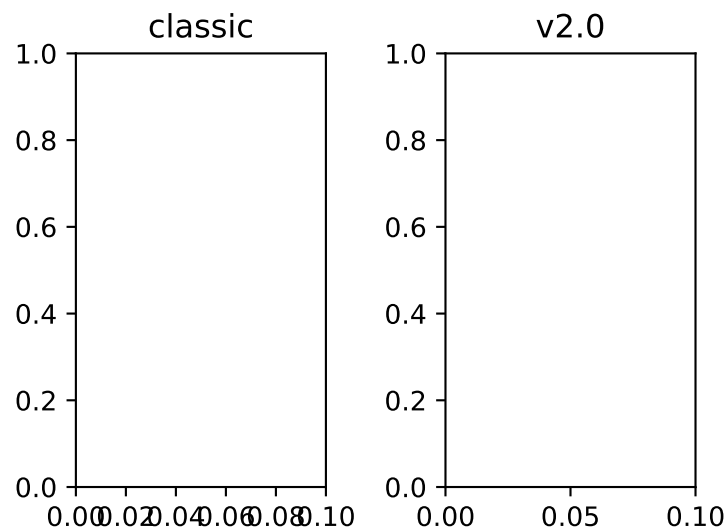

```
xtick.top: True
xtick.direction: in

ytick.right: True
ytick.direction: in
```

in your matplotlibrc file.

Number of ticks

The default Locator used for the x and y axis is AutoLocator which tries to find, up to some maximum number, 'nicely' spaced ticks. The locator now includes an algorithm to estimate the maximum number of ticks that will leave room for the tick labels. By default it also ensures that there are at least two ticks visible.



There is no way, other than using `mpl.style.use('classic')`, to restore the previous behavior as the default. On an axis-by-axis basis you may either control the existing locator via:

```
ax.xaxis.get_major_locator().set_params(nbins=9, steps=[1, 2, 5, 10])
```

or create a new `MaxNLocator`:

```
import matplotlib.ticker as mticker
ax.set_major_locator(mticker.MaxNLocator(nbins=9, steps=[1, 2, 5, 10]))
```

The algorithm used by `MaxNLocator` has been improved, and this may change the choice of tick locations in some cases. This also affects `AutoLocator`, which uses `MaxNLocator` internally.

For a log-scaled axis the default locator is the `LogLocator`. Previously the maximum number of ticks was set to 15, and could not be changed. Now there is a `numticks` kwarg for setting the maximum

to any integer value, to the string 'auto', or to its default value of None which is equivalent to 'auto'. With the 'auto' setting the maximum number will be no larger than 9, and will be reduced depending on the length of the axis in units of the tick font size. As in the case of the AutoLocator, the heuristic algorithm reduces the incidence of overlapping tick labels but does not prevent it.

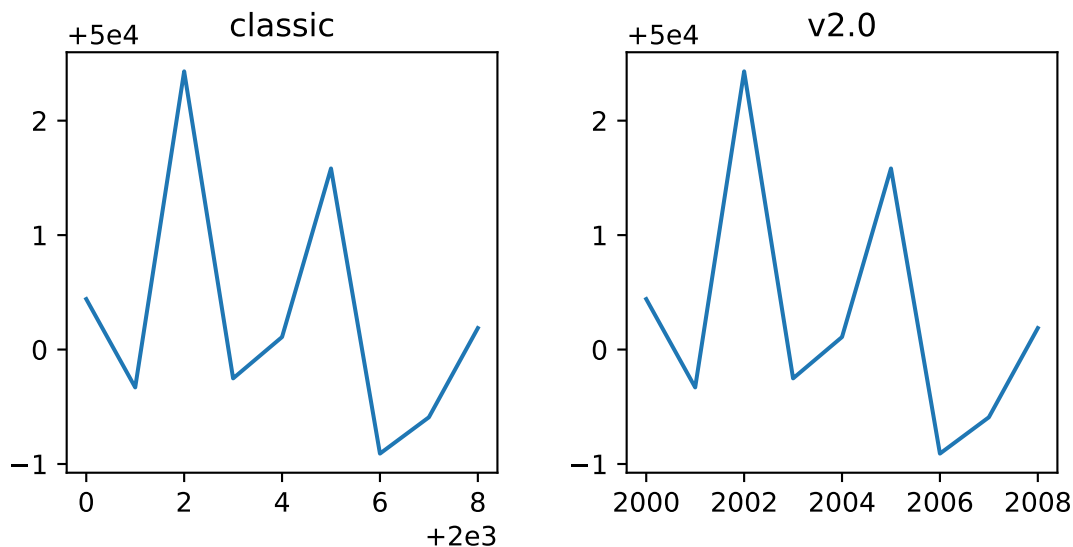
Tick label formatting

LogFormatter labeling of minor ticks

Minor ticks on a log axis are now labeled when the axis view limits span a range less than or equal to the interval between two major ticks. See LogFormatter for details. The minor tick labeling is turned off when using `mpl.style.use('classic')`, but cannot be controlled independently via rcParams.

ScalarFormatter tick label formatting with offsets

With the default of `rcParams['axes.formatter.useoffset'] = True`, an offset will be used when it will save 4 or more digits. This can be controlled with the new rcParam, `axes.formatter.offset_threshold`. To restore the previous behavior of using an offset to save 2 or more digits, use `rcParams['axes.formatter.offset_threshold'] = 2`.



AutoDateFormatter format strings

The default date formats are now all based on ISO format, i.e., with the slowest-moving value first. The date formatters are configurable through the `date.autoformatter.*` rcParams.

Threshold (tick interval >= than)	rcParam	classic	v2.0
365 days	'date.autoformatter.year'	'%Y'	'%Y'
30 days	'date.autoformatter.month'	'%b %Y'	'%Y-%m'
1 day	'date.autoformatter.day'	'%b %d %Y'	'%Y-%m-%d'
1 hour	'date.autoformatter.hour'	'%H:%M:%S'	'%H:%M'
1 minute	'date.autoformatter.minute'	'%H:%M:%S.%f'	'%H:%M:%S'
1 second	'date.autoformatter.second'	'%H:%M:%S.%f'	'%H:%M:%S'
1 microsecond	'date.autoformatter.microsecond'	'%H:%M:%S.%f'	'%H:%M:%S.%f'

Python's %x and %X date formats may be of particular interest to format dates based on the current locale.

The previous default can be restored by:

```
mpl.rcParams['date.autoformatter.year'] = '%Y'
mpl.rcParams['date.autoformatter.month'] = '%b %Y'
mpl.rcParams['date.autoformatter.day'] = '%b %d %Y'
mpl.rcParams['date.autoformatter.hour'] = '%H:%M:%S'
mpl.rcParams['date.autoformatter.minute'] = '%H:%M:%S.%f'
mpl.rcParams['date.autoformatter.second'] = '%H:%M:%S.%f'
mpl.rcParams['date.autoformatter.microsecond'] = '%H:%M:%S.%f'
```

or setting

```
date.autoformatter.year : %Y
date.autoformatter.month : %b %Y
date.autoformatter.day : %b %d %Y
date.autoformatter.hour : %H:%M:%S
date.autoformatter.minute : %H:%M:%S.%f
date.autoformatter.second : %H:%M:%S.%f
date.autoformatter.microsecond : %H:%M:%S.%f
```

in your matplotlibrc file.

mplot3d

- mplot3d now obeys some style-related rcParams, rather than using hard-coded defaults. These include:
 - xtick.major.width
 - ytick.major.width
 - xtick.color

- `ytick.color`
- `axes.linewidth`
- `axes.edgecolor`
- `grid.color`
- `grid.linewidth`
- `grid.linestyle`

Improved color conversion API and RGBA support

The colors gained a new color conversion API with full support for the alpha channel. The main public functions are `is_color_like()`, `matplotlib.colors.to_rgba()`, `matplotlib.colors.to_rgba_array()` and `to_hex()`. RGBA quadruplets are encoded in hex format as `#rrggbbaa`.

A side benefit is that the Qt options editor now allows setting the alpha channel of the artists as well.

New Configuration (`rcParams`)

New `rcparams` added

Parameter	Description
<code>date.autoformatter.year</code>	format string for 'year' scale dates
<code>date.autoformatter.month</code>	format string for 'month' scale dates
<code>date.autoformatter.day</code>	format string for 'day' scale dates
<code>date.autoformatter.hour</code>	format string for 'hour' scale times
<code>date.autoformatter.minute</code>	format string for 'minute' scale times
<code>date.autoformatter.second</code>	format string for 'second' scale times
<code>date.autoformatter.microsecond</code>	format string for 'microsecond' scale times
<code>scatter.marker</code>	default marker for scatter plot
<code>svg.hashsalt</code>	see note
<code>xtick.top</code> , <code>xtick.minor.top</code> , <code>xtick.major.top</code> , <code>xtick.bottom</code> , <code>xtick.minor.bottom</code> , <code>xtick.major.bottom</code> , <code>ytick.left</code> , <code>ytick.minor.left</code> , <code>ytick.major.left</code> , <code>ytick.right</code> , <code>ytick.minor.right</code> , <code>ytick.major.right</code>	Control where major and minor ticks are drawn. The global values are and ed with the corresponding major/minor values.
<code>hist.bins</code>	The default number of bins to use in hist. This can be an <code>int</code> , a list of floats, or 'auto' if numpy ≥ 1.11 is installed.
<code>lines.scale_dashes</code>	Whether the line dash patterns should scale with linewidth.
<code>axes.formatter.offset_threshold</code>	Minimum number of digits saved in tick labels that triggers using an offset.

Added `svg.hashsalt` key to `rcParams`

If `svg.hashsalt` is `None` (which it is by default), the `svg` backend uses `uuid4` to generate the hash salt. If it is not `None`, it must be a string that is used as the hash salt instead of `uuid4`. This allows for deterministic SVG output.

Removed the `svg.image_noscale` `rcParam`

As a result of the extensive changes to image handling, the `svg.image_noscale` `rcParam` has been removed. The same functionality may be achieved by setting `interpolation='none'` on individual images or globally using the `image.interpolation` `rcParam`.

Qualitative colormaps

ColorBrewer's "qualitative" colormaps ("Accent", "Dark2", "Paired", "Pastel1", "Pastel2", "Set1", "Set2", "Set3") were intended for discrete categorical data, with no implication of value, and therefore have been converted to `ListedColormap` instead of `LinearSegmentedColormap`, so the colors will no longer be interpolated and they can be used for choropleths, labeled image features, etc.

Axis offset label now responds to `labelcolor`

Axis offset labels are now colored the same as axis tick markers when `labelcolor` is altered.

Improved offset text choice

The default offset-text choice was changed to only use significant digits that are common to all ticks (e.g. 1231..1239 -> 1230, instead of 1231), except when they straddle a relatively large multiple of a power of ten, in which case that multiple is chosen (e.g. 1999..2001->2000).

Style parameter blacklist

In order to prevent unexpected consequences from using a style, style files are no longer able to set parameters that affect things unrelated to style. These parameters include:

```
'interactive', 'backend', 'backend.qt4', 'webagg.port',  
'webagg.port_retries', 'webagg.open_in_browser', 'backend_fallback',  
'toolbar', 'timezone', 'datapath', 'figure.max_open_warning',  
'savefig.directory', 'tk.window_focus', 'docstring.hardcopy'
```

Change in default font

The default font used by matplotlib in text has been changed to DejaVu Sans and DejaVu Serif for the sans-serif and serif families, respectively. The DejaVu font family is based on the previous matplotlib default –Bitstream Vera– but includes a much wider range of characters.

The default mathtext font has been changed from Computer Modern to the DejaVu family to maintain consistency with regular text. Two new options for the `mathtext.fontset` configuration parameter have been added: `dejavusans` (default) and `dejavuserif`. Both of these options use DejaVu glyphs whenever possible and fall back to STIX symbols when a glyph is not found in DejaVu. To return to the previous behavior, set the rcParam `mathtext.fontset` to `cm`.

Faster text rendering

Rendering text in the Agg backend is now less fuzzy and about 20% faster to draw.

Improvements for the Qt figure options editor

Various usability improvements were implemented for the Qt figure options editor, among which:

- Line style entries are now sorted without duplicates.
- The colormap and normalization limits can now be set for images.
- Line edits for floating values now display only as many digits as necessary to avoid precision loss. An important bug was also fixed regarding input validation using Qt5 and a locale where the decimal separator is `”`, `”`.
- The axes selector now uses shorter, more user-friendly names for axes, and does not crash if there are no axes.
- Line and image entries using the default labels (`”_lineX”`, `”_imageX”`) are now sorted numerically even when there are more than 10 entries.

Improved image support

Prior to version 2.0, matplotlib resampled images by first applying the color map and then resizing the result. Since the resampling was performed on the colored image, this introduced colors in the output image that didn’t actually exist in the color map. Now, images are resampled first (and entirely in floating-point, if the input image is floating-point), and then the color map is applied.

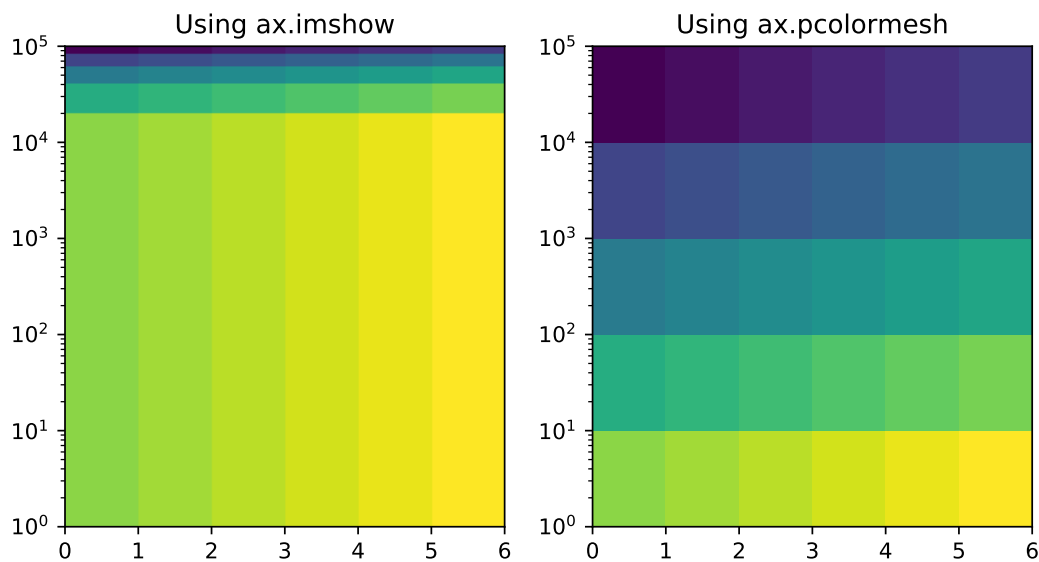
In order to make this important change, the image handling code was almost entirely rewritten. As a side effect, image resampling uses less memory and fewer datatype conversions than before.

The experimental private feature where one could “skew” an image by setting the private member `_image_skew_coordinate` has been removed. Instead, images will obey the transform of the axes on which they are drawn.

Non-linear scales on image plots

`imshow()` now draws data at the requested points in data space after the application of non-linear scales.

The image on the left demonstrates the new, correct behavior. The old behavior can be recreated using `pcolormesh()` as demonstrated on the right.



This can be understood by analogy to plotting a histogram with linearly spaced bins with a logarithmic x-axis. Equal sized bins will be displayed as wider for small x and narrower for large x .

Support for HiDPI (Retina) displays in the NbAgg and WebAgg backends

The NbAgg and WebAgg backends will now use the full resolution of your high-pixel-density display.

Change in the default animation codec

The default animation codec has been changed from `mpeg4` to `h264`, which is more efficient. It can be set via the `animation.codec` rcParam.

Deprecated support for mencoder in animation

The use of `mencoder` for writing video files with `mpl` is problematic; switching to `ffmpeg` is strongly advised. All support for `mencoder` will be removed in version 2.2.

Boxplot Zorder Keyword Argument

The `zorder` parameter now exists for `boxplot()`. This allows the zorder of a boxplot to be set in the plotting function call.

```
boxplot(np.arange(10), zorder=10)
```

Filled + and x markers

New fillable plus and x markers have been added. See the markers module and marker reference examples.

rcount and ccount for `plot_surface()`

As of v2.0, `mplot3d`'s `plot_surface()` now accepts `rcount` and `ccount` arguments for controlling the sampling of the input data for plotting. These arguments specify the maximum number of evenly spaced samples to take from the input data. These arguments are also the new default sampling method for the function, and is considered a style change.

The old `rstride` and `cstride` arguments, which specified the size of the evenly spaced samples, become the default when 'classic' mode is invoked, and are still available for use. There are no plans for deprecating these arguments.

Streamplot Zorder Keyword Argument Changes

The `zorder` parameter for `streamplot()` now has default value of `None` instead of `2`. If `None` is given as `zorder`, `streamplot()` has a default `zorder` of `matplotlib.lines.Line2D.zorder`.

Extension to `matplotlib.backend_bases.GraphicsContextBase`

To support standardizing hatch behavior across the backends we ship the `matplotlib.backend_bases.GraphicsContextBase.get_hatch_color` method as added to `matplotlib.backend_bases.GraphicsContextBase`. This is only used during the render process in the backends we ship so will not break any third-party backends.

If you maintain a third-party backend which extends `GraphicsContextBase` this method is now available to you and should be used to color hatch patterns.

5.3.12 New in Matplotlib 2.1.0

Documentation

The examples have been migrated to use [sphinx gallery](#). This allows better mixing of prose and code in the examples, provides links to download the examples as both a Python script and a

Jupyter notebook, and improves the thumbnail galleries. The examples have been re-organized into [Tutorials](#) and a gallery.

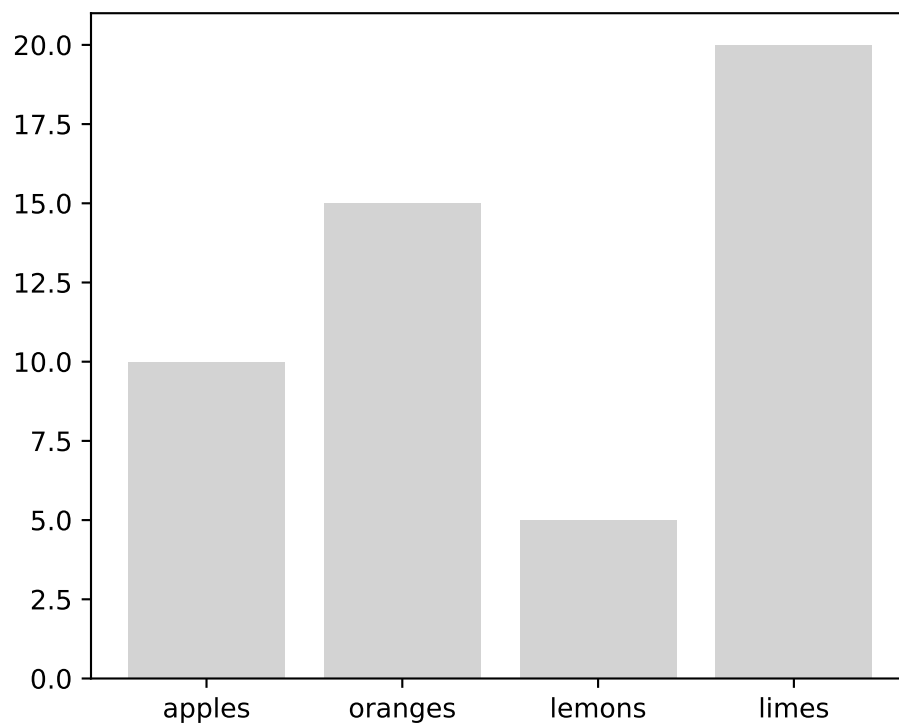
Many docstrings and examples have been clarified and improved.

New features

String categorical values

All plotting functions now support string categorical values as input. For example:

```
data = {'apples': 10, 'oranges': 15, 'lemons': 5, 'limes': 20}
fig, ax = plt.subplots()
ax.bar(data.keys(), data.values(), color='lightgray')
```



Interactive JS widgets for animation

Jake Vanderplas' JSAnimation package has been merged into Matplotlib. This adds to Matplotlib the `HTMLWriter` class for generating a JavaScript HTML animation, suitable for the IPython notebook. This can be activated by default by setting the `animation.html` rc parameter to `jshtml`.

One can also call the `to_jshtml` method to manually convert an animation. This can be displayed using IPython's HTML display class:

```
from IPython.display import HTML
HTML(animation.to_jshtml())
```

The `HTMLWriter` class can also be used to generate an HTML file by asking for the html writer.

Enhancements to polar plot

The polar axes transforms have been greatly re-factored to allow for more customization of view limits and tick labelling. Additional options for view limits allow for creating an annulus, a sector, or some combination of the two.

The `set_rorigin()` method may be used to provide an offset to the minimum plotting radius, producing an annulus.

The `set_theta_zero_location()` method now has an optional offset argument. This argument may be used to further specify the zero location based on the given anchor point.

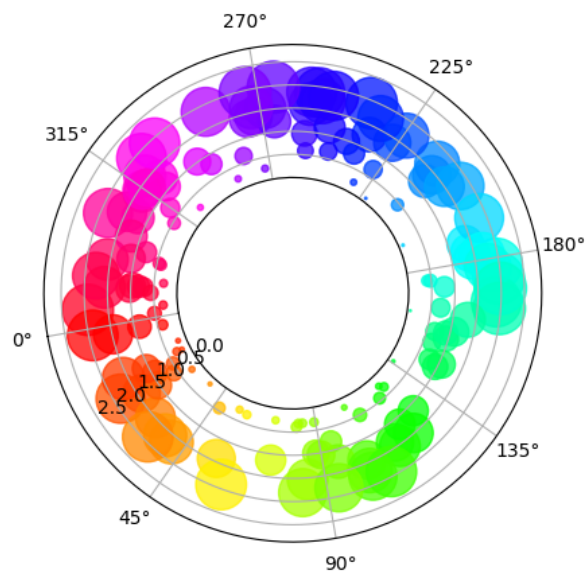


Fig. 30: Polar Offset Demo

The `set_thetamin()` and `set_thetamax()` methods may be used to limit the range of angles plotted, producing sectors of a circle.

Previous releases allowed plots containing negative radii for which the negative values are simply used as labels, and the real radius is shifted by the configured minimum. This release also allows negative radii to be used for grids and ticks, which were previously silently ignored.

Radial ticks have been modified to be parallel to the circular grid line, and angular ticks have been modified to be parallel to the grid line. It may also be useful to rotate tick labels to match the

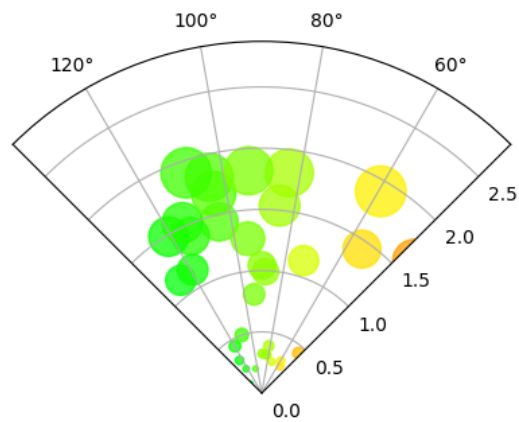


Fig. 31: Polar Sector Demo

boundary. Calling `ax.tick_params(rotation='auto')` will enable the new behavior: radial tick labels will be parallel to the circular grid line, and angular tick labels will be perpendicular to the grid line (i.e., parallel to the outer boundary). Additionally, tick labels now obey the padding settings that previously only worked on Cartesian plots. Consequently, the `frac` argument to `PolarAxes.set_thetagrids` is no longer applied. Tick padding can be modified with the `pad` argument to `Axes.tick_params` or `Axis.set_tick_params`.

Figure class now has `subplots` method

The Figure class now has a `subplots()` method which behaves the same as `pyplot.subplots()` but on an existing figure.

Metadata `savefig` keyword argument

`savefig()` now accepts metadata as a keyword argument. It can be used to store key/value pairs in the image metadata.

- 'png' with Agg backend
- 'pdf' with PDF backend (see `writeInfoDict()` for a list of supported keywords)
- 'eps' and 'ps' with PS backend (only 'Creator' key is accepted)

```
plt.savefig('test.png', metadata={'Software': 'My awesome software'})
```

Busy Cursor

The interactive GUI backends will now change the cursor to busy when Matplotlib is rendering the canvas.

PolygonSelector

A `PolygonSelector` class has been added to `matplotlib.widgets`. See /gallery/widgets/polygon_selector_demo for details.

Added `matplotlib.ticker.PercentFormatter`

The new `PercentFormatter` formatter has some nice features like being able to convert from arbitrary data scales to percents, a customizable percent symbol and either automatic or manual control over the decimal points.

Reproducible PS, PDF and SVG output

The `SOURCE_DATE_EPOCH` environment variable can now be used to set the timestamp value in the PS and PDF outputs. See [source date epoch](#).

Alternatively, calling `savefig` with `metadata={'CreationDate': None}` will omit the timestamp altogether for the PDF backend.

The reproducibility of the output from the PS and PDF backends has so far been tested using various plot elements but only default values of options such as `{ps,pdf}.fonttype` that can affect the output at a low level, and not with the `mathtext` or `usetex` features. When Matplotlib calls external tools (such as PS distillers or LaTeX) their versions need to be kept constant for reproducibility, and they may add sources of nondeterminism outside the control of Matplotlib.

For SVG output, the `svg.hashsalt rc` parameter has been added in an earlier release. This parameter changes some random identifiers in the SVG file to be deterministic. The downside of this setting is that if more than one file is generated using deterministic identifiers and they end up as parts of one larger document, the identifiers can collide and cause the different parts to affect each other.

These features are now enabled in the tests for the PDF and SVG backends, so most test output files (but not all of them) are now deterministic.

Orthographic projection for `mplot3d`

`Axes3D` now accepts `proj_type` keyword argument and has a method `set_proj_type()`. The default option is 'persp' as before, and supplying 'ortho' enables orthographic view.

Compare the z-axis which is vertical in orthographic view, but slightly skewed in the perspective view.

```

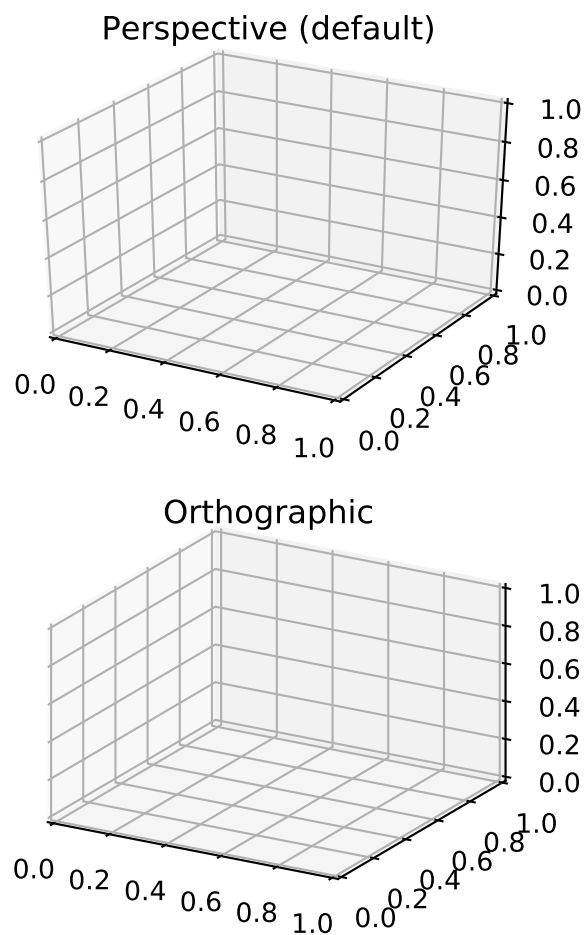
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(4, 6))
ax1 = fig.add_subplot(2, 1, 1, projection='3d')
ax1.set_proj_type('persp')
ax1.set_title('Perspective (default)')

ax2 = fig.add_subplot(2, 1, 2, projection='3d')
ax2.set_proj_type('ortho')
ax2.set_title('Orthographic')

plt.show()

```



voxels function for mplot3d

Axes3D now has a `voxels` method, for visualizing boolean 3D data. Uses could include plotting a sparse 3D heat map, or visualizing a volumetric model.

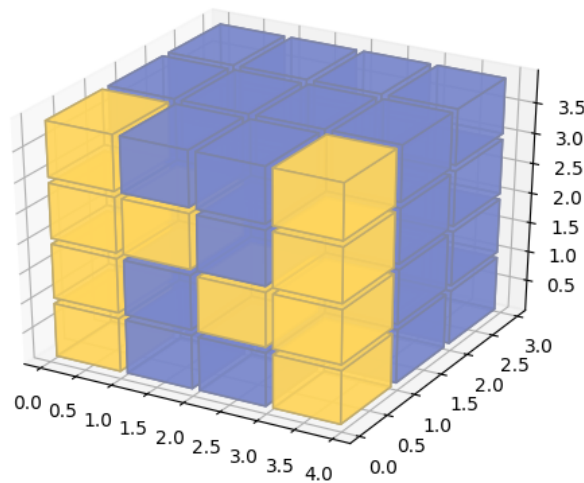


Fig. 32: Voxel Demo

Improvements

CheckButtons widget `get_status` function

A `get_status()` method has been added to the `matplotlib.widgets.CheckButtons` class. This `get_status` method allows user to query the status (True/False) of all of the buttons in the CheckButtons object.

Add `fill_bar` argument to `AnchoredSizeBar`

The `mpl_toolkits` class `AnchoredSizeBar` now has an additional `fill_bar` argument, which makes the size bar a solid rectangle instead of just drawing the border of the rectangle. The default is `None`, and whether or not the bar will be filled by default depends on the value of `size_vertical`. If `size_vertical` is nonzero, `fill_bar` will be set to `True`. If `size_vertical` is zero then `fill_bar` will be set to `False`. If you wish to override this default behavior, set `fill_bar` to `True` or `False` to unconditionally always or never use a filled patch rectangle for the size bar.

```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1.anchored_artists import AnchoredSizeBar
```

(continues on next page)

(continued from previous page)

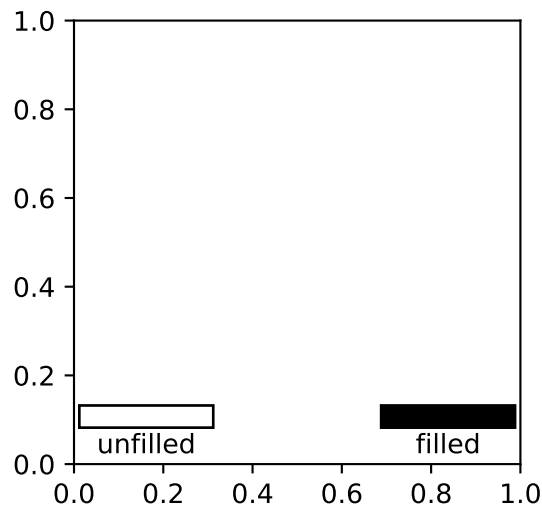
```

fig, ax = plt.subplots(figsize=(3, 3))

bar0 = AnchoredSizeBar(ax.transData, 0.3, 'unfilled', loc=3, frameon=False,
                        size_vertical=0.05, fill_bar=False)
ax.add_artist(bar0)
bar1 = AnchoredSizeBar(ax.transData, 0.3, 'filled', loc=4, frameon=False,
                        size_vertical=0.05, fill_bar=True)
ax.add_artist(bar1)

plt.show()

```



Annotation can use a default arrow style

Annotations now use the default arrow style when setting `arrowprops={}`, rather than no arrow (the new behavior actually matches the documentation).

Barbs and Quiver Support Dates

When using the `quiver()` and `barbs()` plotting methods, it is now possible to pass dates, just like for other methods like `plot()`. This also allows these functions to handle values that need unit-conversion applied.

Hexbin default line color

The default `linecolor` keyword argument for `hexbin()` is now `'face'`, and supplying `'none'` now prevents lines from being drawn around the hexagons.

Figure.legend() can be called without arguments

Calling `Figure.legend()` can now be done with no arguments. In this case a legend will be created that contains all the artists on all the axes contained within the figure.

Multiple legend keys for legend entries

A legend entry can now contain more than one legend key. The extended `HandlerTuple` class now accepts two parameters: `ndivide` divides the legend area in the specified number of sections; `pad` changes the padding between the legend keys.

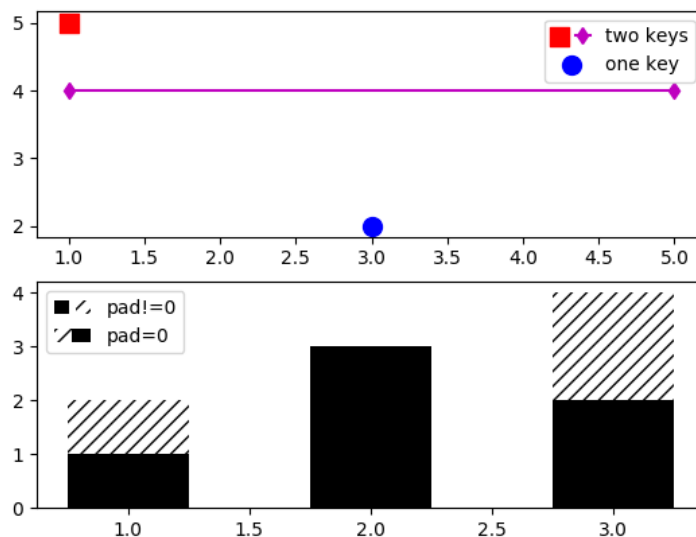


Fig. 33: Multiple Legend Keys

New parameter `clear` for `figure()`

When the pyplot's function `figure()` is called with a `num` parameter, a new window is only created if no existing window with the same value exists. A new bool parameter `clear` was added for explicitly clearing its existing contents. This is particularly useful when utilized in interactive sessions. Since `subplots()` also accepts keyword arguments from `figure()`, it can also be used there:


```
import matplotlib.pyplot as plt

fig0 = plt.figure(num=1)
fig0.suptitle("A fancy plot")
print("fig0.texts: ", [t.get_text() for t in fig0.texts])

fig1 = plt.figure(num=1, clear=False) # do not clear contents of window
fig1.text(0.5, 0.5, "Really fancy!")
print("fig0 is fig1: ", fig0 is fig1)
print("fig1.texts: ", [t.get_text() for t in fig1.texts])

fig2, ax2 = plt.subplots(2, 1, num=1, clear=True) # clear contents
print("fig0 is fig2: ", fig0 is fig2)
print("fig2.texts: ", [t.get_text() for t in fig2.texts])

# The output:
# fig0.texts: ['A fancy plot']
# fig0 is fig1: True
# fig1.texts: ['A fancy plot', 'Really fancy!']
# fig0 is fig2: True
# fig2.texts: []
```

Specify minimum value to format as scalar for `LogFormatterMathtext`

`LogFormatterMathtext` now includes the option to specify a minimum value exponent to format as a scalar (i.e., 0.001 instead of 10^{-3}).

New `quiverkey` angle keyword argument

Plotting a `quiverkey()` now admits the `angle` keyword argument, which sets the angle at which to draw the key arrow.

`Colormap` reversed method

The methods `matplotlib.colors.LinearSegmentedColormap.reversed()` and `matplotlib.colors.ListedColormap.reversed()` return a reversed instance of the `Colormap`. This implements a way for any `Colormap` to be reversed.

`Artist.setp` (and `pyplot.setp`) accept a file argument

The argument is keyword-only. It allows an output file other than `sys.stdout` to be specified. It works exactly like the file argument to `print`.

streamplot streamline generation more configurable

The starting point, direction, and length of the stream lines can now be configured. This allows to follow the vector field for a longer time and can enhance the visibility of the flow pattern in some use cases.

Axis.set_tick_params now responds to rotation

Bulk setting of tick label rotation is now possible via `tick_params()` using the rotation keyword.

```
ax.tick_params(which='both', rotation=90)
```

Ticklabels are turned off instead of being invisible

Internally, the Tick's `labelOn()` attribute is now used to hide tick labels instead of setting the visibility on the tick label objects. This improves overall performance and fixes some issues. As a consequence, in case those labels ought to be shown, `tick_params()` needs to be used, e.g.

```
ax.tick_params(labelbottom=True)
```

Shading in 3D bar plots

A new shade parameter has been added the 3D bar plotting method. The default behavior remains to shade the bars, but now users have the option of setting shade to `False`.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

x = np.arange(2)
y = np.arange(3)
x2d, y2d = np.meshgrid(x, y)
x, y = x2d.ravel(), y2d.ravel()
z = np.zeros_like(x)
dz = x + y

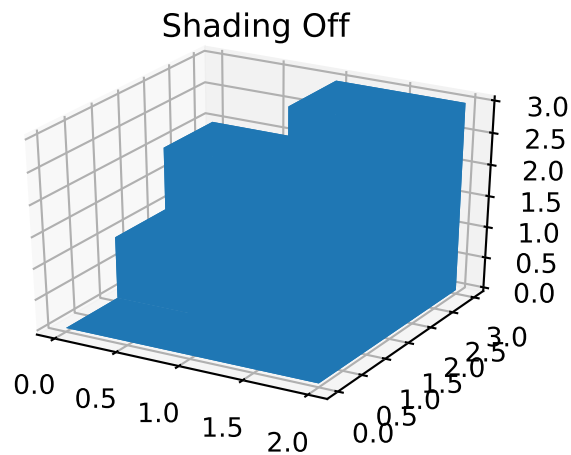
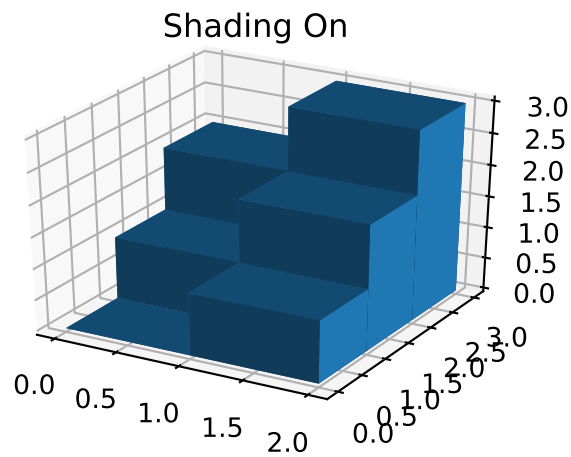
fig = plt.figure(figsize=(4, 6))
ax1 = fig.add_subplot(2, 1, 1, projection='3d')
ax1.bar3d(x, y, z, 1, 1, dz, shade=True)
ax1.set_title('Shading On')

ax2 = fig.add_subplot(2, 1, 2, projection='3d')
ax2.bar3d(x, y, z, 1, 1, dz, shade=False)
```

(continues on next page)

(continued from previous page)

```
ax2.set_title('Shading Off')  
  
plt.show()
```



New which Parameter for autofmt_xdate

A which parameter now exists for the method `autofmt_xdate()`. This allows a user to format major, minor or both tick labels selectively. The default behavior will rotate and align the major tick labels.

```
fig.autofmt_xdate(bottom=0.2, rotation=30, ha='right', which='minor')
```

New Figure Parameter for subplot2grid

A `fig` parameter now exists for the function `subplot2grid()`. This allows a user to specify the figure where the subplots will be created. If `fig` is `None` (default) then the method will use the current figure retrieved by `gcf()`.

```
subplot2grid(shape, loc, rowspan=1, colspan=1, fig=myfig)
```

Interpolation in fill_betweenx

The `interpolate` parameter now exists for the method `fill_betweenx()`. This allows a user to interpolate the data and fill the areas in the crossover points, similarly to `fill_between()`.

New keyword argument `sep` for EngFormatter

A new `sep` keyword argument has been added to `EngFormatter` and provides a means to define the string that will be used between the value and its unit. The default string is `" "`, which preserves the former behavior. Additionally, the separator is now present between the value and its unit even in the absence of SI prefix. There was formerly a bug that was causing strings like `"3.14V"` to be returned instead of the expected `"3.14 V"` (with the default behavior).

Extend MATPLOTLIBRC behavior

The environmental variable can now specify the full file path or the path to a directory containing a `matplotlibrc` file.

density kwarg to hist

The `hist()` method now prefers `density` to `normed` to control if the histogram should be normalized, following a change upstream to NumPy. This will reduce confusion as the behavior has always been that the integral of the histogram is 1 (rather than sum or maximum value).

Internals

New TransformedPatchPath caching object

A newly added `TransformedPatchPath` provides a means to transform a `Patch` into a `Path` via a `Transform` while caching the resulting path. If neither the patch nor the transform have changed, a cached copy of the path is returned.

This class differs from the older `TransformedPath` in that it is able to refresh itself based on the underlying patch while the older class uses an immutable path.

Abstract base class for movie writers

The new `AbstractMovieWriter` class defines the API required by a class that is to be used as the writer in the `matplotlib.animation.Animation.save()` method. The existing `MovieWriter` class now derives from the new abstract base class.

Stricter validation of line style rcParams

The validation of rcParams that are related to line styles (`lines.linestyle`, `boxplot.*.linestyle`, `grid.linestyle` and `contour.negative_linestyle`) now effectively checks that the values are valid line styles. Strings like 'dashed' or '--' are accepted, as well as even-length sequences of on-off ink like `[1, 1.65]`. In this latter case, the offset value is handled internally and should not be provided by the user.

The new validation scheme replaces the former one used for the `contour.negative_linestyle` rcParams, that was limited to 'solid' and 'dashed' line styles.

The validation is case-insensitive. The following are now valid:

```
grid.linestyle          : (1, 3)  # loosely dotted grid lines
contour.negative_linestyle : dashdot # previously only solid or dashed
```

pytest

The automated tests have been switched from nose to pytest.

Performance

Path simplification updates

Line simplification controlled by the `path.simplify` and `path.simplify_threshold` parameters has been improved. You should notice better rendering performance when plotting large amounts of data (as long as the above parameters are set accordingly). Only the line segment portion of paths will be simplified – if you are also drawing markers and experiencing problems with rendering speed, you should consider using the `markevery` option to plot. See the [Performance](#) section in the usage tutorial for more information.

The simplification works by iteratively merging line segments into a single vector until the next line segment's perpendicular distance to the vector (measured in display-coordinate space) is greater than the `path.simplify_threshold` parameter. Thus, higher values of `path.simplify_threshold` result in quicker rendering times. If you are plotting just to explore data and not for publication quality, pixel perfect plots, then a value of 1.0 can be safely used. If you want to make sure your plot reflects your data exactly, then you should set `path.simplify` to false and/or `path.simplify_threshold` to 0. Matplotlib currently defaults to a conservative value of 1/9, smaller values are unlikely to cause any visible differences in your plots.

Implement `intersects_bbox` in `c++`

`intersects_bbox()` has been implemented in `c++` which improves the performance of automatically placing the legend.

GITHUB STATS

GitHub stats for 2018/08/10 - 2019/02/22 (tag: v2.2.3)

These lists are automatically generated, and may be incomplete or contain duplicates.

We closed 16 issues and merged 47 pull requests. The full list can be seen [on GitHub](#)

The following 21 authors contributed 122 commits.

- Abhinuv Nitin Pitale
- Andrew Nelson
- Antony Lee
- Ayappan P
- Bruno Pagani
- Chris Zimmerman
- David Stansby
- Elizabeth Seiver
- Elliott Sales de Andrade
- Eric Wieser
- ImportanceOfBeingErnest
- Jody Klymak
- Kyle Sunden
- Mark Harfouche
- MeeseeksMachine
- Nelle Varoquaux
- Nick Papior
- Ryan May
- simon-kraeusel
- Thomas A Caswell

- Tim Hoffmann

GitHub issues and pull requests:

Pull Requests (47):

- [PR #13462](#): Merge pull request [#13455](#) from [tacaswell/fix_freetype_download_logic](#)
- [PR #13455](#): BLD: only try to get freetype src if src does not exist
- [PR #13445](#): Manual backport of [#11246](#)
- [PR #11246](#): Download jquery during build.
- [PR #13420](#): Doc backports
- [PR #13415](#): Backport [PR #13405](#) on branch v2.2.x (Fix `imshow()`ing PIL-opened images.)
- [PR #13405](#): Fix `imshow()`ing PIL-opened images.
- [PR #12811](#): Backport [PR #12809](#) on branch v2.2.x (Fix `TypeError` when calculating `tick_values`)
- [PR #13181](#): Backport [PRs #12154](#), [#12294](#), [#12297](#), [#12316](#), [#13159](#) & [#13205](#) to fix multiple tests issues
- [PR #12407](#): Fix extension building on Xcode 10
- [PR #12879](#): Pin `pytest` to `<3.8` (for 2.2.x)
- [PR #12809](#): Fix `TypeError` when calculating `tick_values`
- [PR #12618](#): Backport [PR #12548](#) on branch v2.2.x (`undef __XOPEN_SOURCE` breaks the build in AIX)
- [PR #12548](#): `undef __XOPEN_SOURCE` breaks the build in AIX
- [PR #12573](#): BUG: `mplot3d`: Don't crash if `azim` or `elev` are non-integral
- [PR #12508](#): Backport [PR #12478](#) on branch v2.2.x (MAINT: `numpy` deprecates `asscalar` in 1.16)
- [PR #12478](#): MAINT: `numpy` deprecates `asscalar` in 1.16
- [PR #12366](#): TST: Update test images for new Ghostscript.
- [PR #12346](#): Backport [PR #12261](#): FIX: `parasite axis2` demo
- [PR #12343](#): Backport [PR #12321](#) on branch v2.2.x (maint: `setuptools` for freetype had a Catch case for missing `ft2build.h`)
- [PR #12321](#): maint: `setuptools` for freetype had a Catch case for missing `ft2build.h`
- [PR #12316](#): Fix some warnings from Travis
- [PR #12261](#): FIX: `parasite axis2` demo
- [PR #12154](#): Avoid triggering deprecation warnings with `pytest 3.8`.
- [PR #12148](#): BLD: pragmatic fix for building `basic_unit` example on `py37`

- [PR #12125](#): Backport [PR #12117](#) on branch v2.2.x (Fix Agg extent calculations for empty draws)
- [PR #12127](#): Fix circleCI on 2.2.x
- [PR #12117](#): Fix Agg extent calculations for empty draws
- [PR #12101](#): Backport [PR #12091](#) on branch v2.2.x (Respect QT_API even when the backend is not Qt{4,5}{Agg,Cairo}.)
- [PR #12102](#): Backport [PR #12091](#) on branch v3.0.x (Respect QT_API even when the backend is not Qt{4,5}{Agg,Cairo}.)
- [PR #12091](#): Respect QT_API even when the backend is not Qt{4,5}{Agg,Cairo}.
- [PR #12097](#): Backport [PR #12092](#): Update backend_qt5agg to fix PySide2 mem issues
- [PR #12092](#): Update backend_qt5agg to fix PySide2 mem issues
- [PR #12082](#): Backport [PR #12012](#) on branch v2.2.x (FIX: fallback text renderer to fig._cachedRenderer, if none found)
- [PR #12012](#): FIX: fallback text renderer to fig._cachedRenderer, if none found
- [PR #11957](#): FIX: fix importing backend with non-ascii characters
- [PR #11990](#): Backport [PR #11988](#) on branch v2.2.x
- [PR #11988](#): Updated typo in fill_between docstring
- [PR #11977](#): Backport [PR #11973](#) on branch v2.2.x
- [PR #11973](#): TST: Fix duplicate file usage when running in parallel.
- [PR #11900](#): Allow args to pass through _allow_super_init
- [PR #11841](#): Backport [PR #11762](#) on branch v2.2.x
- [PR #11762](#): fix minor grid overlapping
- [PR #11808](#): updates error message to run pytest instead of tests.py standalone
- [PR #11786](#): Fix collections import warnings for Python 3.7
- [PR #11724](#): Fix cairo's image inversion and alpha misapplication.
- [PR #11733](#): MAINT: use collections.abc for 3.7

Issues (16):

- [#11199](#): Open CVE on vendored jquery
- [#12766](#): Incorrect license info in setup.py
- [#13258](#): backport to 2.x fixes for sphinx 1.8?
- [#12140](#): plt.imshow() error on PIL-opened tiff images
- [#12119](#): pypi wheels for 2.2.3 claim to require numpy 1.7, but really require numpy 1.10
- [#13344](#): Figure instances show shared content when redrawn with pycairo

- [#13182](#): Tests errors related to 'MarkDecorator'
- [#13192](#): Use of np.asscalar in Matplotlib v2.2.3
- [#13031](#): Seemingly irrelevant exception about sequence multiplication happening when linestyle used
- [#12535](#): AIX Support - Do not undef __XOPEN__SOURCE
- [#11955](#): UnicodeDecodeError on importing pyplot in python2
- [#12208](#): parasite axis2 demo now crashes with log x-axis
- [#11758](#): Extra Grid Line On Plot (PyPlot)
- [#12003](#): Plotting empty lists with rasterized=True causes crash on saving to pdf
- [#12089](#): PySide2 memory leak during canvas paintEvent
- [#10874](#): RuntimeError: Cannot get window extent w/o renderer

6.1 Previous Github Stats

6.1.1 GitHub Stats

GitHub stats for 2018/03/05 - 2018/08/09 (tag: v2.2.0)

These lists are automatically generated, and may be incomplete or contain duplicates.

We closed 27 issues and merged 101 pull requests. The full list can be seen [on GitHub](#)

The following 28 authors contributed 347 commits.

- 816-8055
- Andreas Gustafsson
- Andrew Nelson
- Antony Lee
- Ben Root
- Charles Ruan
- dahlbaek
- David Stansby
- DietmarSchwertberger
- Elizabeth Seiver
- Elliott Sales de Andrade
- Eric Firing
- Federico Ariza

- [fredrik-1](#)
- [hannah](#)
- [Importance of Being Ernest](#)
- [ImportanceOfBeingErnest](#)
- [Jens Hedegaard Nielsen](#)
- [Jody Klymak](#)
- [joelostblom](#)
- [Jouni K. Seppänen](#)
- [Michael Droettboom](#)
- [Nelle Varoquaux](#)
- [Paul Hobson](#)
- [Paul Ivanov](#)
- [Ryan May](#)
- [Thomas A Caswell](#)
- [Tim Hoffmann](#)

GitHub issues and pull requests:

Pull Requests (101):

- [PR #11835](#): Backport PR #11831 on branch v2.2.x
- [PR #11831](#): animation api documentation changes
- [PR #11834](#): Backport PR #11824 on branch v2.2.x
- [PR #11824](#): facecolor stale missing
- [PR #11827](#): FIX: correct macro logic in file_compat.h
- [PR #11817](#): Merge pull request #11533 from tacaswell/update_for_py37
- [PR #11533](#): MNT: updates for python 3.7
- [PR #11813](#): Backport axes grid work
- [PR #11808](#): updates error message to run pytest instead of tests.py standalone
- [PR #11812](#): Final backports
- [PR #11811](#): Backport: resolve conflict to get this into v2.2.3
- [PR #9993](#): Rewrite and greatly simplify qt_compat.py.
- [PR #11786](#): Fix collections import warnings for Python 3.7
- [PR #11733](#): MAINT: use collections.abc for 3.7
- [PR #11697](#): Backport PR #11694 on branch v2.2.x

- PR #11694: moving toolmanager initialization up before toolbar
- PR #11671: refresh the github stats
- PR #11669: Fix invalid escape sequences on branch v2.2.x
- PR #11638: Backport PR #11636 on branch v2.2.x
- PR #11634: Backport #11624
- PR #11636: Don't flush file object opened for reading
- PR #11624: Don't use Popen(..., shell=True).
- PR #11599: Backport PR #11559 on branch v2.2.x
- PR #11598: Backport PR #10915 on branch v2.2.x
- PR #11588: DOC: warn if user is using constrained layout and use subplots__adjust
- PR #11580: Use plt.subplots()
- PR #11574: Backport PR #11565 on branch v2.2.x
- PR #11565: Update docstring of pyplot.matshow()
- PR #11559: FIX: Toolbar disappears in TkAgg if window if resized
- PR #11567: Backport PR #11500 on branch v2.2.x
- PR #11500: Fix shims with PyQt5 5.11
- PR #11060: Fix inset_axes + doc
- PR #11535: Backport PR #11517 on branch v2.2.x
- PR #11517: Update legend loc default value in docstring
- PR #11489: Backport PR #11485 on branch v2.2.x
- PR #11485: Use null character instead of recasting NULL to char
- PR #11419: Backport PR #11409: plt.box_bug_fix
- PR #11379: Backport PR #11363 on branch v2.2.x
- PR #11409: plt.box_bug_fix
- PR #11407: Properly position markers in step plots.
- PR #11330: FIX: Don't let constrained_layout counter overflow
- PR #11363: Improve docstring of Axes.pcolorfast
- PR #11354: Backport PR #11344 on branch v2.2.x
- PR #11344: Improve docstring of Axes.pcolormesh
- PR #11336: Use raw string literals for docstrings with escapes
- PR #11334: Backport pull request #10858
- PR #11333: Backport #10806 to v2.2.x

- PR #11312: Replace `:ref:sphx_glr_...` by `:doc:/....`
- PR #11327: Backport PR #11317 on branch v2.2.x
- PR #11317: Improve docstring of `Axes.pcolor`
- PR #11262: Use `dummy_threading` if `threading` not available
- PR #11260: Backport PR #11189 on branch v2.2.x
- PR #11047: FIX: image respect norm limits w/ `None`
- PR #11173: Define `__repr__`, not `__str__` for transforms.
- PR #11189: FIX: remove recursive call to `str` in transform `repr`
- PR #11184: Manual backport #11035
- PR #11035: FIX: `constrained_layout` and repeated calls to `suptitle`
- PR #11171: FIX: `str` method for `SymmetricalLogTransform`
- PR #11162: Backport PR #11161 on branch v2.2.x
- PR #11161: DOC: small-doc-improvements1
- PR #11079: Resurrecting `axes_grid1` documentation
- PR #11105: When drawing markers, don't set the `GraphicsContext` alpha.
- PR #10881: FIX: make un-used ticks not be visible
- PR #10968: Improve docstring of `contour`
- PR #11120: Backport PR #10981 on branch v2.2.x
- PR #10981: Document change of label visibility on shared axes
- PR #11114: Backport PR #10792 on branch v2.2.x
- PR #10792: Fixed not being able to set vertical/horizontal alignments in polar graphs
- PR #11111: Pin sphinx!=1.7.3 on v2.2.x
- PR #11107: Pin sphinx to 1.7.2 to unbreak the doc build.
- PR #11090: Clean docstring of `CountourSet`
- PR #11084: Backport PR #10899: Update `cycler` docstrings and favor `kwarg` over two...
- PR #10899: Update `cycler` docstrings and favor `kwarg` over two-args form
- PR #11071: Add note about `hist2d` resetting axis limits
- PR #11067: Remove unnecessary shebang.
- PR #11054: Backport PR #11052 on branch v2.2.x
- PR #11052: Fix `CircleCI` build.
- PR #11023: Backport PR #11022 on branch v2.2.x
- PR #11022: MNT: remove `distutils.sysconfig` import from `toplevel` module

- PR #11011: Backport PR #11002 on branch v2.2.x
- PR #11002: Fix logic error in `ScalarMappable.to_rgba`
- PR #10952: Backport PR #10951 on branch v2.2.x
- PR #10979: Backport PR #10978 on branch v2.2.x
- PR #10978: Remove to-be-deprecated NumPy imports.
- PR #10951: fix wx rubberband: correctly ensure $x_0 \leq x_1$
- PR #10937: Backport PR #10935 on branch v2.2.x
- PR #10935: FIX: Postscript allow empty markers
- PR #10858: FIX: ioerror font cache, second try
- PR #10929: Fix `rctemplate` tests when user config file exists.
- PR #10927: Backport PR #10919 on branch v2.2.x
- PR #10919: Fix overflow when resizing path-to-string buffer.
- PR #10877: Fix invalid escape sequence in docstring.
- PR #10868: Backport PR #10867 on branch v2.2.x
- PR #10867: MNT: move bz2 back to function level
- PR #10860: Backport PR #10856 on branch v2.2.x
- PR #10856: Fix xkcd style garbage collection.
- PR #10853: Backport PR #10662 on branch v2.2.x
- PR #10662: Update docs on `Axes.set_prop_cycle`
- PR #10833: Propagate marker antialias setting to `GraphicsContext`.
- PR #10806: MNT: catch more illegal "
- PR #10595: Improve Figure docstrings

Issues (27):

- #11822: `matplotlib.pyplot` import gives error on MacOSX
- #11832: Marker disappears with `markerfacecolor='None'` when saving as eps file
- #11810: Figure gets collapsed after nth iteration using `constrained_layout`
- #11389: Incorrect processing of `plot_args` when using data argument
- #11635: "import matplotlib.pyplot" fails on NetBSD with version 2.2.2
- #6781: Toolbar disappears in TkAgg if window if resized
- #10890: Bug causes `to_rgba` to fail inside `cm.py`
- #9582: `axes_grid1` docs do not appear on `mpl.org` anymore.
- #11484: Matplotlib does not build on OpenBSD using clang

- [#11302](#): [DOC] Build of documentation fails
- [#11031](#): wrong x-position of marker with `drawstyle='steps-xxx'`
- [#11372](#): Symbols not showing in eps format
- [#10814](#): manually backport [#10806](#)
- [#10072](#): `imshow` doesn't properly display some images
- [#11163](#): `RecursionError` when calling `get_yaxis_transform()` on a Symlog-scaled axis.
- [#11156](#): Error in documentation of `pyplot.errorbar()`
- [#11104](#): `pyplot.plot` conflict: `markerfacecolor = 'none'` and `alpha`
- [#11126](#): Legend positioning behavior change from matplotlib {2.1.0 -> 2.2.2}
- [#10633](#): Type Error when saving jpg image (Pillow 3)
- [#10911](#): Get ticklabels back on shared axis
- [#5609](#): Segfaults with wxpython 3.0.2.0 backend
- [#11021](#): remove `import distutils.sysconfig` from `lib/matplotlib/__init__.py`
- [#10949](#): Qt5Agg blitting issue with matplotlib 2.2.2
- [#10966](#): pillow animation writer failing macOS tests
- [#10921](#): No markers in EPS if `fillstyle='none'`
- [#10889](#): memory error using `savefig` with `ylim` to create pdf of box plots
- [#10866](#): bz2 mistakenly made a hard requirement of matplotlib 2.2.2

LICENSE

Matplotlib only uses BSD compatible code, and its license is based on the [PSF](#) license. See the Open Source Initiative [licenses page](#) for details on individual licenses. Non-BSD compatible licenses (e.g., LGPL) are acceptable in matplotlib toolkits. For a discussion of the motivations behind the licencing choice, see Licenses.

7.1 Copyright Policy

John Hunter began matplotlib around 2003. Since shortly before his passing in 2012, Michael Droettboom has been the lead maintainer of matplotlib, but, as has always been the case, matplotlib is the work of many.

Prior to July of 2013, and the 1.3.0 release, the copyright of the source code was held by John Hunter. As of July 2013, and the 1.3.0 release, matplotlib has moved to a shared copyright model.

matplotlib uses a shared copyright model. Each contributor maintains copyright over their contributions to matplotlib. But, it is important to note that these contributions are typically only changes to the repositories. Thus, the matplotlib source code, in its entirety, is not the copyright of any single person or institution. Instead, it is the collective copyright of the entire matplotlib Development Team. If individual contributors want to maintain a record of what changes/contributions they have specific copyright on, they should indicate their copyright in the commit message of the change, when they commit the change to one of the matplotlib repositories.

The Matplotlib Development Team is the set of all contributors to the matplotlib project. A full list can be obtained from the git version control logs.

7.2 License agreement for matplotlib 2.2.4

1. This LICENSE AGREEMENT is between the Matplotlib Development Team ("MDT"), and the Individual or Organization ("Licensee") accessing and otherwise using matplotlib software in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, MDT hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use matplotlib 2.2.4 alone or in any derivative version, provided, however, that MDT's License Agreement and MDT's notice of

copyright, i.e., "Copyright (c) 2012-2013 Matplotlib Development Team; All Rights Reserved" are retained in matplotlib 2.2.4 alone or in any derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or incorporates matplotlib 2.2.4 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to matplotlib 2.2.4.

4. MDT is making matplotlib 2.2.4 available to Licensee on an "AS IS" basis. MDT MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, MDT MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF MATPLOTLIB 2.2.4 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. MDT SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF MATPLOTLIB 2.2.4 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING MATPLOTLIB 2.2.4, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between MDT and Licensee. This License Agreement does not grant permission to use MDT trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By copying, installing or otherwise using matplotlib 2.2.4, Licensee agrees to be bound by the terms and conditions of this License Agreement.

7.3 License agreement for matplotlib versions prior to 1.3.0

1. This LICENSE AGREEMENT is between John D. Hunter ("JDH"), and the Individual or Organization ("Licensee") accessing and otherwise using matplotlib software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, JDH hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use matplotlib 2.2.4 alone or in any derivative version, provided, however, that JDH's License Agreement and JDH's notice of copyright, i.e., "Copyright (c) 2002-2009 John D. Hunter; All Rights Reserved" are retained in matplotlib 2.2.4 alone or in any derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or incorporates matplotlib 2.2.4 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to matplotlib 2.2.4.

4. JDH is making matplotlib 2.2.4 available to Licensee on an "AS IS" basis. JDH MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE,

BUT NOT LIMITATION, JDH MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF MATPLOTLIB 2.2.4 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. JDH SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF MATPLOTLIB 2.2.4 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING MATPLOTLIB 2.2.4, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between JDH and Licensee. This License Agreement does not grant permission to use JDH trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By copying, installing or otherwise using matplotlib 2.2.4, Licensee agrees to be bound by the terms and conditions of this License Agreement.

CREDITS

Matplotlib was written by John D. Hunter, with contributions from an ever-increasing number of users and developers. The current co-lead developers are Michael Droettboom and Thomas A. Caswell; they are assisted by many [active](#) developers.

The following is a list of contributors extracted from the git revision control history of the project:

4over7, Aaron Boushley, Acanthostega, Adam Ginsburg, Adam Heck, Adam Ortiz, Adrian Price-Whelan, Adrien F. Vincent, Ahmet Bakan, Alan Du, Alejandro Dubrovsky, Alex C. Szatmary, Alex Loew, Alexander Taylor, Alexei Colin, Ali Mehdi, Alistair Muldal, Allan Haldane, Amit Aronovitch, Amy, AmyTeegarden, Andrea Bedini, Andreas Hilboll, Andreas Wallner, Andrew Dawson, Andrew Merrill, Andrew Straw, Andy Zhu, Anton Akhmerov, Antony Lee, Arie, Ariel Hernán Curiale, Arnaud Gardelein, Arpad Horvath, Aseem Bansal, Behram Mistree, Ben Cohen, Ben Gamari, Ben Keller, Ben Root, Benjamin Reedlunn, Binglin Chang, Bradley M. Froehle, Brandon Liu, Brett Cannon, Brett Graham, Brian Mattern, Brian McLaughlin, Bruno Beltran, CJ Carey, Cameron Bates, Cameron Davidson-Pilon, Carissa Brittain, Carl Michal, Carwyn Pelley, Casey Webster, Casper van der Wel, Charles Moad, Chris Beaumont, Chris G, Christian Brueffer, Christian Stadel-Schuldt, Christoph Dann, Christoph Gohlke, Christoph Hoffmann, Cimarron Mittelsteadt, Corey Farwell, Craig M, Craig Tenney, Damon McDougall, Dan Hickstein, Daniel Hyams, Daniel O'Connor, Dara Adib, Darren Dale, David Anderson, David Haberthür, David Huard, David Kaplan, David Kua, David Trémouilles, Dean Malmgren, Dmitry Lupyan, DonaldSeo, Dora Fraeman, Duncan Macleod, Edin Salkovic, Elena Glassman, Elias Pipping, Elliott Sales de Andrade, Emil Mikulic, Eric Dill, Eric Firing, Eric Ma, Eric O. LEBIGOT (EOL), Erik Bray, Eugen Beck, Eugene Yurtsev, Evan Davey, Ezra Peisach, Fabien Maussion, Fabio Zanini, Federico Ariza, Felipe, Fernando Perez, Filipe Fernandes, Florian Rhiem, Francesco Montesano, Francis Colas, François Magimel, Gaute Hope, Gellule Xg, Geoffroy Billotey, Gerald Storer, Giovanni, Graham Poulter, Gregory Ashton, Gregory R. Lee, Grégory Lielens, Guillaume Gay, Gustavo Braganca, Hans Dembinski, Hans Meine, Hans Moritz Günther, Hassan Kibirige, Holger Peters, Hubert Holin, Ian Thomas, Ignas Anikevicius (gns_ank), Ilia Kurenkov, Ioannis Filippidis, Ismo Toijala, J. Goutin, Jack Kelly, Jae-Joon Lee, Jaime Fernandez, Jake Vanderplas, James A. Bednar, James Pallister, James R. Evans, JamesMakela, Jan Schulz, Jan-Philip Gehrcke, Jan-willem De Bleser, Jarrod Millman, Jascha Ulrich, Jason Grout, Jason Liw Yan Chong, Jason Miller, JayP16, Jeff Lutgen, Jeff Whitaker, Jeffrey Bingham, Jens Hedegaard Nielsen, Jeremy Fix, Jeremy O'Donoghue, Jeremy Thurgood, Jessica B. Hamrick, Jim Radford, Jochen Voss, Jody Klymak, Joe Kington, Joel B. Mohler, John Hunter, Jonathan Waltman, Jorrit Wronski, Josef Heinen, Joseph Jon Booker, José Ricardo, Jouni K. Seppänen, Julian Mehne, Julian Taylor, JulianCienfuegos, Julien Lhermitte, Julien Schueller, Julien Woillez, Julien-Charles Lévesque, Kanwar245, Katy Huff, Ken McIvor, Kevin Chan, Kevin Davies, Kevin Keating, Kimmo Palin, Konrad Förstner, Konstantin Tretyakov,

Kristen M. Thyng, Lance Hepler, Larry Bradley, Leeonadoh, Lennart Fricke, Leo Singer, Levi Kilcher, Lion Krischer, Lodato Luciano, Lori J, Loïc Estève, Loïc Séguin-C, Magnus Nord, Majid al-Dosari, Maksym P, Manuel GOACOLOU, Manuel Metz, Marc Abramowitz, Marcos Duarte, Marek Rudnicki, Marianne Corvellec, Marin Gilles, Markus Roth, Markus Rothe, Martin Dengler, Martin Fitzpatrick, Martin Spacek, Martin Teichmann, Martin Thoma, Martin Ueding, Masud Rahman, Mathieu Duponchelle, Matt Giuca, Matt Klein, Matt Li, Matt Newville, Matt Shen, Matt Terry, Matthew Brett, Matthew Emmett, Matthias Bussonnier, Matthieu Caneill, Matěj Týč, Maximilian Albert, Maximilian Trescher, Mellissa Cross, Michael, Michael Droettboom, Michael Sarahan, Michael Welter, Michiel de Hoon, Michka Popoff, Mike Kaufman, Mikhail Korobov, MinRK, Minty Zhang, MirandaXM, Miriam Sierig, Muhammad Mehdi, Neil, Neil Crighton, Nelle Varoquaux, Niall Robinson, Nic Eggert, Nicholas Devenish, Nick Semenkovich, Nicolas P. Rougier, Nicolas Pinto, Nikita Kniazev, Niklas Koep, Nikolay Vyahhi, Norbert Nemec, OceanWolf, Oleg Selivanov, Olga Botvinnik, Oliver Willekens, Parfenov Sergey, Pascal Bugnion, Patrick Chen, Patrick Marsh, Paul, Paul Barret, Paul G, Paul Hobson, Paul Ivanov, Pauli Virtanen, Per Parker, Perry Greenfield, Pete Bachant, Peter Iannucci, Peter St. John, Peter Würtz, Phil Elson, Pierre Haessig, Pim Schellart, Piti Ongmongkolkul, Puneeth Chaganti, Ramiro Gómez, Randy Olson, Reinier Heeres, Remi Rampin, Richard Hattersley, Richard Trieu, Ricky, Robert Johansson, Robin Dunn, Rohan Walker, Roland Wirth, Russell Owen, RutgerK, Ryan Blomberg, Ryan D'Souza, Ryan Dale, Ryan May, Ryan Nelson, RyanPan, Salil Vanvari, Sameer D'Costa, Sandro Tosi, Scott Lasley, Scott Lawrence, Scott Stevenson, Sebastian Pinnau, Sebastian Raschka, Sergey Kholodilov, Sergey Kuposov, Silviu Tantos, Simon Cross, Simon Gibbons, Skelpdar, Skipper Seabold, Slav Basharov, Spencer McIntyre, Stanley, Simon, Stefan Lehmann, Stefan van der Walt, Stefano Rivera, Stephen Horst, Sterling Smith, Steve Chaplin, Steven Silvester, Stuart Mumford, Takafumi Arakaki, Takeshi Kanmae, Tamas Gal, Thomas A Caswell, Thomas Hisch, Thomas Kluyver, Thomas Lake, Thomas Robitaille, Thomas Spura, Till Stensitzki, Timo Vanwynsberghe, Tobias Hoppe, Tobias Megies, Todd Jennings, Todd Miller, Tomas Kazmar, Tony S Yu, Tor Colvin, Travis Oliphant, Trevor Bekolay, Ulrich Döbramysl, Umair Idris, Vadim Markovtsev, Valentin Haenel, Victor Zabalza, Viktor Kerkez, Vlad Seghete, Víctor Terrón, Víctor Zabalza, Wen Li, Wendell Smith, Werner F Bruhin, Wes Campaigne, Wieland Hoffmann, William Manley, Wouter Overmeire, Xiaowen Tang, Yann Tambouret, Yaron de Leeuw, Yu Feng, Yunfei Yang, Yuri D'Elia, Yuval Langer, Zac Hatfield-Dodds, Zach Pincus, Zair Mubashar, alex, anykraus, arokem, aseagram, aszilagy, bblay, bev-a-tron, blackw1ng, blah blah, burrbull, butterw, cammil, captainwhippet, chadawagner, chebee7i, danielballan, davidovitch, daydreamt, domspad, donald, drevicko, e-q, elpres, endolith, fardal, fiteja, fgb, fibersnet, frenchwr, fvgoto, gitj, gluap, goir, hugadams, insertroar, itziakos, jbbrokaw, juan.gonzalez, kcrisman, kelsiegr, khyox, kikocorreoso, kramer65, kshramt, lichri12, limtaesu, marky, masamson, mbyt, mcelrath, mdipierro, mrkrd, nickystringer, nwin, pkienzle, proffholzer, pupssman, rahiel, rhoef, rsnape, s9w, sdementen, sfroid, sohero, spiessbuerger, stahlous, switham, syngtron, torfbolt, u55, ugurthemaster, vbr, xbtsw, and xuanyuansen.

Some earlier contributors not included above are (with apologies to any we have missed):

Charles Twardy, Gary Ruben, John Gill, David Moore, Paul Barrett, Jared Wahlstrand, Jim Benson, Paul McGuire, Andrew Dalke, Nadia Dencheva, Baptiste Carvello, Sigve Tjoraand, Ted Drain, James Amundson, Daishi Harada, Nicolas Young, Paul Kienzle, John Porter, and Jonathon Taylor.

We also thank all who have reported bugs, commented on proposed changes, or otherwise contributed to Matplotlib's development and usefulness.

Part II

The Matplotlib FAQ

INSTALLATION

Contents

- Installation
 - Report a compilation problem
 - matplotlib compiled fine, but nothing shows up when I use it
 - How to completely remove Matplotlib
 - Linux Notes
 - OSX Notes
 - * Which python for OSX?
 - * Installing OSX binary wheels
 - Python.org Python
 - Macports Python
 - Homebrew Python
 - pip problems
 - * Checking your installation
 - Windows Notes
 - Install from source

9.1 Report a compilation problem

See Getting help.

9.2 matplotlib compiled fine, but nothing shows up when I use it

The first thing to try is a clean install and see if that helps. If not, the best way to test your install is by running a script, rather than working interactively from a python shell or an integrated development environment such as IDLE which add additional complexities. Open up a UNIX shell or a DOS command prompt and run, for example:

```
python -c "from pylab import *; plot(); show()" --verbose-helpful
```

This will give you additional information about which backends matplotlib is loading, version information, and more. At this point you might want to make sure you understand matplotlib's [configuration](#) process, governed by the matplotlibrc configuration file which contains instructions within and the concept of the matplotlib backend.

If you are still having trouble, see [Getting help](#).

9.3 How to completely remove Matplotlib

Occasionally, problems with Matplotlib can be solved with a clean installation of the package. In order to fully remove an installed Matplotlib:

1. Delete the caches from your Matplotlib configuration directory.
2. Delete any Matplotlib directories or eggs from your installation directory.

9.4 Linux Notes

To install Matplotlib at the system-level, we recommend that you use your distribution's package manager. This will guarantee that Matplotlib's dependencies will be installed as well.

If, for some reason, you cannot use the package manager, you may use the wheels available on PyPI:

```
python -mpip install matplotlib
```

or build Matplotlib from source.

9.5 OSX Notes

9.5.1 Which python for OSX?

Apple ships OSX with its own Python, in `/usr/bin/python`, and its own copy of Matplotlib. Unfortunately, the way Apple currently installs its own copies of NumPy, Scipy and Matplotlib means that these packages are difficult to upgrade (see [system python packages](#)). For that reason we strongly suggest that you install a fresh version of Python and use that as the basis for installing

libraries such as NumPy and Matplotlib. One convenient way to install matplotlib with other useful Python software is to use one of the excellent Python scientific software collections that are now available:

- [Anaconda](#) from [Continuum Analytics](#)
- [Canopy](#) from [Enthought](#)

These collections include Python itself and a wide range of libraries; if you need a library that is not available from the collection, you can install it yourself using standard methods such as `pip`. Continuum and Enthought offer their own installation support for these collections; see the [Ananconda](#) and [Canopy](#) web pages for more information.

Other options for a fresh Python install are the standard installer from [python.org](#), or installing Python using a general OSX package management system such as [homebrew](#) or [macports](#). Power users on OSX will likely want one of homebrew or macports on their system to install open source software packages, but it is perfectly possible to use these systems with another source for your Python binary, such as Anaconda, Canopy or Python.org Python.

9.5.2 Installing OSX binary wheels

If you are using recent Python from <https://www.python.org>, Macports or Homebrew, then you can use the standard `pip` installer to install Matplotlib binaries in the form of wheels.

Python.org Python

Install `pip` following the [standard pip install instructions](#). For the impatient, open a new Terminal.app window and:

```
curl -O https://bootstrap.pypa.io/get-pip.py
```

Then (Python 2):

```
python get-pip.py
```

or (Python 3):

```
python3 get-pip.py
```

You can now install matplotlib and all its dependencies with

```
python -mpip install matplotlib
```

or

```
python3 -mpip install matplotlib
```

Macports Python

For Python 2:

```
sudo port install py27-pip
sudo python2 -mpip install matplotlib
```

For Python 3:

```
sudo port install py36-pip
sudo python3.6 -mpip install matplotlib
```

Homebrew Python

For Python 2:

```
python2 -mpip install matplotlib
```

For Python 3:

```
python3 -mpip install matplotlib
```

You might also want to install IPython or the Jupyter notebook (`pythonX -mpip install ipython`, `pythonX -mpip install notebook`, where `pythonX` is set as above).

pip problems

If you get errors with pip trying to run a compiler like gcc or clang, then the first thing to try is to [install xcode](#) and retry the install. If that does not work, then check [Getting help](#).

9.5.3 Checking your installation

The new version of Matplotlib should now be on your Python "path". Check this with one of these commands at the Terminal.app command line:

```
python2 -c 'import matplotlib; print matplotlib.__version__, matplotlib.__file__'
```

(Python 2) or:

```
python3 -c 'import matplotlib; print(matplotlib.__version__, matplotlib.__file__)'
```

(Python 3). You should see something like this:

```
2.1.0 /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/
↳matplotlib/___init___.pyc
```

where 2.1.0 is the Matplotlib version you just installed, and the path following depends on whether you are using Python.org Python, Homebrew or Macports. If you see another version, or you get an error like this:

```
Traceback (most recent call last):  
  File "<string>", line 1, in <module>  
ImportError: No module named matplotlib
```

then check that the Python binary is the one you expected by doing one of these commands in Terminal.app:

```
which python2
```

or:

```
which python3
```

If you get the result `/usr/bin/python2.7`, then you are getting the Python installed with OSX, which is probably not what you want. Try closing and restarting Terminal.app before running the check again. If that doesn't fix the problem, depending on which Python you wanted to use, consider reinstalling Python.org Python, or check your homebrew or macports setup. Remember that the disk image installer only works for Python.org Python, and will not get picked up by other Pythons. If all these fail, please let us know.

9.6 Windows Notes

See [Windows](#).

9.7 Install from source

Clone the main source using one of:

```
git clone git@github.com:matplotlib/matplotlib.git
```

or:

```
git clone git://github.com/matplotlib/matplotlib.git
```

and build and install as usual with:

```
cd matplotlib  
python -mpip install .
```

Note: If you are on Debian/Ubuntu, you can get all the dependencies required to build Matplotlib with:

```
sudo apt-get build-dep python-matplotlib
```

If you are on Fedora/RedHat, you can get all the dependencies required to build matplotlib by first installing yum-builddep and then running:

```
su -c 'yum-builddep python-matplotlib'
```

This does not build Matplotlib, but it does get all of the build dependencies, which will make building from source easier.

If you want to be able to follow the development branch as it changes just replace the last step with:

```
python -mpip install -e .
```

This creates links and installs the command line script in the appropriate places.

Note: OSX users please see the [Building on macOS](#) guide.

Windows users please see the [Building on Windows](#) guide.

Then, if you want to update your matplotlib at any time, just do:

```
git pull
```

When you run git pull, if the output shows that only Python files have been updated, you are all set. If C files have changed, you need to run pip install -e . again to compile them.

There is more information on using git in the developer docs.

Contents

- How-To
 - Plotting: howto
 - * Plot `numpy.datetime64` values
 - * Find all objects in a figure of a certain type
 - * How to prevent ticklabels from having an offset
 - * Save transparent figures
 - * Save multiple plots to one pdf file
 - * Move the edge of an axes to make room for tick labels
 - * Automatically make room for tick labels
 - * Configure the tick widths
 - * Align my ylabels across multiple subplots
 - * Skip dates where there is no data
 - * Control the depth of plot elements
 - * Make the aspect ratio for plots equal
 - * Multiple y-axis scales
 - * Generate images without having a window appear
 - * Use `show()`
 - * Interpreting box plots and violin plots
 - Contributing: howto
 - * Request a new feature
 - * Reporting a bug or submitting a patch
 - * Contribute to Matplotlib documentation

- Matplotlib in a web application server
 - * Matplotlib with apache
 - * Matplotlib with django
 - * Matplotlib with zope
 - * Clickable images for HTML
- Search examples
- Cite Matplotlib

10.1 Plotting: howto

10.1.1 Plot `numpy.datetime64` values

As of Matplotlib 2.2, `numpy.datetime64` objects are handled the same way as `datetime.datetime` objects.

If you prefer the pandas converters and locators, you can register their converter with the `matplotlib.units` module:

```
from pandas.tseries import converter as pdtc
pdtc.register()
```

If you only want to use the `pandas` converter for `datetime64` values

```
from pandas.tseries import converter as pdtc
import matplotlib.units as munits
import numpy as np

munits.registry[np.datetime64] = pdtc.DatetimeConverter()
```

10.1.2 Find all objects in a figure of a certain type

Every Matplotlib artist (see [Artist tutorial](#)) has a method called `findobj()` that can be used to recursively search the artist for any artists it may contain that meet some criteria (e.g., match all `Line2D` instances or match some arbitrary filter function). For example, the following snippet finds every object in the figure which has a `set_color` property and makes the object blue:

```
def myfunc(x):
    return hasattr(x, 'set_color')

for o in fig.findobj(myfunc):
    o.set_color('blue')
```

You can also filter on class instances:

```
import matplotlib.text as text
for o in fig.findobj(text.Text):
    o.set_fontstyle('italic')
```

10.1.3 How to prevent ticklabels from having an offset

The default formatter will use an offset to reduce the length of the ticklabels. To turn this feature off on a per-axis basis:

```
ax.get_xaxis().get_major_formatter().set_useOffset(False)
```

set the rcParam `axes.formatter.useoffset`, or use a different formatter. See `ticker` for details.

10.1.4 Save transparent figures

The `savefig()` command has a keyword argument `transparent` which, if `'True'`, will make the figure and axes backgrounds transparent when saving, but will not affect the displayed image on the screen.

If you need finer grained control, e.g., you do not want full transparency or you want to affect the screen displayed version as well, you can set the alpha properties directly. The figure has a `Rectangle` instance called `patch` and the axes has a `Rectangle` instance called `patch`. You can set any property on them directly (`facecolor`, `edgecolor`, `linewidth`, `linestyle`, `alpha`). e.g.:

```
fig = plt.figure()
fig.patch.set_alpha(0.5)
ax = fig.add_subplot(111)
ax.patch.set_alpha(0.5)
```

If you need all the figure elements to be transparent, there is currently no global alpha setting, but you can set the alpha channel on individual elements, e.g.:

```
ax.plot(x, y, alpha=0.5)
ax.set_xlabel('volts', alpha=0.5)
```

10.1.5 Save multiple plots to one pdf file

Many image file formats can only have one image per file, but some formats support multi-page files. Currently only the pdf backend has support for this. To make a multi-page pdf file, first initialize the file:

```
from matplotlib.backends.backend_pdf import PdfPages
pp = PdfPages('multipage.pdf')
```

You can give the PdfPages object to `savefig()`, but you have to specify the format:

```
plt.savefig(pp, format='pdf')
```

An easier way is to call `PdfPages.savefig()`:

```
pp.savefig()
```

Finally, the multipage pdf object has to be closed:

```
pp.close()
```

10.1.6 Move the edge of an axes to make room for tick labels

For subplots, you can control the default spacing on the left, right, bottom, and top as well as the horizontal and vertical spacing between multiple rows and columns using the `matplotlib.figure.Figure.subplots_adjust()` method (in `pyplot` it is `subplots_adjust()`). For example, to move the bottom of the subplots up to make room for some rotated x tick labels:

```
fig = plt.figure()
fig.subplots_adjust(bottom=0.2)
ax = fig.add_subplot(111)
```

You can control the defaults for these parameters in your `matplotlibrc` file; see [Customizing Matplotlib with style sheets and rcParams](#). For example, to make the above setting permanent, you would set:

```
figure.subplot.bottom : 0.2  # the bottom of the subplots of the figure
```

The other parameters you can configure are, with their defaults

`left = 0.125` the left side of the subplots of the figure

`right = 0.9` the right side of the subplots of the figure

`bottom = 0.1` the bottom of the subplots of the figure

`top = 0.9` the top of the subplots of the figure

`wspace = 0.2` the amount of width reserved for space between subplots, expressed as a fraction of the average axis width

`hspace = 0.2` the amount of height reserved for space between subplots, expressed as a fraction of the average axis height

If you want additional control, you can create an `Axes` using the `axes()` command (or equivalently the figure `add_axes()` method), which allows you to specify the location explicitly:

```
ax = fig.add_axes([left, bottom, width, height])
```

where all values are in fractional (0 to 1) coordinates. See [/gallery/subplots_axes_and_figures/axes_demo](#) for an example of placing axes manually.

10.1.7 Automatically make room for tick labels

Note: This is now easier to handle than ever before. Calling `tight_layout()` can fix many common layout issues. See the [Tight Layout guide](#).

The information below is kept here in case it is useful for other purposes.

In most use cases, it is enough to simply change the subplots adjust parameters as described in [Move the edge of an axes to make room for tick labels](#). But in some cases, you don't know ahead of time what your tick labels will be, or how large they will be (data and labels outside your control may be being fed into your graphing application), and you may need to automatically adjust your subplot parameters based on the size of the tick labels. Any Text instance can report its extent in window coordinates (a negative x coordinate is outside the window), but there is a rub.

The `RendererBase` instance, which is used to calculate the text size, is not known until the figure is drawn (`draw()`). After the window is drawn and the text instance knows its renderer, you can call `get_window_extent()`. One way to solve this chicken and egg problem is to wait until the figure is draw by connecting (`mpl_connect()`) to the "on_draw" signal (`DrawEvent`) and get the window extent there, and then do something with it, e.g., move the left of the canvas over; see [Event handling and picking](#).

Here is an example that gets a bounding box in relative figure coordinates (0..1) of each of the labels and uses it to move the left of the subplots over so that the tick labels fit in the figure:

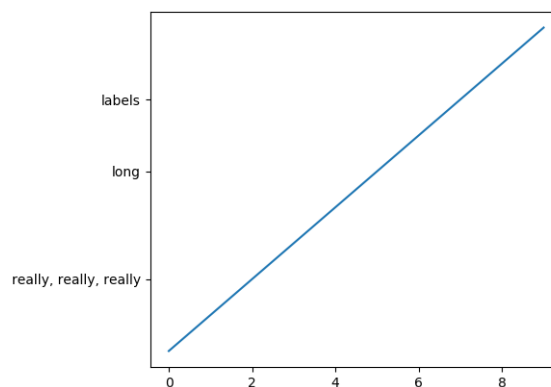


Fig. 1: Auto Subplots Adjust

10.1.8 Configure the tick widths

Wherever possible, it is recommended to use the `tick_params()` or `set_tick_params()` methods to modify tick properties:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax.plot(range(10))

ax.tick_params(width=10)

plt.show()
```

For more control of tick properties that are not provided by the above methods, it is important to know that in Matplotlib, the ticks are markers. All Line2D objects support a line (solid, dashed, etc) and a marker (circle, square, tick). The tick width is controlled by the "markeredgewidth" property, so the above effect can also be achieved by:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax.plot(range(10))

for line in ax.get_xticklines() + ax.get_yticklines():
    line.set_mmarkeredgewidth(10)

plt.show()
```

The other properties that control the tick marker, and all markers, are markerfacecolor, markeredgewidth, markeredgewidth, markersize. For more information on configuring ticks, see [Axis containers](#) and [Tick containers](#).

10.1.9 Align my ylabels across multiple subplots

If you have multiple subplots over one another, and the y data have different scales, you can often get ylabels that do not align vertically across the multiple subplots, which can be unattractive. By default, Matplotlib positions the x location of the ylabel so that it does not overlap any of the y ticks. You can override this default behavior by specifying the coordinates of the label. The example below shows the default behavior in the left subplots, and the manual setting in the right subplots.

10.1.10 Skip dates where there is no data

When plotting time series, e.g., financial time series, one often wants to leave out days on which there is no data, e.g., weekends. By passing in dates on the x-axis, you get large horizontal gaps on periods when there is not data. The solution is to pass in some proxy x-data, e.g., evenly sampled indices, and then use a custom formatter to format these as dates. The example below shows how to use an 'index formatter' to achieve the desired plot:

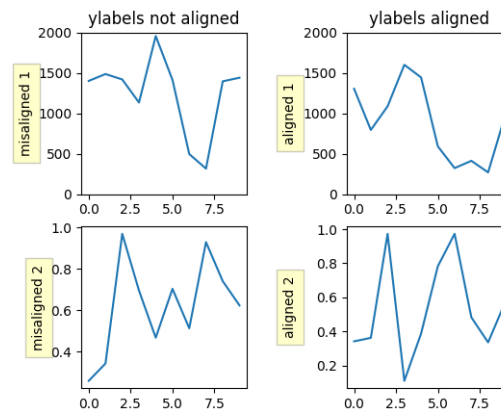


Fig. 2: Align Ylabels

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib.ticker as ticker

r = mlab.csv2rec('../data/aapl.csv')
r.sort()
r = r[-30:] # get the last 30 days

N = len(r)
ind = np.arange(N) # the evenly spaced plot indices

def format_date(x, pos=None):
    thisind = np.clip(int(x+0.5), 0, N-1)
    return r.date[thisind].strftime('%Y-%m-%d')

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(ind, r.adj_close, 'o-')
ax.xaxis.set_major_formatter(ticker.FuncFormatter(format_date))
fig.autofmt_xdate()

plt.show()
```

10.1.11 Control the depth of plot elements

Within an axes, the order that the various lines, markers, text, collections, etc appear is determined by the `set_zorder()` property. The default order is patches, lines, text, with collections of lines and collections of patches appearing at the same level as regular lines and patches, respectively:

```
line, = ax.plot(x, y, zorder=10)
```

You can also use the Axes property `set_axisbelow()` to control whether the grid lines are placed above or below your other plot elements.

10.1.12 Make the aspect ratio for plots equal

The Axes property `set_aspect()` controls the aspect ratio of the axes. You can set it to be 'auto', 'equal', or some ratio which controls the ratio:

```
ax = fig.add_subplot(111, aspect='equal')
```

10.1.13 Multiple y-axis scales

A frequent request is to have two scales for the left and right y-axis, which is possible using `twinx()` (more than two scales are not currently supported, though it is on the wish list). This works pretty well, though there are some quirks when you are trying to interactively pan and zoom, because both scales do not get the signals.

The approach uses `twinx()` (and its sister `twiny()`) to use 2 different axes, turning the axes rectangular frame off on the 2nd axes to keep it from obscuring the first, and manually setting the tick locs and labels as desired. You can use separate `matplotlib.ticker` formatters and locators as desired because the two axes are independent.

10.1.14 Generate images without having a window appear

The easiest way to do this is use a non-interactive backend (see [What is a backend?](#)) such as Agg (for PNGs), PDF, SVG or PS. In your figure-generating script, just call the `matplotlib.use()` directive before importing `pylab` or `pyplot`:

```
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
plt.plot([1,2,3])
plt.savefig('myfig')
```

See also:

Matplotlib in a web application server for information about running matplotlib inside of a web application.

10.1.15 Use `show()`

When you want to view your plots on your display, the user interface backend will need to start the GUI mainloop. This is what `show()` does. It tells Matplotlib to raise all of the figure windows

created so far and start the mainloop. Because this mainloop is blocking by default (i.e., script execution is paused), you should only call this once per script, at the end. Script execution is resumed after the last window is closed. Therefore, if you are using Matplotlib to generate only images and do not want a user interface window, you do not need to call `show` (see [Generate images without having a window appear](#) and [What is a backend?](#)).

Note: Because closing a figure window invokes the destruction of its plotting elements, you should call `savefig()` before calling `show` if you wish to save the figure as well as view it.

New in version v1.0.0: `show` now starts the GUI mainloop only if it isn't already running. Therefore, multiple calls to `show` are now allowed.

Having `show` block further execution of the script or the python interpreter depends on whether Matplotlib is set for interactive mode or not. In non-interactive mode (the default setting), execution is paused until the last figure window is closed. In interactive mode, the execution is not paused, which allows you to create additional figures (but the script won't finish until the last figure window is closed).

Note: Support for interactive/non-interactive mode depends upon the backend. Until version 1.0.0 (and subsequent fixes for 1.0.1), the behavior of the interactive mode was not consistent across backends. As of v1.0.1, only the macosx backend differs from other backends because it does not support non-interactive mode.

Because it is expensive to draw, you typically will not want Matplotlib to redraw a figure many times in a script such as the following:

```
plot([1,2,3])      # draw here ?
xlabel('time')     # and here ?
ylabel('volts')    # and here ?
title('a simple plot') # and here ?
show()
```

However, it is possible to force Matplotlib to draw after every command, which might be what you want when working interactively at the python console (see [Using matplotlib in a python shell](#)), but in a script you want to defer all drawing until the call to `show`. This is especially important for complex figures that take some time to draw. `show()` is designed to tell Matplotlib that you're all done issuing commands and you want to draw the figure now.

Note: `show()` should typically only be called at most once per script and it should be the last line of your script. At that point, the GUI takes control of the interpreter. If you want to force a figure draw, use `draw()` instead.

Many users are frustrated by `show` because they want it to be a blocking call that raises the figure, pauses the script until they close the figure, and then allow the script to continue running until the next figure is created and the next `show` is made. Something like this:

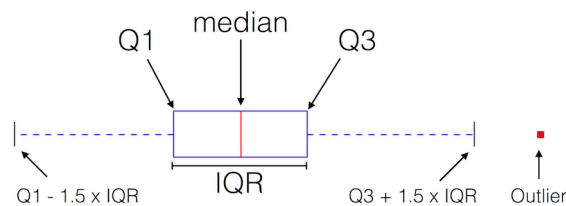

```
# WARNING : illustrating how NOT to use show
for i in range(10):
    # make figure i
    show()
```

This is not what `show` does and unfortunately, because doing blocking calls across user interfaces can be tricky, is currently unsupported, though we have made significant progress towards supporting blocking events.

New in version v1.0.0: As noted earlier, this restriction has been relaxed to allow multiple calls to `show`. In most backends, you can now expect to be able to create new figures and raise them in a subsequent call to `show` after closing the figures from a previous call to `show`.

10.1.16 Interpreting box plots and violin plots

Tukey's **box plots** (Robert McGill, John W. Tukey and Wayne A. Larsen: "The American Statistician" Vol. 32, No. 1, Feb., 1978, pp. 12-16) are statistical plots that provide useful information about the data distribution such as skewness. However, bar plots with error bars are still the common standard in most scientific literature, and thus, the interpretation of box plots can be challenging for the unfamiliar reader. The figure below illustrates the different visual features of a box plot.



Q1: *Quartile 1*, or median of the *left* data subset after dividing the original data set into 2 subsets via the median (25% of the data points fall below this threshold)

Q3: *Quartile 3*, median of the *right* data subset (75% of the data points fall below this threshold)

IQR: *Interquartile-range*, $Q3 - Q1$

Outliers: Data points are considered to be outliers if
 $value < Q1 - 1.5 \times IQR$ or
 $value > Q3 + 1.5 \times IQR$



Sebastian Raschka, 2016
 This work is licensed under a Creative Commons Attribution 4.0 International License

Violin plots are closely related to box plots but add useful information such as the distribution of the sample data (density trace). Violin plots were added in Matplotlib 1.4.

10.2 Contributing: howto

10.2.1 Request a new feature

Is there a feature you wish Matplotlib had? Then ask! The best way to get started is to email the developer [mailing list](#) for discussion. This is an open source project developed primarily in the contributors free time, so there is no guarantee that your feature will be added. The best way to get the feature you need added is to contribute it your self.

10.2.2 Reporting a bug or submitting a patch

The development of Matplotlib is organized through [github](#). If you would like to report a bug or submit a patch please use that interface.

To report a bug [create an issue](#) on github (this requires having a github account). Please include a [Short, Self Contained, Correct \(Compilable\), Example](#) demonstrating what the bug is. Including a clear, easy to test example makes it easy for the developers to evaluate the bug. Expect that the bug reports will be a conversation. If you do not want to register with github, please email bug reports to the [mailing list](#).

The easiest way to submit patches to Matplotlib is through pull requests on github. Please see the [The Matplotlib Developers' Guide](#) for the details.

10.2.3 Contribute to Matplotlib documentation

Matplotlib is a big library, which is used in many ways, and the documentation has only scratched the surface of everything it can do. So far, the place most people have learned all these features are through studying the examples ([Search examples](#)), which is a recommended and great way to learn, but it would be nice to have more official narrative documentation guiding people through all the dark corners. This is where you come in.

There is a good chance you know more about Matplotlib usage in some areas, the stuff you do every day, than many of the core developers who wrote most of the documentation. Just pulled your hair out compiling Matplotlib for windows? Write a FAQ or a section for the Installation page. Are you a digital signal processing wizard? Write a tutorial on the signal analysis plotting functions like `xcorr()`, `psd()` and `spectrogram()`. Do you use Matplotlib with [django](#) or other popular web application servers? Write a FAQ or tutorial and we'll find a place for it in the [User's Guide](#). Bundle Matplotlib in a [py2exe](#) app? ... I think you get the idea.

Matplotlib is documented using the [sphinx](#) extensions to restructured text (ReST). sphinx is an extensible python framework for documentation projects which generates HTML and PDF, and is pretty easy to write; you can see the source for this document or any page on this site by clicking on the Show Source link at the end of the page in the sidebar.

The sphinx website is a good resource for learning sphinx, but we have put together a cheat-sheet at [Writing documentation](#) which shows you how to get started, and outlines the Matplotlib conventions and extensions, e.g., for including plots directly from external code in your documents.

Once your documentation contributions are working (and hopefully tested by actually building the docs) you can submit them as a patch against git. See [Install git and Reporting a bug or submitting a patch](#). Looking for something to do? Search for [TODO](#) or look at the open issues on [github](#).

10.3 Matplotlib in a web application server

Many users report initial problems trying to use matplotlib in web application servers, because by default Matplotlib ships configured to work with a graphical user interface which may require an X11 connection. Since many barebones application servers do not have X11 enabled, you may get errors if you don't configure Matplotlib for use in these environments. Most importantly, you need to decide what kinds of images you want to generate (PNG, PDF, SVG) and configure the appropriate default backend. For 99% of users, this will be the Agg backend, which uses the C++ [antigrain](#) rendering engine to make nice PNGs. The Agg backend is also configured to recognize requests to generate other output formats (PDF, PS, EPS, SVG). The easiest way to configure Matplotlib to use Agg is to call:

```
# do this before importing pylab or pyplot
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
```

For more on configuring your backend, see [What is a backend?](#).

Alternatively, you can avoid pylab/pyplot altogether, which will give you a little more control, by calling the API directly as shown in [/gallery/api/agg_oo_sgskip](#).

You can either generate hardcopy on the filesystem by calling `savefig`:

```
# do this before importing pylab or pyplot
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot([1,2,3])
fig.savefig('test.png')
```

or by saving to a file handle:

```
import sys
fig.savefig(sys.stdout)
```

Here is an example using [Pillow](#). First, the figure is saved to a BytesIO object which is then fed to Pillow for further processing:

```
from io import BytesIO
from PIL import Image
```

(continues on next page)

(continued from previous page)

```
imgdata = BytesIO()
fig.savefig(imgdata, format='png')
imgdata.seek(0) # rewind the data
im = Image.open(imgdata)
```

10.3.1 Matplotlib with apache

TODO; see [Contribute to Matplotlib documentation](#).

10.3.2 Matplotlib with django

TODO; see [Contribute to Matplotlib documentation](#).

10.3.3 Matplotlib with zope

TODO; see [Contribute to Matplotlib documentation](#).

10.3.4 Clickable images for HTML

Andrew Dalke of [Dalke Scientific](#) has written a nice [article](#) on how to make html click maps with Matplotlib agg PNGs. We would also like to add this functionality to SVG. If you are interested in contributing to these efforts that would be great.

10.4 Search examples

The nearly 300 code examples-index included with the Matplotlib source distribution are full-text searchable from the search page, but sometimes when you search, you get a lot of results from the The Matplotlib API or other documentation that you may not be interested in if you just want to find a complete, free-standing, working piece of example code. To facilitate example searches, we have tagged every code example page with the keyword codex for code example which shouldn't appear anywhere else on this site except in the FAQ. So if you want to search for an example that uses an ellipse, search for codex ellipse.

10.5 Cite Matplotlib

If you want to refer to Matplotlib in a publication, you can use "Matplotlib: A 2D Graphics Environment" by J. D. Hunter In Computing in Science & Engineering, Vol. 9, No. 3. (2007), pp. 90-95 (see [this reference page](#)):

```
@article{Hunter:2007,
  Address = {10662 LOS VAQUEROS CIRCLE, PO BOX 3014, LOS ALAMITOS, CA,
  ↪90720-1314 USA},
  Author = {Hunter, John D.},
  Date-Added = {2010-09-23 12:22:10 -0700},
  Date-Modified = {2010-09-23 12:22:10 -0700},
  Isi = {000245668100019},
  Isi-Recid = {155389429},
  Journal = {Computing In Science \& Engineering},
  Month = {May-Jun},
  Number = {3},
  Pages = {90--95},
  Publisher = {IEEE COMPUTER SOC},
  Times-Cited = {21},
  Title = {Matplotlib: A 2D graphics environment},
  Type = {Editorial Material},
  Volume = {9},
  Year = {2007},
  Abstract = {Matplotlib is a 2D graphics package used for Python for application
    development, interactive scripting, and publication-quality image
    generation across user interfaces and operating systems.},
  BdsK-Url-1 = {http://gateway.isiknowledge.com/gateway/Gateway.cgi?GWVersion=2&
  ↪SrcAuth=Alerting&SrcApp=Alerting&DestApp=WOS&DestLinkType=FullRecord;
  ↪KeyUT=000245668100019}}
```

TROUBLESHOOTING

Contents

- Troubleshooting
 - Obtaining Matplotlib version
 - matplotlib install location
 - matplotlib configuration and cache directory locations
 - Getting help
 - Problems with recent git versions

11.1 Obtaining Matplotlib version

To find out your Matplotlib version number, import it and print the `__version__` attribute:

```
>>> import matplotlib
>>> matplotlib.__version__
'0.98.0'
```

11.2 matplotlib install location

You can find what directory Matplotlib is installed in by importing it and printing the `__file__` attribute:

```
>>> import matplotlib
>>> matplotlib.__file__
'/home/jdhunter/dev/lib64/python2.5/site-packages/matplotlib/__init__.pyc'
```

11.3 matplotlib configuration and cache directory locations

Each user has a Matplotlib configuration directory which may contain a `matplotlibrc` file. To locate your matplotlib/ configuration directory, use `matplotlib.get_configdir()`:

```
>>> import matplotlib as mpl
>>> mpl.get_configdir()
'/home/darren/.config/matplotlib'
```

On unix-like systems, this directory is generally located in your HOME directory under the `.config/` directory.

In addition, users have a cache directory. On unix-like systems, this is separate from the configuration directory by default. To locate your `.cache/` directory, use `matplotlib.get_cachedir()`:

```
>>> import matplotlib as mpl
>>> mpl.get_cachedir()
'/home/darren/.cache/matplotlib'
```

On windows, both the config directory and the cache directory are the same and are in your Documents and Settings or Users directory by default:

```
>>> import matplotlib as mpl
>>> mpl.get_configdir()
'C:\\Documents and Settings\\jdhunter\\.matplotlib'
>>> mpl.get_cachedir()
'C:\\Documents and Settings\\jdhunter\\.matplotlib'
```

If you would like to use a different configuration directory, you can do so by specifying the location in your `MPLCONFIGDIR` environment variable – see Setting environment variables in Linux and OS-X. Note that `MPLCONFIGDIR` sets the location of both the configuration directory and the cache directory.

11.4 Getting help

There are a number of good resources for getting help with Matplotlib. There is a good chance your question has already been asked:

- The [mailing list archive](#).
- [Github issues](#).
- Stackoverflow questions tagged [matplotlib](#).

If you are unable to find an answer to your question through search, please provide the following information in your e-mail to the [mailing list](#):

- Your operating system (Linux/UNIX users: post the output of `uname -a`).
- Matplotlib version:

```
python -c "import matplotlib; print matplotlib.__version__"
```

- Where you obtained Matplotlib (e.g., your Linux distribution's packages, Github, PyPi, or Anaconda or Enthought Canopy).
- Any customizations to your matplotlibrc file (see [Customizing Matplotlib with style sheets and rcParams](#)).
- If the problem is reproducible, please try to provide a minimal, standalone Python script that demonstrates the problem. This is the critical step. If you can't post a piece of code that we can run and reproduce your error, the chances of getting help are significantly diminished. Very often, the mere act of trying to minimize your code to the smallest bit that produces the error will help you find a bug in your code that is causing the problem.
- You can get helpful debugging output from Matplotlib by using the [logging](#) library in your code and posting the verbose output to the lists. For a command-line version of this, try:

```
python -c "from logging import *; basicConfig(level=DEBUG); from pylab import *; plot();
↪ show()"
```

If you want to put the debugging hooks in your own code, then the most simple way to do so is to insert the following before any calls to import matplotlib:

```
import logging
logging.basicConfig(level=logging.DEBUG)

import matplotlib.pyplot as plt
```

Note that if you want to use [logging](#) in your own code, but do not want verbose Matplotlib output, you can set the logging level for Matplotlib independently:

```
import logging
# set DEBUG for everything
logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger('matplotlib')
# set WARNING for Matplotlib
logger.setLevel(logging.WARNING)
```

The [logging](#) module is very flexible, and can be a valuable tool in chasing down errors.

If you compiled Matplotlib yourself, please also provide:

- any changes you have made to setup.py or setupext.py.
- the output of:

```
rm -rf build
python setup.py build
```

The beginning of the build output contains lots of details about your platform that are useful for the Matplotlib developers to diagnose your problem.

- your compiler version – e.g., `gcc --version`.

Including this information in your first e-mail to the mailing list will save a lot of time.

You will likely get a faster response writing to the mailing list than filing a bug in the bug tracker. Most developers check the bug tracker only periodically. If your problem has been determined to be a bug and can not be quickly solved, you may be asked to file a bug in the tracker so the issue doesn't get lost.

11.5 Problems with recent git versions

First make sure you have a clean build and install (see [How to completely remove Matplotlib](#)), get the latest git update, install it and run a simple test script in debug mode:

```
rm -rf /path/to/site-packages/matplotlib*
git clean -xdf
git pull
python -mpip install -v . > build.out
python examples/pylab_examples/simple_plot.py --verbose-debug > run.out
```

and post `build.out` and `run.out` to the [matplotlib-devel](#) mailing list (please do not post git problems to the [users list](#)).

Of course, you will want to clearly describe your problem, what you are expecting and what you are getting, but often a clean build and install will help. See also [Getting help](#).

ENVIRONMENT VARIABLES

Contents

- Environment Variables
 - Setting environment variables in Linux and OS-X
 - * BASH/KSH
 - * CSH/TCSH
 - Setting environment variables in windows

HOME

The user's home directory. On linux, `~` is shorthand for HOME.

PATH

The list of directories searched to find executable programs

PYTHONPATH

The list of directories that is added to Python's standard search list when importing packages and modules

MPLCONFIGDIR

This is the directory used to store user customizations to matplotlib, as well as some caches to improve performance. If MPLCONFIGDIR is not defined, HOME/.config/matplotlib is generally used on unix-like systems and HOME/.matplotlib is used on other platforms, if they are writable. Otherwise, the python standard library `tempfile.gettempdir()` is used to find a base directory in which the matplotlib subdirectory is created.

MPLBACKEND

This optional variable can be set to choose the matplotlib backend. See [What is a backend?](#).

12.1 Setting environment variables in Linux and OS-X

To list the current value of PYTHONPATH, which may be empty, try:

```
echo $PYTHONPATH
```

The procedure for setting environment variables in depends on what your default shell is. BASH seems to be the most common, but CSH is also common. You should be able to determine which by running at the command prompt:

```
echo $SHELL
```

12.1.1 BASH/KSH

To create a new environment variable:

```
export PYTHONPATH=~/.Python
```

To prepend to an existing environment variable:

```
export PATH=~/.bin:${PATH}
```

The search order may be important to you, do you want ~/.bin to be searched first or last? To append to an existing environment variable:

```
export PATH=${PATH} ~/.bin
```

To make your changes available in the future, add the commands to your ~/.bashrc file.

12.1.2 CSH/TCSH

To create a new environment variable:

```
setenv PYTHONPATH ~/.Python
```

To prepend to an existing environment variable:

```
setenv PATH ~/.bin:${PATH}
```

The search order may be important to you, do you want ~/.bin to be searched first or last? To append to an existing environment variable:

```
setenv PATH ${PATH} ~/.bin
```

To make your changes available in the future, add the commands to your ~/.cshrc file.

12.2 Setting environment variables in windows

Open the Control Panel (Start → Control Panel), start the System program. Click the Advanced tab and select the Environment Variables button. You can edit or add to the User Variables.

WORKING WITH MATPLOTLIB IN VIRTUAL ENVIRONMENTS

When running Matplotlib in a [virtual environment](#) you may discover a few issues. Matplotlib itself has no issue with virtual environments. However, some of the external GUI frameworks that Matplotlib uses for interactive figures may be tricky to install in a virtual environment. Everything below assumes some familiarity with the Matplotlib backends as found in [What is a backend?](#).

If you only use the IPython and Jupyter Notebook's inline and notebook backends, or non-interactive backends, you should not have any issues and can ignore everything below.

Likewise, the Tk framework (TkAgg backend) does not require any external dependencies and is normally always available. On certain Linux distributions, a package named python-tk (or similar) needs to be installed.

Otherwise, the situation (at the time of writing) is as follows:

GUI framework	pip-installable?	conda or conda-forge-installable?
PyQt5	on Python>=3.5	yes
PyQt4	PySide: on Windows and OSX	yes
PyGObject	no	on Linux
PyGTK	no	no
wxPython	yes ¹	yes

In other cases, you need to install the package in the global (system) site-packages, and somehow make it available from within the virtual environment. This can be achieved by any of the following methods (in all cases, the system-wide Python and the virtualenv Python must be of the same version):

- Using virtualenv's `--system-site-packages` option when creating an environment adds all system-wide packages to the virtual environment. However, this breaks the isolation between the virtual environment and the system install. Among other issues it results in hard to debug problems with system packages shadowing the environment packages. If you use [virtualenvwrapper](#), this can be toggled with the `toggleglobalsitepackages` command.
- [vext](#) allows controlled access from within the virtualenv to specific system-wide packages without the overall shadowing issue. A specific package needs to be installed for each framework,

¹ OSX and Windows wheels available on PyPI. Linux wheels available but not on PyPI, see <https://wxpython.org/pages/downloads/>.

e.g. `vext.pyqt5`, etc. It is recommended to use `vext` $\geq 0.7.0$ as earlier versions misconfigure the logging system.

If you are using Matplotlib on OSX, you may also want to consider the OSX framework FAQ.

WORKING WITH MATPLOTLIB ON OSX

Contents

- Working with Matplotlib on OSX
 - Introduction
 - Short version
 - * VirtualEnv
 - * Pyenv
 - * Conda
 - Long version
 - * PYTHONHOME Function
 - PYTHONHOME and Jupyter
 - PYTHONHOME Script
 - * PythonW Compiler

14.1 Introduction

On OSX, two different types of Python builds exist: a regular build and a framework build. In order to interact correctly with OSX through the native GUI frameworks you need a framework build of Python. At the time of writing the macosx and WXAgg backends require a framework build to function correctly. This can result in issues for a Python installation not build as a framework and may also happen in virtual envs and when using (Ana)Conda. From Matplotlib 1.5 onwards, both backends check that a framework build is available and fail if a non framework build is found.

Without this check a partially functional figure is created. Among the issues with it is that it is produced in the background and cannot be put in front of any other window. Several solutions and work arounds exist see below.

14.2 Short version

14.2.1 VirtualEnv

If you are on Python 3, use `venv` instead of `virtualenv`:

```
python -m venv my-virtualenv
source my-virtualenv/bin/activate
```

Otherwise you will need one of the workarounds below.

14.2.2 Pyenv

If you are using `pyenv` and `virtualenv` you can enable your python version to be installed as a framework:

```
PYTHON_CONFIGURE_OPTS="--enable-framework" pyenv install x.x.x
```

14.2.3 Conda

The default python provided in (Ana)Conda is not a framework build. However, the Conda developers have made it easy to install a framework build in both the main environment and in Conda envs. To use this install `python.app` `conda install python.app` and use `pythonw` rather than `python`

14.3 Long version

Unfortunately `virtualenv` creates a non framework build even if created from a framework build of Python. As documented above you can use `venv` as an alternative on Python 3.

The issue has been reported on the `virtualenv` bug tracker [here](#) and [here](#)

Until this is fixed, one of the following workarounds can be used:

14.3.1 PYTHONHOME Function

The best known work around is to use the non `virtualenv` python along with the `PYTHONHOME` environment variable. This can be done by defining a function in your `.bashrc` using

```
function frameworkpython {
  if [[ ! -z "$VIRTUAL_ENV" ]]; then
    PYTHONHOME=$VIRTUAL_ENV /usr/local/bin/python "$@"
  else
    /usr/local/bin/python "$@"
  fi
}
```

This function can then be used in all of your virtualenvs without having to fix every single one of them.

With this in place you can run `frameworkpython` to get an interactive framework build within the virtualenv. To run a script you can do `frameworkpython test.py` where `test.py` is a script that requires a framework build. To run an interactive IPython session with the framework build within the virtual environment you can do `frameworkpython -m IPython`

PYTHONHOME and Jupyter

This approach can be followed even if using [Jupyter](#) notebooks: you just need to setup a kernel with the suitable PYTHONHOME definition. The `jupyter-virtualenv-osx` script automates the creation of such a kernel.

PYTHONHOME Script

An alternative work around borrowed from the [WX wiki](#), is to use the non virtualenv python along with the PYTHONHOME environment variable. This can be implemented in a script as below. To use this modify PYVER and PATHTOPYTHON and put the script in the virtualenv bin directory i.e. `PATHTOENV/bin/frameworkpython`

```
#!/bin/bash

# what real Python executable to use
PYVER=2.7
PATHTOPYTHON=/usr/local/bin/
PYTHON=${PATHTOPYTHON}python${PYVER}

# find the root of the virtualenv, it should be the parent of the dir this script is in
ENV=`$PYTHON -c "import os; print(os.path.abspath(os.path.join(os.path.dirname(\"$0\"), '..\n→')))"`

# now run Python with the virtualenv set as Python's HOME
export PYTHONHOME=$ENV
exec $PYTHON "$@"
```

With this in place you can run `frameworkpython` as above but will need to add this script to every virtualenv

14.3.2 PythonW Compiler

In addition `virtualenv-pythonw-osx` provides an alternative workaround which may be used to solve the issue.

Part III

Toolkits

Toolkits are collections of application-specific functions that extend Matplotlib.

MPLOT3D

`mpl_toolkits.mplot3d` provides some basic 3D plotting (scatter, surf, line, mesh) tools. Not the fastest or most feature complete 3D library out there, but it ships with Matplotlib and thus may be a lighter weight solution for some use cases. Check out the [mplot3d tutorial](#) for more information.

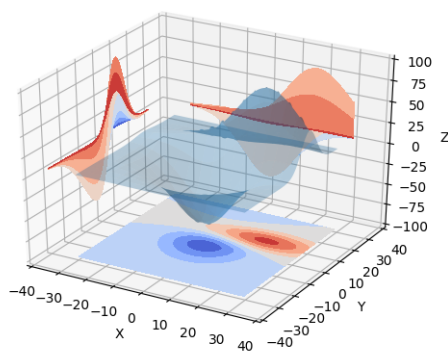


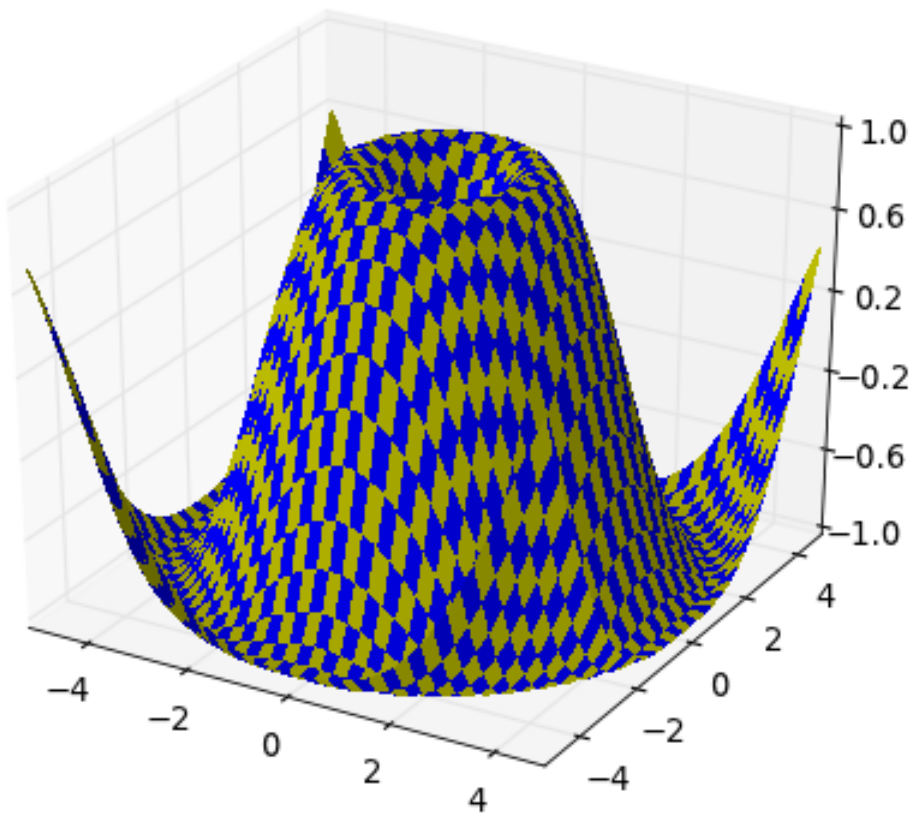
Fig. 1: Contourf3d 2

15.1 mplot3d

15.1.1 Matplotlib mplot3d toolkit

The `mplot3d` toolkit adds simple 3D plotting capabilities to matplotlib by supplying an axes object that can create a 2D projection of a 3D scene. The resulting graph will have the same look and feel as regular 2D plots.

See the [mplot3d tutorial](#) for more information on how to use this toolkit.



The interactive backends also provide the ability to rotate and zoom the 3D scene. One can rotate the 3D scene by simply clicking-and-dragging the scene. Zooming is done by right-clicking the scene and dragging the mouse up and down. Note that one does not use the zoom button like one would use for regular 2D plots.

mplot3d FAQ

How is mplot3d different from MayaVi?

[MayaVi2](#) is a very powerful and featureful 3D graphing library. For advanced 3D scenes and excellent rendering capabilities, it is highly recommended to use MayaVi2.

mplot3d was intended to allow users to create simple 3D graphs with the same "look-and-feel" as matplotlib's 2D plots. Furthermore, users can use the same toolkit that they are already familiar with to generate both their 2D and 3D plots.

My 3D plot doesn't look right at certain viewing angles

This is probably the most commonly reported issue with mplot3d. The problem is that – from some viewing angles – a 3D object would appear in front of another object, even though it is physically behind it. This can result in plots that do not look "physically correct."

Unfortunately, while some work is being done to reduce the occurrence of this artifact, it is currently an intractable problem, and can not be fully solved until matplotlib supports 3D graphics rendering at its core.

The problem occurs due to the reduction of 3D data down to 2D + z-order scalar. A single value represents the 3rd dimension for all parts of 3D objects in a collection. Therefore, when the bounding boxes of two collections intersect, it becomes possible for this artifact to occur. Furthermore, the intersection of two 3D objects (such as polygons or patches) can not be rendered properly in matplotlib's 2D rendering engine.

This problem will likely not be solved until OpenGL support is added to all of the backends (patches are greatly welcomed). Until then, if you need complex 3D scenes, we recommend using [MayaVi](#).

I don't like how the 3D plot is laid out, how do I change that?

Historically, mplot3d has suffered from a hard-coding of parameters used to control visuals such as label spacing, tick length, and grid line width. Work is being done to eliminate this issue. For matplotlib v1.1.0, there is a semi-official manner to modify these parameters. See the note in the `axis3d` section of the `mplot3d` API documentation for more information.

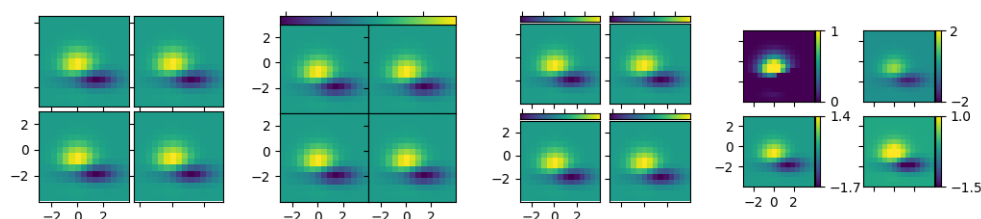
15.2 Links

- `mpl3d` API: `mplot3d` API

MATPLOTLIB AXES_GRID1 TOOLKIT

The matplotlib `mpl_toolkits.axes_grid1` toolkit is a collection of helper classes to ease displaying multiple images in matplotlib. While the `aspect` parameter in matplotlib adjust the position of the single axes, `axes_grid1` toolkit provides a framework to adjust the position of multiple axes according to their aspects.

See [What is axes_grid1 toolkit?](#) for a guide on the usage of `axes_grid1`.



The submodules of the `axes_grid1` API are:

<code>axes_grid1.anchored_artists</code>	
<code>axes_grid1.axes_divider</code>	The <code>axes_divider</code> module provides helper classes to adjust the positions of multiple axes at drawing time.
<code>axes_grid1.axes_grid</code>	
<code>axes_grid1.axes_rgb</code>	
<code>axes_grid1.axes_size</code>	provides a classes of simple units that will be used with <code>AxesDivider</code> class (or others) to determine the size of each axes.
<code>axes_grid1.colorbar</code>	Colorbar toolkit with two classes and a function:
<code>axes_grid1.inset_locator</code>	A collection of functions and objects for creating or placing inset axes.
<code>axes_grid1.mpl_axes</code>	
<code>axes_grid1.parasite_axes</code>	

16.1 mpl_toolkits.axes_grid1.anchored_artists

16.1.1 Classes

<code>AnchoredAuxTransformBox(transform, loc[, ...])</code>	An anchored container with transformed coordinates.
<code>AnchoredDrawingArea(width, height, xdescent, ...)</code>	An anchored container with a fixed size and fillable <code>DrawingArea</code> .
<code>AnchoredEllipse(transform, width, height, ...)</code>	Draw an anchored ellipse of a given size.
<code>AnchoredSizeBar(transform, size, label, loc)</code>	Draw a horizontal scale bar with a center-aligned label underneath.

`mpl_toolkits.axes_grid1.anchored_artists.AnchoredAuxTransformBox`

```
class mpl_toolkits.axes_grid1.anchored_artists.AnchoredAuxTransformBox(transform, loc,
                                                                           pad=0.4, borderpad=0.5,
                                                                           prop=None,
                                                                           frameon=True,
                                                                           **kwargs)
```

An anchored container with transformed coordinates.

Artists added to the `drawing_area` are scaled according to the coordinates of the transformation used. The dimensions of this artist will scale to contain the artists added.

Parameters

`transform` [`matplotlib.transforms.Transform`] The transformation object for the coordinate system in use, i.e., `matplotlib.axes.Axes.transData`.

`loc` [`int`] Location of this artist. Valid location codes are:

```
'upper right' : 1,
'upper left'  : 2,
'lower left'  : 3,
'lower right' : 4,
'right'       : 5,
'center left' : 6,
'center right': 7,
'lower center': 8,
'upper center': 9,
'center'      : 10
```

`pad` [`int` or `float`, optional] Padding around the child objects, in fraction of the font size. Defaults to 0.4.

`borderpad` [`int` or `float`, optional] Border padding, in fraction of the font size. Defaults to 0.5.

`prop` [matplotlib.font_manager.FontProperties, optional] Font property used as a reference for paddings.

`frameon` [bool, optional] If True, draw a box around this artists. Defaults to True.

`**kwargs` : Keyworded arguments to pass to matplotlib.offsetbox.AnchoredOffsetbox.

Examples

To display an ellipse in the upper left, with a width of 0.1 and height of 0.4 in data coordinates:

```
>>> box = AnchoredAuxTransformBox(ax.transData, loc=2)
>>> el = Ellipse((0,0), width=0.1, height=0.4, angle=30)
>>> box.drawing_area.add_artist(el)
>>> ax.add_artist(box)
```

Attributes

`drawing_area` [matplotlib.offsetbox.AuxTransformBox] A container for artists to display.

Examples using `mpl_toolkits.axes_grid1.anchored_artists.AnchoredAuxTransformBox`

- `sphx_glr_gallery_userdemo_anchored_box03.py`

`mpl_toolkits.axes_grid1.anchored_artists.AnchoredDrawingArea`

```
class mpl_toolkits.axes_grid1.anchored_artists.AnchoredDrawingArea(width, height,
                                                                    xdescent, ydescent, loc, pad=0.4,
                                                                    borderpad=0.5,
                                                                    prop=None,
                                                                    frameon=True,
                                                                    **kwargs)
```

An anchored container with a fixed size and fillable DrawingArea.

Artists added to the `drawing_area` will have their coordinates interpreted as pixels. Any transformations set on the artists will be overridden.

Parameters

`width, height` [int or float] width and height of the container, in pixels.

`xdescent, ydescent` [int or float] descent of the container in the x- and y- direction, in pixels.

`loc` [int] Location of this artist. Valid location codes are:

```
'upper right' : 1,  
'upper left'  : 2,  
'lower left'  : 3,  
'lower right' : 4,  
'right'       : 5,  
'center left' : 6,  
'center right': 7,  
'lower center': 8,  
'upper center': 9,  
'center'      : 10
```

`pad` [int or float, optional] Padding around the child objects, in fraction of the font size. Defaults to 0.4.

`borderpad` [int or float, optional] Border padding, in fraction of the font size. Defaults to 0.5.

`prop` [matplotlib.font_manager.FontProperties, optional] Font property used as a reference for paddings.

`frameon` [bool, optional] If True, draw a box around this artists. Defaults to True.

****kwargs** : Keyworded arguments to pass to matplotlib.offsetbox.AnchoredOffsetbox.

Examples

To display blue and red circles of different sizes in the upper right of an axes `ax`:

```
>>> ada = AnchoredDrawingArea(20, 20, 0, 0, loc=1, frameon=False)  
>>> ada.drawing_area.add_artist(Circle((10, 10), 10, fc="b"))  
>>> ada.drawing_area.add_artist(Circle((30, 10), 5, fc="r"))  
>>> ax.add_artist(ada)
```

Attributes

`drawing_area` [matplotlib.offsetbox.DrawingArea] A container for artists to display.

Examples using `mpl_toolkits.axes_grid1.anchored_artists.AnchoredDrawingArea`

- `sphinx_glr_gallery_axes_grid1_simple_anchored_artists.py`
- `sphinx_glr_gallery_userdemo_anchored_box02.py`

`mpl_toolkits.axes_grid1.anchored_artists.AnchoredEllipse`

```
class mpl_toolkits.axes_grid1.anchored_artists.AnchoredEllipse(transform, width, height,
                                                                angle, loc, pad=0.1, borderpad=0.1, prop=None,
                                                                frameon=True, **kwargs)
```

Draw an anchored ellipse of a given size.

Parameters

`transform` [`matplotlib.transforms.Transform`] The transformation object for the coordinate system in use, i.e., `matplotlib.axes.Axes.transData`.

`width, height` [`int` or `float`] Width and height of the ellipse, given in coordinates of transform.

`angle` [`int` or `float`] Rotation of the ellipse, in degrees, anti-clockwise.

`loc` [`int`] Location of this size bar. Valid location codes are:

```
'upper right' : 1,
'upper left'  : 2,
'lower left'  : 3,
'lower right' : 4,
'right'       : 5,
'center left' : 6,
'center right': 7,
'lower center': 8,
'upper center': 9,
'center'      : 10
```

`pad` [`int` or `float`, optional] Padding around the ellipse, in fraction of the font size. Defaults to 0.1.

`borderpad` [`int` or `float`, optional] Border padding, in fraction of the font size. Defaults to 0.1.

`frameon` [`bool`, optional] If True, draw a box around the ellipse. Defaults to True.

`prop` [`matplotlib.font_manager.FontProperties`, optional] Font property used as a reference for paddings.

`**kwargs` : Keyworded arguments to pass to `matplotlib.offsetbox.AnchoredOffsetbox`.

Attributes

`ellipse` [`matplotlib.patches.Ellipse`] Ellipse patch drawn.

Examples using `mpl_toolkits.axes_grid1.anchored_artists.AnchoredEllipse`

- `sphinx_glr_gallery_axes_grid1_simple_anchored_artists.py`

`mpl_toolkits.axes_grid1.anchored_artists.AnchoredSizeBar`

```
class mpl_toolkits.axes_grid1.anchored_artists.AnchoredSizeBar(transform, size, label, loc,
                                                                pad=0.1, borderpad=0.1,
                                                                sep=2, frameon=True,
                                                                size_vertical=0,
                                                                color='black', label_top=False, font-
                                                                properties=None,
                                                                fill_bar=None, **kwargs)
```

Draw a horizontal scale bar with a center-aligned label underneath.

Parameters

`transform` [`matplotlib.transforms.Transform`] The transformation object for the coordinate system in use, i.e., `matplotlib.axes.Axes.transData`.

`size` [`int` or `float`] Horizontal length of the size bar, given in coordinates of `transform`.

`label` [`str`] Label to display.

`loc` [`int`] Location of this size bar. Valid location codes are:

```
'upper right' : 1,
'upper left'  : 2,
'lower left'  : 3,
'lower right' : 4,
'right'       : 5,
'center left' : 6,
'center right': 7,
'lower center': 8,
'upper center': 9,
'center'      : 10
```

`pad` [`int` or `float`, optional] Padding around the label and size bar, in fraction of the font size. Defaults to 0.1.

`borderpad` [`int` or `float`, optional] Border padding, in fraction of the font size. Defaults to 0.1.

`sep` [`int` or `float`, optional] Separation between the label and the size bar, in points. Defaults to 2.

`frameon` [`bool`, optional] If True, draw a box around the horizontal bar and label. Defaults to True.

`size_vertical` [`int` or `float`, optional] Vertical length of the size bar, given in coordinates of `transform`. Defaults to 0.

`color` [`str`, optional] Color for the size bar and label. Defaults to black.

`label_top` [bool, optional] If True, the label will be over the size bar. Defaults to False.

`fontproperties` [matplotlib.font_manager.FontProperties, optional] Font properties for the label text.

`fill_bar` [bool, optional] If True and if `size_vertical` is nonzero, the size bar will be filled in with the color specified by the size bar. Defaults to True if `size_vertical` is greater than zero and False otherwise.

`**kwargs` : Keyworded arguments to pass to `matplotlib.offsetbox.AnchoredOffsetbox`.

Notes

If `prop` is passed as a keyworded argument, but `fontproperties` is not, then `prop` is assumed to be the intended `fontproperties`. Using both `prop` and `fontproperties` is not supported.

Examples

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> from mpl_toolkits.axes_grid1.anchored_artists import AnchoredSizeBar
>>> fig, ax = plt.subplots()
>>> ax.imshow(np.random.random((10,10)))
>>> bar = AnchoredSizeBar(ax.transData, 3, '3 data units', 4)
>>> ax.add_artist(bar)
>>> fig.show()
```

Using all the optional parameters

```
>>> import matplotlib.font_manager as fm
>>> fontprops = fm.FontProperties(size=14, family='monospace')
>>> bar = AnchoredSizeBar(ax.transData, 3, '3 units', 4, pad=0.5, sep=5, borderpad=0.5, frameon=False, size_vertical=0.5, color='white', fontproperties=fontprops)
```

Attributes

`size_bar` [matplotlib.offsetbox.AuxTransformBox] Container for the size bar.

`txt_label` [matplotlib.offsetbox.TextArea] Container for the label of the size bar.

Examples using `mpl_toolkits.axes_grid1.anchored_artists.AnchoredSizeBar`

- `sphx_glr_gallery_axes_grid1_simple_anchored_artists.py`
- `sphx_glr_gallery_axes_grid1_inset_locator_demo2.py`

16.2 mpl_toolkits.axes_grid1.axes_divider

The `axes_divider` module provides helper classes to adjust the positions of multiple axes at drawing time.

Divider: this is the class that is used to calculate the axes position. It divides the given rectangular area into several sub rectangles. You initialize the divider by setting the horizontal and vertical lists of sizes that the division will be based on. You then use the `new_locator` method, whose return value is a callable object that can be used to set the `axes_locator` of the axes.

16.2.1 Classes

<code>AxesDivider(axes[, xref, yref])</code>	Divider based on the pre-existing axes.
<code>AxesLocator(axes_divider, nx, ny[, nx1, ny1])</code>	A simple callable object, initialized with <code>AxesDivider</code> class, returns the position and size of the given cell.
<code>Divider(fig, pos, horizontal, vertical[, ...])</code>	This class calculates the axes position.
<code>HBoxDivider(fig, *args, **kwargs)</code>	
<code>LocatableAxes(*kl, **kw)</code>	
<code>LocatableAxesBase(*kl, **kw)</code>	
<code>SubplotDivider(fig, *args, **kwargs)</code>	The <code>Divider</code> class whose rectangle area is specified as a subplot geometry.
<code>VBoxDivider(fig, *args, **kwargs)</code>	The <code>Divider</code> class whose rectangle area is specified as a subplot geometry.

`mpl_toolkits.axes_grid1.axes_divider.AxesDivider`

```
class mpl_toolkits.axes_grid1.axes_divider.AxesDivider(axes, xref=None, yref=None)
```

Divider based on the pre-existing axes.

Parameters

`axes` [Axes]

`xref`

`yref`

```
append_axes(position, size, pad=None, add_to_figure=True, **kwargs)
```

create an axes at the given position with the same height (or width) of the main axes.

`position` ["left"|"right"|"bottom"|"top"]

`size` and `pad` should be `axes_grid.axes_size` compatible.

```
get_anchor()
```

return the anchor

```
get_aspect()
    return aspect
```

```
get_position()
    return the position of the rectangle.
```

```
get_subplotspec()
```

```
new_horizontal(size, pad=None, pack_start=False, **kwargs)
    Add a new axes on the right (or left) side of the main axes.
```

Parameters

size [axes_size or float or string] A width of the axes. If float or string is given, `from_any` function is used to create the size, with `ref_size` set to `AxesX` instance of the current axes.

pad [axes_size or float or string] Pad between the axes. It takes same argument as `size`.

pack_start [bool] If False, the new axes is appended at the end of the list, i.e., it became the right-most axes. If True, it is inserted at the start of the list, and becomes the left-most axes.

kwargs All extra keywords arguments are passed to the created axes. If `axes_class` is given, the new axes will be created as an instance of the given class. Otherwise, the same class of the main axes will be used.

```
new_vertical(size, pad=None, pack_start=False, **kwargs)
    Add a new axes on the top (or bottom) side of the main axes.
```

Parameters

size [axes_size or float or string] A height of the axes. If float or string is given, `from_any` function is used to create the size, with `ref_size` set to `AxesX` instance of the current axes.

pad [axes_size or float or string] Pad between the axes. It takes same argument as `size`.

pack_start [bool] If False, the new axes is appended at the end of the list, i.e., it became the right-most axes. If True, it is inserted at the start of the list, and becomes the left-most axes.

kwargs All extra keywords arguments are passed to the created axes. If `axes_class` is given, the new axes will be created as an instance of the given class. Otherwise, the same class of the main axes will be used.

```
mpl\_toolkits.axes\_grid1.axes\_divider.AxesLocator
```

```
class mpl\_toolkits.axes\_grid1.axes\_divider.AxesLocator(axes_divider, nx, ny, nx1=None,
                                                    ny1=None)
```

A simple callable object, initialized with `AxesDivider` class, returns the position and size of

the given cell.

Parameters

`axes_divider` [AxesDivider]

`nx, nx1` [int] Integers specifying the column-position of the cell. When `nx1` is `None`, a single `nx`-th column is specified. Otherwise location of columns spanning between `nx` to `nx1` (but excluding `nx1`-th column) is specified.

`ny, ny1` [int] Same as `nx` and `nx1`, but for row positions.

`get_subplotspec()`

`mpl_toolkits.axes_grid1.axes_divider.Divider`

`class mpl_toolkits.axes_grid1.axes_divider.Divider(fig, pos, horizontal, vertical, aspect=None, anchor='C')`

This class calculates the axes position. It divides the given rectangular area into several sub-rectangles. You initialize the divider by setting the horizontal and vertical lists of sizes (`mpl_toolkits.axes_grid1.axes_size`) that the division will be based on. You then use the `new_locator` method to create a callable object that can be used as the `axes_locator` of the axes.

Parameters

`fig` [Figure]

`pos` [tuple of 4 floats] position of the rectangle that will be divided

`horizontal` [list of `axes_size`] sizes for horizontal division

`vertical` [list of `axes_size`] sizes for vertical division

`aspect` [bool] if `True`, the overall rectangular area is reduced so that the relative part of the horizontal and vertical scales have the same scale.

`anchor` [{`'C'`, `'SW'`, `'S'`, `'SE'`, `'E'`, `'NE'`, `'N'`, `'NW'`, `'W'`}] placement of the reduced rectangle when `aspect` is `True`

`add_auto_adjustable_area(use_axes, pad=0.1, adjust_dirs=None)`

`append_size(position, size)`

`get_anchor()`

return the anchor

`get_aspect()`

return aspect

`get_horizontal()`

return horizontal sizes

`get_horizontal_sizes(renderer)`

`get_locator()`

`get_position()`
return the position of the rectangle.

`get_position_runtime(ax, renderer)`

`get_vertical()`
return vertical sizes

`get_vertical_sizes(renderer)`

`get_vsize_hsize()`

`locate(nx, ny, nx1=None, ny1=None, axes=None, renderer=None)`

Parameters

`nx, nx1` [int] Integers specifying the column-position of the cell. When `nx1` is None, a single `nx`-th column is specified. Otherwise location of columns spanning between `nx` to `nx1` (but excluding `nx1`-th column) is specified.

`ny, ny1` [int] Same as `nx` and `nx1`, but for row positions.

`axes`

`renderer`

`new_locator(nx, ny, nx1=None, ny1=None)`

Returns a new locator (`mpl_toolkits.axes_grid.axes_divider.AxesLocator`) for specified cell.

Parameters

`nx, nx1` [int] Integers specifying the column-position of the cell. When `nx1` is None, a single `nx`-th column is specified. Otherwise location of columns spanning between `nx` to `nx1` (but excluding `nx1`-th column) is specified.

`ny, ny1` [int] Same as `nx` and `nx1`, but for row positions.

`set_anchor(anchor)`

Parameters

`anchor` [{'C', 'SW', 'S', 'SE', 'E', 'NE', 'N', 'NW', 'W'}]

anchor position

value	description
'C'	Center
'SW'	bottom left
'S'	bottom
'SE'	bottom right
'E'	right
'NE'	top right
'N'	top
'NW'	top left
'W'	left

`set_aspect(aspect=False)`

Parameters

`aspect` [bool]

`set_horizontal(h)`

Parameters

`h` [list of axes_size] sizes for horizontal division

`set_locator(_locator)`

`set_position(pos)`

set the position of the rectangle.

Parameters

`pos` [tuple of 4 floats] position of the rectangle that will be divided

`set_vertical(v)`

Parameters

`v` [list of axes_size] sizes for vertical division

`mpl_toolkits.axes_grid1.axes_divider.HBoxDivider`

`class mpl_toolkits.axes_grid1.axes_divider.HBoxDivider(fig, *args, **kwargs)`

`locate(nx, ny, nx1=None, ny1=None, axes=None, renderer=None)`

Parameters

`axes_divider` [AxesDivider]

`nx, nx1` [int] Integers specifying the column-position of the cell. When `nx1` is `None`, a single `nx`-th column is specified. Otherwise location of columns spanning between `nx` to `nx1` (but excluding `nx1`-th column) is specified.

`ny, ny1` [int] Same as `nx` and `nx1`, but for row positions.

`axes`

`renderer`

`new_locator(nx, nx1=None)`

returns a new locator (`mpl_toolkits.axes_grid1.axes_divider.AxesLocator`) for specified cell.

Parameters

`nx, nx1` [int] Integers specifying the column-position of the cell. When `nx1` is `None`, a single `nx`-th column is specified. Otherwise location of columns spanning between `nx` to `nx1` (but excluding `nx1`-th column) is specified.

`ny, ny1` [int] Same as `nx` and `nx1`, but for row positions.

Examples using `mpl_toolkits.axes_grid1.axes_divider.HBoxDivider`

- `sphinx_glr_gallery_axes_grid1_demo_axes_hbox_divider.py`

`mpl_toolkits.axes_grid1.axes_divider.LocatableAxes`

```
class mpl_toolkits.axes_grid1.axes_divider.LocatableAxes(*kl, **kw)
```

`mpl_toolkits.axes_grid1.axes_divider.LocatableAxesBase`

```
class mpl_toolkits.axes_grid1.axes_divider.LocatableAxesBase(*kl, **kw)
```

```
    apply_aspect(position=None)
```

```
    draw(renderer=None, inframe=False)
```

```
    get_axes_locator()
```

```
    set_axes_locator(locator)
```

`mpl_toolkits.axes_grid1.axes_divider.SubplotDivider`

`class mpl_toolkits.axes_grid1.axes_divider.SubplotDivider(fig, *args, **kwargs)`

The Divider class whose rectangle area is specified as a subplot geometry.

Parameters

`fig` [matplotlib.figure.Figure]

`args` [tuple (numRows, numCols, plotNum)] The array of subplots in the figure has dimensions numRows, numCols, and plotNum is the number of the subplot being created. plotNum starts at 1 in the upper left corner and increases to the right.

If `numRows <= numCols <= plotNum < 10`, args can be the decimal integer `numRows * 100 + numCols * 10 + plotNum`.

`change_geometry(numrows, numcols, num)`
change subplot geometry, e.g., from 1,1,1 to 2,2,3

`get_geometry()`
get the subplot geometry, e.g., 2,2,3

`get_position()`
return the bounds of the subplot box

`get_subplotspec()`
get the SubplotSpec instance

`set_subplotspec(subplotspec)`
set the SubplotSpec instance

`update_params()`
update the subplot position from fig.subplotspars

`mpl_toolkits.axes_grid1.axes_divider.VBoxDivider`

`class mpl_toolkits.axes_grid1.axes_divider.VBoxDivider(fig, *args, **kwargs)`

The Divider class whose rectangle area is specified as a subplot geometry.

`locate(nx, ny, nx1=None, ny1=None, axes=None, renderer=None)`

Parameters

`axes_divider` [AxesDivider]

`nx, nx1` [int] Integers specifying the column-position of the cell. When `nx1` is None, a single `nx`-th column is specified. Otherwise location of columns spanning between `nx` to `nx1` (but excluding `nx1`-th column) is specified.

`ny, ny1` [int] Same as `nx` and `nx1`, but for row positions.

`axes`

renderer

`new_locator(ny, ny1=None)`

returns a new locator (`mpl_toolkits.axes_grid.axes_divider.AxesLocator`) for specified cell.

Parameters

`ny, ny1` [int] Integers specifying the row-position of the cell. When `ny1` is `None`, a single `ny`-th row is specified. Otherwise location of rows spanning between `ny` to `ny1` (but excluding `ny1`-th row) is specified.

16.2.2 Functions

`locatable_axes_factory(axes_class)`

`make_axes_area_auto_adjustable(ax[, ...])`

`make_axes_locatable(axes)`

`mpl_toolkits.axes_grid1.axes_divider.locatable_axes_factory`

`mpl_toolkits.axes_grid1.axes_divider.locatable_axes_factory(axes_class)`

`mpl_toolkits.axes_grid1.axes_divider.make_axes_area_auto_adjustable`

`mpl_toolkits.axes_grid1.axes_divider.make_axes_area_auto_adjustable(ax,`
`use_axes=None,`
`pad=0.1, ad-`
`just_dirs=None)`

Examples using `mpl_toolkits.axes_grid1.axes_divider.make_axes_area_auto_adjustable`

- `sphx_glr_gallery_axes_grid1_make_room_for_ylabel_using_axesgrid.py`

`mpl_toolkits.axes_grid1.axes_divider.make_axes_locatable`

`mpl_toolkits.axes_grid1.axes_divider.make_axes_locatable(axes)`

16.3 `mpl_toolkits.axes_grid1.axes_grid`

16.3.1 Classes

AxesGrid	alias of <code>mpl_toolkits.axes_grid1.axes_grid.ImageGrid</code>
<code>CbarAxes(*kl, **kwargs)</code>	
<code>CbarAxesBase</code>	
<code>Grid(fig, rect, nrows_ncols[, ngrids, ...])</code>	A class that creates a grid of Axes.
<code>ImageGrid(fig, rect, nrows_ncols[, ngrids, ...])</code>	A class that creates a grid of Axes.

`mpl_toolkits.axes_grid1.axes_grid.AxesGrid`

`mpl_toolkits.axes_grid1.axes_grid.AxesGrid`
alias of `mpl_toolkits.axes_grid1.axes_grid.ImageGrid`

`mpl_toolkits.axes_grid1.axes_grid.CbarAxes`

`class mpl_toolkits.axes_grid1.axes_grid.CbarAxes(*kl, **kwargs)`

`cla()`
Clear the current axes.

`mpl_toolkits.axes_grid1.axes_grid.CbarAxesBase`

`class mpl_toolkits.axes_grid1.axes_grid.CbarAxesBase`

`colorbar(mappable, **kwargs)`

`toggle_label(b)`

`mpl_toolkits.axes_grid1.axes_grid.Grid`

`class mpl_toolkits.axes_grid1.axes_grid.Grid(fig, rect, nrows_ncols, ngrids=None, direction='row', axes_pad=0.02, add_all=True, share_all=False, share_x=True, share_y=True, label_mode='L', axes_class=None)`

A class that creates a grid of Axes. In matplotlib, the axes location (and size) is specified in the normalized figure coordinates. This may not be ideal for images that needs to be displayed with a given aspect ratio. For example, displaying images of a same size with some fixed padding between them cannot be easily done in matplotlib. AxesGrid is used in such case.

Build an Grid instance with a grid `nrows*ncols` Axes in Figure `fig` with `rect=[left, bottom, width, height]` (in Figure coordinates) or the subplot position code (e.g., "121").

Optional keyword arguments:

Key-word	De-fault	Description
direction	"row"	["row" "column"]
axes_pad	0.02	float pad between axes given in inches or tuple-like of floats, (horizontal padding, vertical padding)
add_all	True	bool
share_all	False	bool
share_x	True	bool
share_y	True	bool
label_mode	"L"	["L" "1" "all"]
axes_class	None	a type object which must be a subclass of Axes

`get_aspect()`

get aspect

`get_axes_locator()`

`get_axes_pad()`

get axes_pad

Returns

tuple Padding in inches, (horizontal pad, vertical pad)

`get_divider()`

`get_geometry()`

get geometry of the grid. Returns a tuple of two integer, representing number of rows and number of columns.

`get_vsize_hsize()`

`set_aspect(aspect)`

set aspect

`set_axes_locator(locator)`

`set_axes_pad(axes_pad)`

set axes_pad

`set_label_mode(mode)`

set label_mode

`mpl_toolkits.axes_grid1.axes_grid.ImageGrid`

```
class mpl_toolkits.axes_grid1.axes_grid.ImageGrid(fig,          rect,          nrows_ncols,
                                                    ngrids=None,        direction='row',
                                                    axes_pad=0.02,        add_all=True,
                                                    share_all=False,    aspect=True,    la-
                                                    bel_mode='L',      cbar_mode=None,
                                                    cbar_location='right',
                                                    cbar_pad=None,        cbar_size='5%',
                                                    cbar_set_cax=True,
                                                    axes_class=None)
```

A class that creates a grid of Axes. In matplotlib, the axes location (and size) is specified in the normalized figure coordinates. This may not be ideal for images that needs to be displayed with a given aspect ratio. For example, displaying images of a same size with some fixed padding between them cannot be easily done in matplotlib. ImageGrid is used in such case.

Build an ImageGrid instance with a grid `nrows*ncols` Axes in Figure `fig` with `rect=[left, bottom, width, height]` (in Figure coordinates) or the subplot position code (e.g., "121").

Optional keyword arguments:

Key-word	De-fault	Description
direction	"row"	["row" "column"]
axes_pad	0.02	float pad between axes given in inches or tuple-like of floats, (horizontal padding, vertical padding)
add_all	True	bool
share_all	False	bool
aspect	True	bool
la-bel_mode	"L"	["L" "1" "all"]
cbar_mode	None	["each" "single" "edge"]
cbar_location	"right"	["left" "right" "bottom" "top"]
cbar_pad	None	
cbar_size	"5%"	
cbar_set_cax	True	bool
axes_class	None	a type object which must be a subclass of axes_grid's subclass of Axes

`cbar_set_cax` [if True, each axes in the grid has a `cax`] attribute that is bind to associated `cbar` axes.

16.4 mpl_toolkits.axes_grid1.axes_rgb

16.4.1 Classes

`RGBAxes(*kl, **kwargs)`

Parameters

<code>RGBAxesBase(*kl, **kwargs)</code>	base class for a 4-panel imshow (RGB, R, G, B)
---	--

`mpl_toolkits.axes_grid1.axes_rgb.RGBAxes`

`class mpl_toolkits.axes_grid1.axes_rgb.RGBAxes(*kl, **kwargs)`

Parameters

`pad` [float] fraction of the axes height to put as padding. defaults to 0.0
`add_all` [bool] True: Add the {rgb, r, g, b} axes to the figure defaults to True.
`axes_class` [matplotlib.axes.Axes]
`kl` : Unpacked into `axes_class()` init for RGB
`kwargs` : Unpacked into `axes_class()` init for RGB, R, G, B axes

Examples using `mpl_toolkits.axes_grid1.axes_rgb.RGBAxes`

- `sphx_glr_gallery_pyplots_whats_new_99_axes_grid.py`
- `sphx_glr_gallery_axes_grid1_simple_rgb.py`
- `sphx_glr_gallery_axes_grid1_demo_axes_rgb.py`

`mpl_toolkits.axes_grid1.axes_rgb.RGBAxesBase`

`class mpl_toolkits.axes_grid1.axes_rgb.RGBAxesBase(*kl, **kwargs)`
 base class for a 4-panel imshow (RGB, R, G, B)

Layout: +-----+---+ || R | + +---+ | RGB | G | + +---+ || B | +-----+---+

Attributes

`_defaultAxesClass` [matplotlib.axes.Axes] defaults to 'Axes' in RGBAxes child class. No default in abstract base class
`RGB` [`_defaultAxesClass`] The axes object for the three-channel imshow
`R` [`_defaultAxesClass`] The axes object for the red channel imshow

G `[_defaultAxesClass]` The axes object for the green channel imshow

B `[_defaultAxesClass]` The axes object for the blue channel imshow

Parameters

pad `[float]` fraction of the axes height to put as padding. defaults to 0.0

add_all `[bool]` True: Add the {rgb, r, g, b} axes to the figure defaults to True.

axes_class `[matplotlib.axes.Axes]`

kl : Unpacked into axes_class() init for RGB

kwargs : Unpacked into axes_class() init for RGB, R, G, B axes

add_RGB_to_figure()

Add the red, green and blue axes to the RGB composite's axes figure

imshow_rgb(r, g, b, **kwargs)

Create the four images {rgb, r, g, b}

Parameters

r `[array-like]` The red array

g `[array-like]` The green array

b `[array-like]` The blue array

kwargs `[imshow kwargs]` kwargs get unpacked into the imshow calls for the four images

Returns

rgb `[matplotlib.image.AxesImage]`

r `[matplotlib.image.AxesImage]`

g `[matplotlib.image.AxesImage]`

b `[matplotlib.image.AxesImage]`

16.4.2 Functions

imshow_rgb(ax, r, g, b, **kwargs)

make_rgb_axes(ax[, pad, axes_class, pad : fraction of the axes height.
add_all])

mpl_toolkits.axes_grid1.axes_rgb.imshow_rgb

mpl_toolkits.axes_grid1.axes_rgb.imshow_rgb(ax, r, g, b, **kwargs)

`mpl_toolkits.axes_grid1.axes_rgb.make_rgb_axes`

`mpl_toolkits.axes_grid1.axes_rgb.make_rgb_axes(ax, pad=0.01, axes_class=None, add_all=True)`
`pad` : fraction of the axes height.

Examples using `mpl_toolkits.axes_grid1.axes_rgb.make_rgb_axes`

- `sphinx_glr_gallery_axes_grid1_demo_axes_rgb.py`

16.5 `mpl_toolkits.axes_grid1.axes_size`

provides a classes of simple units that will be used with `AxesDivider` class (or others) to determine the size of each axes. The unit classes define `get_size` method that returns a tuple of two floats, meaning relative and absolute sizes, respectively.

Note that this class is nothing more than a simple tuple of two floats. Take a look at the `Divider` class to see how these two values are used.

16.5.1 Classes

<code>Add(a, b)</code>	
<code>AddList(add_list)</code>	
<code>AxesX(axes[, aspect, ref_ax])</code>	Scaled size whose relative part corresponds to the data width of the axes multiplied by the aspect.
<code>AxesY(axes[, aspect, ref_ax])</code>	Scaled size whose relative part corresponds to the data height of the axes multiplied by the aspect.
<code>Fixed(fixed_size)</code>	Simple fixed size with absolute part = <code>fixed_size</code> and relative part = 0
<code>Fraction(fraction, ref_size)</code>	An instance whose size is a fraction of the <code>ref_size</code> .
<code>GetExtentHelper(ax, direction)</code>	
<code>MaxExtent(artist_list, w_or_h)</code>	Size whose absolute part is the largest width (or height) of the given <code>artist_list</code> .
<code>MaxHeight(artist_list)</code>	Size whose absolute part is the largest height of the given <code>artist_list</code> .
<code>MaxWidth(artist_list)</code>	Size whose absolute part is the largest width of the given <code>artist_list</code> .
<code>Padded(size, pad)</code>	Return a instance where the absolute part of size is increase by the amount of <code>pad</code> .

Continued on next page

Table 8 – continued from previous page

Scalable	alias of <code>mpl_toolkits.axes_grid1.axes_size.Scaled</code>
<code>Scaled(scalable_size)</code>	Simple scaled(?) size with absolute part = 0 and relative part = <code>scalable_size</code>
<code>SizeFromFunc(func)</code>	

`mpl_toolkits.axes_grid1.axes_size.Add`

```
class mpl_toolkits.axes_grid1.axes_size.Add(a, b)
```

```
    get_size(renderer)
```

`mpl_toolkits.axes_grid1.axes_size.AddList`

```
class mpl_toolkits.axes_grid1.axes_size.AddList(add_list)
```

```
    get_size(renderer)
```

`mpl_toolkits.axes_grid1.axes_size.AxesX`

```
class mpl_toolkits.axes_grid1.axes_size.AxesX(axes, aspect=1.0, ref_ax=None)
```

Scaled size whose relative part corresponds to the data width of the axes multiplied by the aspect.

```
    get_size(renderer)
```

Examples using `mpl_toolkits.axes_grid1.axes_size.AxesX`

- `sphinx_glr_gallery_axes_grid1_simple_axes_divider3.py`
- `sphinx_glr_gallery_axes_grid1_demo_axes_hbox_divider.py`

`mpl_toolkits.axes_grid1.axes_size.AxesY`

```
class mpl_toolkits.axes_grid1.axes_size.AxesY(axes, aspect=1.0, ref_ax=None)
```

Scaled size whose relative part corresponds to the data height of the axes multiplied by the aspect.

```
    get_size(renderer)
```

Examples using `mpl_toolkits.axes_grid1.axes_size.AxesY`

- `sphinx_gallery_axes_grid1_simple_axes_divider3.py`
- `sphinx_gallery_axes_grid1_demo_axes_hbox_divider.py`

`mpl_toolkits.axes_grid1.axes_size.Fixed`

```
class mpl_toolkits.axes_grid1.axes_size.Fixed(fixed_size)
    Simple fixed size with absolute part = fixed_size and relative part = 0
    get_size(renderer)
```

Examples using `mpl_toolkits.axes_grid1.axes_size.Fixed`

- `sphinx_gallery_axes_grid1_simple_axes_divider2.py`
- `sphinx_gallery_axes_grid1_simple_axes_divider3.py`
- `sphinx_gallery_axes_grid1_demo_axes_hbox_divider.py`

`mpl_toolkits.axes_grid1.axes_size.Fraction`

```
class mpl_toolkits.axes_grid1.axes_size.Fraction(fraction, ref_size)
    An instance whose size is a fraction of the ref_size.
```

```
>>> s = Fraction(0.3, AxesX(ax))
```

```
get_size(renderer)
```

`mpl_toolkits.axes_grid1.axes_size.GetExtentHelper`

```
class mpl_toolkits.axes_grid1.axes_size.GetExtentHelper(ax, direction)
```

`mpl_toolkits.axes_grid1.axes_size.MaxExtent`

```
class mpl_toolkits.axes_grid1.axes_size.MaxExtent(artist_list, w_or_h)
    Size whose absolute part is the largest width (or height) of the given artist_list.
    add_artist(a)

    get_size(renderer)
```


`mpl_toolkits.axes_grid1.axes_size.MaxHeight`

```
class mpl_toolkits.axes_grid1.axes_size.MaxHeight(artist_list)
    Size whose absolute part is the largest height of the given artist_list.

    add_artist(a)

    get_size(renderer)
```

`mpl_toolkits.axes_grid1.axes_size.MaxWidth`

```
class mpl_toolkits.axes_grid1.axes_size.MaxWidth(artist_list)
    Size whose absolute part is the largest width of the given artist_list.

    add_artist(a)

    get_size(renderer)
```

`mpl_toolkits.axes_grid1.axes_size.Padded`

```
class mpl_toolkits.axes_grid1.axes_size.Padded(size, pad)
    Return a instance where the absolute part of size is increase by the amount of pad.

    get_size(renderer)
```

`mpl_toolkits.axes_grid1.axes_size.Scalable`

```
mpl_toolkits.axes_grid1.axes_size.Scalable
    alias of mpl_toolkits.axes_grid1.axes_size.Scaled
```

`mpl_toolkits.axes_grid1.axes_size.Scaled`

```
class mpl_toolkits.axes_grid1.axes_size.Scaled(scalable_size)
    Simple scaled(?) size with absolute part = 0 and relative part = scalable_size

    get_size(renderer)
```

Examples using `mpl_toolkits.axes_grid1.axes_size.Scaled`

- `sphx_glr_gallery_axes_grid1_simple_axes_divider2.py`
- `sphx_glr_gallery_axes_grid1_demo_axes_hbox_divider.py`

```
mpl_toolkits.axes_grid1.axes_size.SizeFromFunc
```

```
class mpl_toolkits.axes_grid1.axes_size.SizeFromFunc(func)
```

```
    get_size(renderer)
```

16.5.2 Functions

```
from_any(size[, fraction_ref])
```

Creates Fixed unit when the first argument is a float, or a Fraction unit if that is a string that ends with %.

```
mpl_toolkits.axes_grid1.axes_size.from_any
```

```
mpl_toolkits.axes_grid1.axes_size.from_any(size, fraction_ref=None)
```

Creates Fixed unit when the first argument is a float, or a Fraction unit if that is a string that ends with %. The second argument is only meaningful when Fraction unit is created.:

```
>>> a = Size.from_any(1.2) # => Size.Fixed(1.2)
>>> Size.from_any("50%", a) # => Size.Fraction(0.5, a)
```

16.6 mpl_toolkits.axes_grid1.colorbar

Colorbar toolkit with two classes and a function:

ColorbarBase the base class with full colorbar drawing functionality. It can be used as-is to make a colorbar for a given colormap; a mappable object (e.g., image) is not needed.

Colorbar the derived class for use with images or contour plots.

make_axes() a function for resizing an axes and adding a second axes suitable for a colorbar

The `colorbar()` method uses `make_axes()` and `Colorbar`; the `colorbar()` function is a thin wrapper over `colorbar()`.

```
class mpl_toolkits.axes_grid1.colorbar.CbarAxesLocator(locator=None, extend='neither',
                                                         orientation='vertical')
```

CbarAxesLocator is a `axes_locator` for colorbar axes. It adjust the position of the axes to make a room for extended ends, i.e., the extended ends are located outside the axes area.

locator [the `bbox` returned from the locator is used as a] initial axes location. If `None`, `axes.bbox` is used.

extend : same as in `ColorbarBase` **orientation** : same as in `ColorbarBase`

`get_end_vertices()`
return a tuple of two vertices for the colorbar extended ends. The first vertices is for the minimum end, and the second is for the maximum end.

`get_original_position(axes, renderer)`
get the original position of the axes.

`get_path_ends()`
get the paths for extended ends

`get_path_patch()`
get the path for axes patch

`class mpl_toolkits.axes_grid1.colorbar.Colorbar(ax, mappable, **kw)`

`add_lines(CS)`
Add the lines from a non-filled ContourSet to the colorbar.

`update_bruteforce(mappable)`
Update the colorbar artists to reflect the change of the associated mappable.

`class mpl_toolkits.axes_grid1.colorbar.ColorbarBase(ax, cmap=None, norm=None, alpha=1.0, values=None, boundaries=None, orientation='vertical', extend='neither', spacing='uniform', ticks=None, format=None, drawedges=False, filled=True)`

Draw a colorbar in an existing axes.

This is a base class for the Colorbar class, which is the basis for the `colorbar()` method and `pylab` function.

It is also useful by itself for showing a colormap. If the `cmap` kwarg is given but boundaries and values are left as `None`, then the colormap will be displayed on a 0-1 scale. To show the under- and over-value colors, specify the `norm` as:

`colors.Normalize(clip=False)`

To show the colors versus index instead of on the 0-1 scale, use:

`norm=colors.NoNorm.`

Useful attributes:

`ax` the Axes instance in which the colorbar is drawn

`lines` a LineCollection if lines were drawn, otherwise `None`

`dividers` a LineCollection if `drawedges` is `True`, otherwise `None`

Useful public methods are `set_label()` and `add_lines()`.

`add_lines(levels, colors, linewidths)`

Draw lines on the colorbar. It deletes preexisting lines.

`set_alpha(alpha)`
 set alpha value.

`set_label_text(label, **kw)`
 Label the long axis of the colorbar

`update_artists()`
 Update the colorbar associated artists, filled and ends. Note that lines are not updated.
 This needs to be called whenever `clim` of associated image changes.

`mpl_toolkits.axes_grid1.colorbar.colorbar(mappable, cax=None, ax=None, **kw)`

Create a colorbar for a `ScalarMappable` instance.

Documentation for the `pylab` thin wrapper:

Add a colorbar to a plot.

Function signatures for the `pyplot` interface; all but the first are also method signatures for the `colorbar()` method:

```
colorbar(**kwargs)
colorbar(mappable, **kwargs)
colorbar(mappable, cax=cax, **kwargs)
colorbar(mappable, ax=ax, **kwargs)
```

arguments:

`mappable` the `Image`, `ContourSet`, etc. to which the colorbar applies; this argument is mandatory for the `colorbar()` method but optional for the `colorbar()` function, which sets the default to the current image.

keyword arguments:

`cax` `None` | axes object into which the colorbar will be drawn

`ax` `None` | parent axes object from which space for a new colorbar axes will be stolen

Additional keyword arguments are of two kinds:

axes properties:

Property	Description
orientation	vertical or horizontal
fraction	0.15; fraction of original axes to use for colorbar
pad	0.05 if vertical, 0.15 if horizontal; fraction of original axes between colorbar and new image axes
shrink	1.0; fraction by which to shrink the colorbar
aspect	20; ratio of long to short dimensions

colorbar properties:

Property	Description
extend	['neither' 'both' 'min' 'max'] If not 'neither', make pointed end(s) for out-of-range values. These are set for a given colormap using the colormap <code>set_under</code> and <code>set_over</code> methods.
spacing	['uniform' 'proportional'] Uniform spacing gives each discrete color the same space; proportional makes the space proportional to the data interval.
ticks	[None list of ticks Locator object] If None, ticks are determined automatically from the input.
format	[None format string Formatter object] If None, the ScalarFormatter is used. If a format string is given, e.g., <code>'%.3f'</code> , that is used. An alternative Formatter object may be given instead.
drawedges	bool Whether to draw lines at color boundaries.

The following will probably be useful only in the context of indexed colors (that is, when the mappable has `norm=NoNorm()`), or other unusual circumstances.

Property	Description
boundaries	None or a sequence
values	None or a sequence which must be of length 1 less than the sequence of boundaries. For each region delimited by adjacent entries in boundaries, the color mapped to the corresponding value in values will be used.

If mappable is a `ContourSet`, its `extend` kwarg is included automatically.

Note that the `shrink` kwarg provides a simple way to keep a vertical colorbar, for example, from being taller than the axes of the mappable to which the colorbar is attached; but it is

a manual method requiring some trial and error. If the colorbar is too tall (or a horizontal colorbar is too wide) use a smaller value of shrink.

For more precise control, you can manually specify the positions of the axes objects in which the mappable and the colorbar are drawn. In this case, do not use any of the axes properties kwargs.

It is known that some vector graphics viewer (svg and pdf) renders white gaps between segments of the colorbar. This is due to bugs in the viewers not matplotlib. As a workaround the colorbar can be rendered with overlapping segments:

```
cbar = colorbar()
cbar.solids.set_edgecolor("face")
draw()
```

However this has negative consequences in other circumstances. Particularly with semi transparent images ($\alpha < 1$) and colorbar extensions and is not enabled by default see (issue #1188).

returns: Colorbar instance; see also its base class, ColorbarBase. Call the `set_label()` method to label the colorbar.

The `transData` of the `cax` is adjusted so that the limits in the longest axis actually corresponds to the limits in colorbar range. On the other hand, the shortest axis has a data limits of `[1,2]`, whose unconventional value is to prevent underflow when log scale is used.

`mpl_toolkits.axes_grid1.colorbar.make_axes(parent, **kw)`

Resize and reposition a parent axes, and return a child axes suitable for a colorbar

```
cax, kw = make_axes(parent, **kw)
```

Keyword arguments may include the following (with defaults):

orientation 'vertical' or 'horizontal'

Property	Description
orientation	vertical or horizontal
fraction	0.15; fraction of original axes to use for colorbar
pad	0.05 if vertical, 0.15 if horizontal; fraction of original axes between colorbar and new image axes
shrink	1.0; fraction by which to shrink the colorbar
aspect	20; ratio of long to short dimensions

All but the first of these are stripped from the input kw set.

Returns (cax, kw), the child axes and the reduced kw dictionary.

16.7 mpl_toolkits.axes_grid1.inset_locator

A collection of functions and objects for creating or placing inset axes.

16.7.1 Classes

AnchoredLocatorBase(bbox_to_anchor, ...[, ...])	
AnchoredSizeLocator(bbox_to_anchor, x_size, ...)	
AnchoredZoomLocator(parent_axes, zoom, loc)	
BboxConnector(bbox1, bbox2, loc1[, loc2])	Connect two bboxes with a straight line.
BboxConnectorPatch(bbox1, bbox2, loc1a, ...)	Connect two bboxes with a quadrilateral.
BboxPatch(bbox, **kwargs)	Patch showing the shape bounded by a Bbox.
InsetPosition(parent, lbwh)	An object for positioning an inset axes.

`mpl_toolkits.axes_grid1.inset_locator.AncoredLocatorBase`

```
class mpl_toolkits.axes_grid1.inset_locator.AncoredLocatorBase(bbox_to_anchor, offset-  
                                                                box, loc, borderpad=0.5,  
                                                                bbox_transform=None)  
  
    draw(renderer)  
        draw the artist
```

`mpl_toolkits.axes_grid1.inset_locator.AncoredSizeLocator`

```
class mpl_toolkits.axes_grid1.inset_locator.AncoredSizeLocator(bbox_to_anchor,  
                                                                x_size,          y_size,  
                                                                loc,          borderpad=0.5,  
                                                                bbox_transform=None)  
  
    get_extent(renderer)  
        return the extent of the artist. The extent of the child added with the pad is returned
```

`mpl_toolkits.axes_grid1.inset_locator.AncoredZoomLocator`

```
class mpl_toolkits.axes_grid1.inset_locator.AncoredZoomLocator(parent_axes, zoom,  
                                                                loc,          borderpad=0.5,  
                                                                bbox_to_anchor=None,  
                                                                bbox_transform=None)  
  
    get_extent(renderer)  
        return the extent of the artist. The extent of the child added with the pad is returned
```

`mpl_toolkits.axes_grid1.inset_locator.BboxConnector`

```
class mpl_toolkits.axes_grid1.inset_locator.BboxConnector(bbox1, bbox2, loc1, loc2=None,  
                                                           **kwargs)
```

Connect two bboxes with a straight line.

Parameters

`bbox1, bbox2` [`matplotlib.transforms.Bbox`] Bounding boxes to connect.

`loc1` [{1, 2, 3, 4}] Corner of `bbox1` to draw the line. Valid values are:

```
'upper right' : 1,  
'upper left'  : 2,  
'lower left'  : 3,  
'lower right' : 4
```

`loc2` [{1, 2, 3, 4}, optional] Corner of `bbox2` to draw the line. If `None`, defaults to `loc1`. Valid values are:

```
'upper right' : 1,  
'upper left'  : 2,  
'lower left'  : 3,  
'lower right' : 4
```

`**kwargs` Patch properties for the line drawn. Valid arguments include:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
an- tialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash- seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

static connect_bbox(bbox1, bbox2, loc1, loc2=None)

Helper function to obtain a Path from one bbox to another.

Parameters

bbox1, bbox2 [matplotlib.transforms.Bbox] Bounding boxes to connect.

loc1 [{1, 2, 3, 4}] Corner of bbox1 to use. Valid values are:

```
'upper right' : 1,
'upper left'  : 2,
'lower left'  : 3,
'lower right' : 4
```

loc2 [{1, 2, 3, 4}, optional] Corner of bbox2 to use. If None, defaults to loc1. Valid values are:

```
'upper right' : 1,
'upper left'  : 2,
'lower left'  : 3,
'lower right' : 4
```

Returns

path [matplotlib.path.Path] A line segment from the loc1 corner of bbox1 to the loc2 corner of bbox2.

static get_bbox_edge_pos(bbox, loc)

Helper function to obtain the location of a corner of a bbox

Parameters

bbox [matplotlib.transforms.Bbox]

loc [{1, 2, 3, 4}] Corner of bbox. Valid values are:

```
'upper right' : 1,
'upper left'  : 2,
'lower left'  : 3,
'lower right' : 4
```

Returns

x, y [float] Coordinates of the corner specified by loc.

get_path()

Return the path of this patch

Examples using `mpl_toolkits.axes_grid1.inset_locator.BboxConnector`

- `sphinx_glr_gallery_subplots_axes_and_figures_axes_zoom_effect.py`

`mpl_toolkits.axes_grid1.inset_locator.BboxConnectorPatch`

```
class mpl_toolkits.axes_grid1.inset_locator.BboxConnectorPatch(bbox1,  bbox2,  loc1a,
                                                                loc2a,  loc1b,  loc2b,
                                                                **kwargs)
```

Connect two bboxes with a quadrilateral.

The quadrilateral is specified by two lines that start and end at corners of the bboxes. The four sides of the quadrilateral are defined by the two lines given, the line between the two corners specified in bbox1 and the line between the two corners specified in bbox2.

Parameters

bbox1, bbox2 [matplotlib.transforms.Bbox] Bounding boxes to connect.

loc1a, loc2a [{1, 2, 3, 4}] Corners of bbox1 and bbox2 to draw the first line.

Valid values are:

```
'upper right' : 1,  
'upper left'  : 2,  
'lower left'  : 3,  
'lower right' : 4
```

loc1b, loc2b [{1, 2, 3, 4}] Corners of bbox1 and bbox2 to draw the second line. Valid values are:

```
'upper right' : 1,  
'upper left'  : 2,  
'lower left'  : 3,  
'lower right' : 4
```

**kwargs Patch properties for the line drawn:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

`get_path()`
Return the path of this patch

Examples using `mpl_toolkits.axes_grid1.inset_locator.BboxConnectorPatch`

- `sphx_glr_gallery_subplots_axes_and_figures_axes_zoom_effect.py`

`mpl_toolkits.axes_grid1.inset_locator.BboxPatch`

`class mpl_toolkits.axes_grid1.inset_locator.BboxPatch(bbox, **kwargs)`

Patch showing the shape bounded by a Bbox.

Parameters

`bbox` [`matplotlib.transforms.Bbox`] Bbox to use for the extents of this patch.

`**kwargs` Patch properties. Valid arguments include:

Property	Description
<code>agg_filter</code>	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
<code>alpha</code>	float or None
<code>animated</code>	bool
<code>an-tialiased</code> or <code>aa</code>	bool or None
<code>capstyle</code>	['butt' 'round' 'projecting']
<code>clip_box</code>	a Bbox instance
<code>clip_on</code>	bool
<code>clip_path</code>	[(Path, Transform) Patch None]
<code>color</code>	matplotlib color spec
<code>contains</code>	a callable function
<code>edgecolor</code> or <code>ec</code>	mpl color spec, None, 'none', or 'auto'
<code>facecolor</code> or <code>fc</code>	mpl color spec, or None for default, or 'none' for no color
<code>figure</code>	a Figure instance
<code>fill</code>	bool
<code>gid</code>	an id string
<code>hatch</code>	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '.' '*']
<code>joinstyle</code>	['miter' 'round' 'bevel']
<code>label</code>	object
<code>linestyle</code> or <code>ls</code>	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
<code>linewidth</code> or <code>lw</code>	float or None for default
<code>path_effects</code>	<code>AbstractPathEffect</code>
<code>picker</code>	[None bool float callable]
<code>rasterized</code>	bool or None
<code>sketch_params</code>	(scale: float, length: float, randomness: float)
<code>snap</code>	bool or None
<code>transform</code>	<code>Transform</code>
<code>url</code>	a url string
<code>visible</code>	bool
<code>zorder</code>	float

`get_path()`
Return the path of this patch

Examples using `mpl_toolkits.axes_grid1.inset_locator.BboxPatch`

- `sphx_glr_gallery_subplots_axes_and_figures_axes_zoom_effect.py`

`mpl_toolkits.axes_grid1.inset_locator.InsetPosition`

`class mpl_toolkits.axes_grid1.inset_locator.InsetPosition(parent, lbwh)`

An object for positioning an inset axes.

This is created by specifying the normalized coordinates in the axes, instead of the figure.

Parameters

`parent` [`matplotlib.axes.Axes`] Axes to use for normalizing coordinates.

`lbwh` [iterable of four floats] The left edge, bottom edge, width, and height of the inset axes, in units of the normalized coordinate of the parent axes.

See also:

`matplotlib.axes.Axes.set_axes_locator()`

Examples

The following bounds the inset axes to a box with 20% of the parent axes's height and 40% of the width. The size of the axes specified (`[0, 0, 1, 1]`) ensures that the axes completely fills the bounding box:

```
>>> parent_axes = plt.gca()
>>> ax_ins = plt.axes([0, 0, 1, 1])
>>> ip = InsetPosition(ax, [0.5, 0.1, 0.4, 0.2])
>>> ax_ins.set_axes_locator(ip)
```

16.7.2 Functions

<code>inset_axes(parent_axes, width, height[, ...])</code>	Create an inset axes with a given width and height.
<code>mark_inset(parent_axes, inset_axes, loc1, ...)</code>	Draw a box to mark the location of an area represented by an inset axes.
<code>zoomed_inset_axes(parent_axes, zoom[, loc, ...])</code>	Create an anchored inset axes by scaling a parent axes.

`mpl_toolkits.axes_grid1.inset_locator.inset_axes`

```
mpl_toolkits.axes_grid1.inset_locator.inset_axes(parent_axes, width, height,
                                                  loc=1, bbox_to_anchor=None,
                                                  bbox_transform=None,
                                                  axes_class=None, axes_kwargs=None,
                                                  borderpad=0.5)
```

Create an inset axes with a given width and height.

Both sizes used can be specified either in inches or percentage. For example,:

```
inset_axes(parent_axes, width='40%', height='30%', loc=3)
```

creates an inset axes in the lower left corner of `parent_axes` which spans over 30% in height and 40% in width of the parent axes. Since the usage of `inset_axes` may become slightly tricky when exceeding such standard cases, it is recommended to read the examples.

Parameters

`parent_axes` [matplotlib.axes.Axes] Axes to place the inset axes.

`width, height` [float or str] Size of the inset axes to create. If a float is provided, it is the size in inches, e.g. `width=1.3`. If a string is provided, it is the size in relative units, e.g. `width='40%'`. By default, i.e. if neither `bbox_to_anchor` nor `bbox_transform` are specified, those are relative to the `parent_axes`. Otherwise they are to be understood relative to the bounding box provided via `bbox_to_anchor`.

`loc` [int or string, optional, default to 1] Location to place the inset axes. The valid locations are:

```
'upper right' : 1,
'upper left'  : 2,
'lower left'  : 3,
'lower right' : 4,
'right'       : 5,
'center left' : 6,
'center right': 7,
'lower center': 8,
'upper center': 9,
'center'      : 10
```

`bbox_to_anchor` [tuple or matplotlib.transforms.BboxBase, optional] Bbox that the inset axes will be anchored to. If `None`, `parent_axes.bbox` is used. If a tuple, can be either `[left, bottom, width, height]`, or `[left, bottom]`. If the kwargs `width` and/or `height` are specified in relative units, the 2-tuple `[left, bottom]` cannot be used. Note that the units of the bounding box are determined through the transform in use. When using `bbox_to_anchor` it almost always makes sense to also specify a `bbox_transform`. This might often be the axes transform `parent_axes.transAxes`.

`bbox_transform` [matplotlib.transforms.Transform, optional] Transformation for the `bbox` that contains the inset axes. If `None`, a `transforms.IdentityTransform` is used (i.e. pixel coordinates). This is useful when not providing any argument to `bbox_to_anchor`. When using `bbox_to_anchor` it almost always makes sense to also specify a `bbox_transform`. This might often be the axes transform `parent_axes.transAxes`. Inversely, when specifying the axes- or figure-transform here, be aware that not specifying `bbox_to_anchor` will use `parent_axes.bbox`, the units of which are in display (pixel) coordinates.

`axes_class` [matplotlib.axes.Axes type, optional] If specified, the inset axes created will be created with this class's constructor.

`axes_kwargs` [dict, optional] Keyworded arguments to pass to the constructor of the inset axes. Valid arguments include:

	Property	Description
<code>adjustable</code>	['box' 'datalim']	
<code>agg_filter</code>	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array	
<code>alpha</code>	float (0.0 transparent through 1.0 opaque)	
<code>anchor</code>	['C' 'SW' 'S' 'SE' 'E' 'NE' 'N' 'NW' 'W']	
<code>animated</code>	bool	
<code>aspect</code>	unknown	
<code>autoscale_on</code>	bool	
<code>autoscalex_on</code>	bool	
<code>autoscaley_on</code>	bool	
<code>axes_locator</code>	a callable object which takes an axes instance and renderer and returns a <code>bbox</code> .	
<code>axisbelow</code>	[bool 'line']	
<code>clip_box</code>	a <code>Bbox</code> instance	
<code>clip_on</code>	bool	
<code>clip_path</code>	[(Path, Transform) Patch None]	
<code>contains</code>	a callable function	
<code>facecolor</code>	color	
<code>fc</code>	color	
<code>figure</code>	Figure	
<code>frame_on</code>	bool	
<code>gid</code>	an id string	
<code>label</code>	object	
<code>navigate</code>	bool	
<code>navigate_mode</code>	unknown	
<code>path_effects</code>	AbstractPathEffect	
<code>picker</code>	[None bool float callable]	
<code>position</code>	unknown	
<code>rasterization_zorder</code>	float or None	
<code>rasterized</code>	bool or None	
<code>sketch_params</code>	(scale: float, length: float, randomness: float)	

Continued on next page

Table 12 – continued from previous page

	Property	Description
snap	bool or None	
title	unknown	
transform	Transform	
url	a url string	
visible	bool	
xbound	(lower: float, upper: float)	
xlabel	unknown	
xlim	(left: float, right: float)	
xmargin	float greater than -0.5	
xscale	['linear' 'log' 'symlog' 'logit' ...]	
xticklabels	list of string labels	
xticks	list of tick locations.	
ybound	(lower: float, upper: float)	
ylabel	unknown	
ylim	(bottom: float, top: float)	
ymargin	float greater than -0.5	
yscale	['linear' 'log' 'symlog' 'logit' ...]	
yticklabels	list of string labels	
yticks	list of tick locations.	
zorder	float	

borderpad [float, optional] Padding between inset axes and the bbox_to_anchor. Defaults to 0.5. The units are axes font size, i.e. for a default font size of 10 points borderpad = 0.5 is equivalent to a padding of 5 points.

Returns

inset_axes [axes_class] Inset axes object created.

Examples using `mpl_toolkits.axes_grid1.inset_locator.inset_axes`

- `sphx_glr_gallery_axes_grid1_demo_colorbar_with_inset_locator.py`
- `sphx_glr_gallery_axes_grid1_demo_colorbar_of_inset_axes.py`
- `sphx_glr_gallery_axes_grid1_inset_locator_demo.py`

`mpl_toolkits.axes_grid1.inset_locator.mark_inset`

`mpl_toolkits.axes_grid1.inset_locator.mark_inset(parent_axes, inset_axes, loc1, loc2, **kwargs)`

Draw a box to mark the location of an area represented by an inset axes.

This function draws a box in `parent_axes` at the bounding box of `inset_axes`, and shows a connection with the inset axes by drawing lines at the corners, giving a "zoomed in" effect.

Parameters

`parent_axes` [matplotlib.axes.Axes] Axes which contains the area of the inset axes.

`inset_axes` [matplotlib.axes.Axes] The inset axes.

`loc1, loc2` [{1, 2, 3, 4}] Corners to use for connecting the inset axes and the area in the parent axes.

****kwargs** Patch properties for the lines and box drawn:

Property	Description
<code>agg_filter</code>	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
<code>alpha</code>	float or None
<code>animated</code>	bool
<code>antialiased</code> or <code>aa</code>	bool or None
<code>capstyle</code>	['butt' 'round' 'projecting']
<code>clip_box</code>	a Bbox instance
<code>clip_on</code>	bool
<code>clip_path</code>	[(Path, Transform) Patch None]
<code>color</code>	matplotlib color spec
<code>contains</code>	a callable function
<code>edgecolor</code> or <code>ec</code>	mpl color spec, None, 'none', or 'auto'
<code>facecolor</code> or <code>fc</code>	mpl color spec, or None for default, or 'none' for no color
<code>figure</code>	a Figure instance
<code>fill</code>	bool
<code>gid</code>	an id string
<code>hatch</code>	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '.' '*']
<code>joinstyle</code>	['miter' 'round' 'bevel']
<code>label</code>	object
<code>linestyle</code> or <code>ls</code>	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
<code>linewidth</code> or <code>lw</code>	float or None for default
<code>path_effects</code>	AbstractPathEffect
<code>picker</code>	[None bool float callable]
<code>rasterized</code>	bool or None
<code>sketch_params</code>	(scale: float, length: float, randomness: float)
<code>snap</code>	bool or None
<code>transform</code>	Transform
<code>url</code>	a url string
<code>visible</code>	bool
<code>zorder</code>	float

Returns

- pp [matplotlib.patches.Patch] The patch drawn to represent the area of the inset axes.
- p1, p2 [matplotlib.patches.Patch] The patches connecting two corners of the inset axes and its area.

Examples using `mpl_toolkits.axes_grid1.inset_locator.mark_inset`

- `sphx_glr_gallery_axes_grid1_inset_locator_demo2.py`

`mpl_toolkits.axes_grid1.inset_locator.zoomed_inset_axes`

```
mpl_toolkits.axes_grid1.inset_locator.zoomed_inset_axes(parent_axes, zoom, loc=1,
                                                         bbox_to_anchor=None,
                                                         bbox_transform=None,
                                                         axes_class=None,
                                                         axes_kwargs=None, borderpad=0.5)
```

Create an anchored inset axes by scaling a parent axes. For usage, also see the examples.

Parameters

- parent_axes [matplotlib.axes.Axes] Axes to place the inset axes.
- zoom [float] Scaling factor of the data axes. `zoom > 1` will enlargen the coordinates (i.e., "zoomed in"), while `zoom < 1` will shrink the coordinates (i.e., "zoomed out").
- loc [int or string, optional, default to 1] Location to place the inset axes. The valid locations are:

```
'upper right' : 1,
'upper left'  : 2,
'lower left'  : 3,
'lower right' : 4,
'right'       : 5,
'center left' : 6,
'center right': 7,
'lower center': 8,
'upper center': 9,
'center'      : 10
```

- bbox_to_anchor [tuple or matplotlib.transforms.BboxBase, optional] Bbox that the inset axes will be anchored to. If `None`, `parent_axes.bbox` is used. If a tuple, can be either `[left, bottom, width, height]`, or `[left, bottom]`. If the kwargs `width` and/or `height` are specified in relative units, the 2-tuple `[left, bottom]` cannot be used. Note that the units of the bounding box are

determined through the transform in use. When using `bbox_to_anchor` it almost always makes sense to also specify a `bbox_transform`. This might often be the axes transform `parent_axes.transAxes`.

`bbox_transform` [matplotlib.transforms.Transform, optional] Transformation for the `bbox` that contains the inset axes. If `None`, a `transforms.IdentityTransform` is used (i.e. pixel coordinates). This is useful when not providing any argument to `bbox_to_anchor`. When using `bbox_to_anchor` it almost always makes sense to also specify a `bbox_transform`. This might often be the axes transform `parent_axes.transAxes`. Inversely, when specifying the axes- or figure-transform here, be aware that not specifying `bbox_to_anchor` will use `parent_axes.bbox`, the units of which are in display (pixel) coordinates.

`axes_class` [matplotlib.axes.Axes type, optional] If specified, the inset axes created will be created with this class's constructor.

`axes_kwargs` [dict, optional] Keyworded arguments to pass to the constructor of the inset axes. Valid arguments include:

	Property	Description
<code>adjustable</code>	['box' 'datalim']	
<code>agg_filter</code>		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
<code>alpha</code>		float (0.0 transparent through 1.0 opaque)
<code>anchor</code>	['C' 'SW' 'S' 'SE' 'E' 'NE' 'N' 'NW' 'W']	
<code>animated</code>	bool	
<code>aspect</code>	unknown	
<code>autoscale_on</code>	bool	
<code>autoscalex_on</code>	bool	
<code>autoscaley_on</code>	bool	
<code>axes_locator</code>		a callable object which takes an axes instance and renderer and returns a <code>bbox</code> .
<code>axisbelow</code>	[bool 'line']	
<code>clip_box</code>		a <code>Bbox</code> instance
<code>clip_on</code>	bool	
<code>clip_path</code>	[(Path, Transform) Patch None]	
<code>contains</code>		a callable function
<code>facecolor</code>	color	
<code>fc</code>	color	
<code>figure</code>	Figure	
<code>frame_on</code>	bool	
<code>gid</code>		an id string
<code>label</code>		object
<code>navigate</code>		bool
<code>navigate_mode</code>		unknown
<code>path_effects</code>		<code>AbstractPathEffect</code>
<code>picker</code>		[None bool float callable]
<code>position</code>		unknown

Continued on next page

Table 13 – continued from previous page

	Property	Description
<code>rasterization_zorder</code>	float or None	
<code>rasterized</code>	bool or None	
<code>sketch_params</code>	(scale: float, length: float, randomness: float)	
<code>snap</code>	bool or None	
<code>title</code>	unknown	
<code>transform</code>	Transform	
<code>url</code>	a url string	
<code>visible</code>	bool	
<code>xbound</code>	(lower: float, upper: float)	
<code>xlabel</code>	unknown	
<code>xlim</code>	(left: float, right: float)	
<code>xmargin</code>	float greater than -0.5	
<code>yscale</code>	['linear' 'log' 'symlog' 'logit' ...]	
<code>xticklabels</code>	list of string labels	
<code>xticks</code>	list of tick locations.	
<code>ybound</code>	(lower: float, upper: float)	
<code>ylabel</code>	unknown	
<code>ylim</code>	(bottom: float, top: float)	
<code>ymargin</code>	float greater than -0.5	
<code>yscale</code>	['linear' 'log' 'symlog' 'logit' ...]	
<code>yticklabels</code>	list of string labels	
<code>yticks</code>	list of tick locations.	
<code>zorder</code>	float	

`borderpad` [float, optional] Padding between inset axes and the `bbox_to_anchor`. Defaults to 0.5. The units are axes font size, i.e. for a default font size of 10 points `borderpad = 0.5` is equivalent to a padding of 5 points.

Returns

`inset_axes` [`axes_class`] Inset axes object created.

Examples using `mpl_toolkits.axes_grid1.inset_locator.zoomed_inset_axes`

- `sphinx_glr_gallery_axes_grid1_demo_colorbar_of_inset_axes.py`
- `sphinx_glr_gallery_axes_grid1_inset_locator_demo2.py`

16.8 `mpl_toolkits.axes_grid1.mpl_axes`

16.8.1 Classes

`Axes(*kl, **kw)`

`SimpleAxisArtist(axis, axisnum, spine)`

`SimpleChainedObjects(objects)`

`mpl_toolkits.axes_grid1.mpl_axes.Axes`

`class mpl_toolkits.axes_grid1.mpl_axes.Axes(*kl, **kw)`

`class AxisDict()` -> new empty dictionary `dict(mapping)` -> new dictionary initialized from a mapping object's (key, value) pairs `dict(iterable)` -> new dictionary initialized as if via: `d = {} for k, v in iterable: d[k] = v` `dict(**kwargs)` -> new dictionary initialized with the name=value pairs in the keyword argument list. For example: `dict(one=1, two=2)`

`axis`

`cla()`

Clear the current axes.

`mpl_toolkits.axes_grid1.mpl_axes.SimpleAxisArtist`

`class mpl_toolkits.axes_grid1.mpl_axes.SimpleAxisArtist(axis, axisnum, spine)`

`label`

`major_ticklabels`

`major_ticks`

`set_label(txt)`

Set the label to `s` for auto legend.

Parameters

`s` [object] `s` will be converted to a string by calling `str` (unicode on Py2).

`set_visible(b)`

Set the artist's visibility.

Parameters

`b` [bool]

`toggle(all=None, ticks=None, ticklabels=None, label=None)`

`mpl_toolkits.axes_grid1.mpl_axes.SimpleChainedObjects`

`class mpl_toolkits.axes_grid1.mpl_axes.SimpleChainedObjects(objects)`

16.9 `mpl_toolkits.axes_grid1.parasite_axes`

16.9.1 Classes

<code>HostAxes</code>	alias of <code>mpl_toolkits.axes_grid1.parasite_axes.AxesHostAxes</code>
<hr/>	
<code>HostAxesBase(*args, **kwargs)</code>	
<code>ParasiteAxes</code>	alias of <code>mpl_toolkits.axes_grid1.parasite_axes.AxesParasite</code>
<hr/>	
<code>ParasiteAxesAuxTrans</code>	alias of <code>mpl_toolkits.axes_grid1.parasite_axes.AxesParasiteParasiteAuxTrans</code>
<hr/>	
<code>ParasiteAxesAuxTransBase(parent_axes, ...[, ...])</code>	
<hr/>	
<code>ParasiteAxesBase(parent_axes, **kwargs)</code>	

`mpl_toolkits.axes_grid1.parasite_axes.HostAxes`

`mpl_toolkits.axes_grid1.parasite_axes.HostAxes`
alias of `mpl_toolkits.axes_grid1.parasite_axes.AxesHostAxes`

`mpl_toolkits.axes_grid1.parasite_axes.HostAxesBase`

`class mpl_toolkits.axes_grid1.parasite_axes.HostAxesBase(*args, **kwargs)`

`cla()`

`draw(renderer)`

`get_aux_axes(tr, viewlim_mode='equal', axes_class=None)`

`get_tightbbox(renderer, call_axes_locator=True)`

`twin(aux_trans=None, axes_class=None)`

create a twin of `Axes` for generating a plot with a shared x-axis but independent y axis. The y-axis of self will have ticks on left and the returned axes will have ticks on the right

`twinx(axes_class=None)`
 create a twin of Axes for generating a plot with a sharex x-axis but independent y axis.
 The y-axis of self will have ticks on left and the returned axes will have ticks on the right

`twiny(axes_class=None)`
 create a twin of Axes for generating a plot with a shared y-axis but independent x axis.
 The x-axis of self will have ticks on bottom and the returned axes will have ticks on the top

`mpl_toolkits.axes_grid1.parasite_axes.ParasiteAxes`

`mpl_toolkits.axes_grid1.parasite_axes.ParasiteAxes`
 alias of `mpl_toolkits.axes_grid1.parasite_axes.AxesParasite`

`mpl_toolkits.axes_grid1.parasite_axes.ParasiteAxesAuxTrans`

`mpl_toolkits.axes_grid1.parasite_axes.ParasiteAxesAuxTrans`
 alias of `mpl_toolkits.axes_grid1.parasite_axes.AxesParasiteParasiteAuxTrans`

`mpl_toolkits.axes_grid1.parasite_axes.ParasiteAxesAuxTransBase`

`class mpl_toolkits.axes_grid1.parasite_axes.ParasiteAxesAuxTransBase(parent_axes,
 aux_transform,
 viewlim_mode=None,
 **kwargs)`

`apply_aspect(position=None)`

`contour(*XYCL, **kwargs)`

`contourf(*XYCL, **kwargs)`

`get_viewlim_mode()`

`pcolor(*XYC, **kwargs)`

`pcolormesh(*XYC, **kwargs)`

`set_viewlim_mode(mode)`

`update_viewlim()`

`mpl_toolkits.axes_grid1.parasite_axes.ParasiteAxesBase`

`class mpl_toolkits.axes_grid1.parasite_axes.ParasiteAxesBase(parent_axes, **kargs)`

`cla()`

`get_images_artists()`

16.9.2 Functions

<code>host_axes(*args, **kwargs)</code>	Create axes that can act as a hosts to parasitic axes.
---	--

<code>host_axes_class_factory([axes_class])</code>
--

<code>host_subplot(*args, **kwargs)</code>	Create a subplot that can act as a host to parasitic axes.
--	--

<code>host_subplot_class_factory(axes_class)</code>

<code>parasite_axes_auxtrans_class_factory([...])</code>
--

<code>parasite_axes_class_factory([axes_class])</code>
--

`mpl_toolkits.axes_grid1.parasite_axes.host_axes`

`mpl_toolkits.axes_grid1.parasite_axes.host_axes(*args, **kwargs)`

Create axes that can act as a hosts to parasitic axes.

Parameters

`figure` [matplotlib.figure.Figure] Figure to which the axes will be added. Defaults to the current figure `pyplot.gcf()`.

`*args, **kwargs` : Will be passed on to the underlying Axes object creation.

`mpl_toolkits.axes_grid1.parasite_axes.host_axes_class_factory`

`mpl_toolkits.axes_grid1.parasite_axes.host_axes_class_factory(axes_class=None)`

`mpl_toolkits.axes_grid1.parasite_axes.host_subplot`

`mpl_toolkits.axes_grid1.parasite_axes.host_subplot(*args, **kwargs)`

Create a subplot that can act as a host to parasitic axes.

Parameters

`figure` [matplotlib.figure.Figure] Figure to which the subplot will be added. Defaults to the current figure `pyplot.gcf()`.

`*args, **kwargs` : Will be passed on to the underlying Axes object creation.

`mpl_toolkits.axes_grid1.parasite_axes.host_subplot_class_factory`

`mpl_toolkits.axes_grid1.parasite_axes.host_subplot_class_factory(axes_class)`

`mpl_toolkits.axes_grid1.parasite_axes.parasite_axes_auxtrans_class_factory`

`mpl_toolkits.axes_grid1.parasite_axes.parasite_axes_auxtrans_class_factory(axes_class=None)`

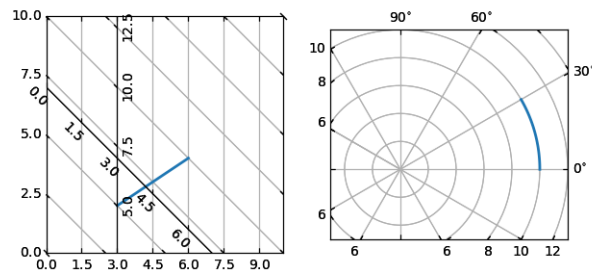
`mpl_toolkits.axes_grid1.parasite_axes.parasite_axes_class_factory`

`mpl_toolkits.axes_grid1.parasite_axes.parasite_axes_class_factory(axes_class=None)`

MATPLOTLIB AXISARTIST TOOLKIT

The `axisartist` namespace includes a derived `Axes` implementation (`mpl_toolkits.axisartist.Axes`). The biggest difference is that the artists that are responsible for drawing axis lines, ticks, ticklabels, and axis labels are separated out from the `mpl`'s `Axis` class. This change was strongly motivated to support curvilinear grid.

You can find a tutorial describing usage of `axisartist` at the [axisartist](#) user guide.



The submodules of the `axisartist` API are:

<code>axisartist.angle_helper</code>	
<code>axisartist.axes_divider</code>	
<code>axisartist.axes_grid</code>	
<code>axisartist.axes_rgb</code>	
<code>axisartist.axis_artist</code>	<code>axis_artist.py</code> module provides axis-related artists.
<code>axisartist.axisline_style</code>	
<code>axisartist.axislines</code>	<code>Axislines</code> includes modified implementation of the <code>Axes</code> class.
<code>axisartist.clip_path</code>	
<code>axisartist.floating_axes</code>	An experimental support for curvilinear grid.
<code>axisartist.grid_finder</code>	
<code>axisartist.grid_helper_curvilinear</code>	An experimental support for curvilinear grid.
<code>axisartist.parasite_axes</code>	

17.1 mpl_toolkits.axisartist.angle_helper

17.1.1 Classes

`ExtremeFinderCycle(nx, ny[, lon_cycle, ...])` When there is a cycle, e.g., longitude goes from 0-360.

`FormatterDMS`

`FormatterHMS`

`LocatorBase(den[, include_last])`

`LocatorD(den[, include_last])`

`LocatorDM(den[, include_last])`

`LocatorDMS(den[, include_last])`

`LocatorH(den[, include_last])`

`LocatorHM(den[, include_last])`

`LocatorHMS(den[, include_last])`

`mpl_toolkits.axisartist.angle_helper.ExtremeFinderCycle`

```
class mpl_toolkits.axisartist.angle_helper.ExtremeFinderCycle(nx, ny, lon_cycle=360.0,
                                                                lat_cycle=None,
                                                                lon_minmax=None,
                                                                lat_minmax=(-90, 90))
```

When there is a cycle, e.g., longitude goes from 0-360.

Examples using `mpl_toolkits.axisartist.angle_helper.ExtremeFinderCycle`

- `sphx_glr_gallery_axisartist_demo_curvelinear_grid2.py`
- `sphx_glr_gallery_axisartist_demo_floating_axis.py`
- `sphx_glr_gallery_axisartist_demo_axis_direction.py`
- `sphx_glr_gallery_axisartist_simple_axis_pad.py`
- `sphx_glr_gallery_axisartist_demo_curvelinear_grid.py`

`mpl_toolkits.axisartist.angle_helper.FormatterDMS`

```
class mpl_toolkits.axisartist.angle_helper.FormatterDMS
```

```
    deg_mark = '^{\circ}'
```

```
    fmt_d = '%d^{\circ}'
```

Examples using `mpl_toolkits.axisartist.angle_helper.FormatterDMS`

- mpl_toolkits.axisartist.angle_helper.FormatterHMS

17.1. `mpl_toolkits.axisartist.angle_helper`

```
fmt_ds = '%d.%s^\mathrm{h}$'
```

```
fmt_s_partial = '%02d^\mathrm{s}$'
```

```
fmt_ss_partial = '%02d.%s^\mathrm{s}$'
```

```
min_mark = '^^\mathrm{m}'
```

```
sec_mark = '^^\mathrm{s}'
```

Examples using `mpl_toolkits.axisartist.angle_helper.FormatterHMS`

- `sphx_glr_gallery_axisartist_demo_floating_axes.py`

`mpl_toolkits.axisartist.angle_helper.LocatorBase`

```
class mpl_toolkits.axisartist.angle_helper.LocatorBase(den, include_last=True)
```

```
    nbins
```

```
    set_params(nbins=None)
```

`mpl_toolkits.axisartist.angle_helper.LocatorD`

```
class mpl_toolkits.axisartist.angle_helper.LocatorD(den, include_last=True)
```

`mpl_toolkits.axisartist.angle_helper.LocatorDM`

```
class mpl_toolkits.axisartist.angle_helper.LocatorDM(den, include_last=True)
```

`mpl_toolkits.axisartist.angle_helper.LocatorDMS`

```
class mpl_toolkits.axisartist.angle_helper.LocatorDMS(den, include_last=True)
```

Examples using `mpl_toolkits.axisartist.angle_helper.LocatorDMS`

- `sphx_glr_gallery_axisartist_demo_floating_axis.py`
- `sphx_glr_gallery_axisartist_demo_axis_direction.py`
- `sphx_glr_gallery_axisartist_simple_axis_pad.py`
- `sphx_glr_gallery_axisartist_demo_curvelinear_grid.py`

`mpl_toolkits.axisartist.angle_helper.LocatorH`

```
class mpl_toolkits.axisartist.angle_helper.LocatorH(den, include_last=True)
```

`mpl_toolkits.axisartist.angle_helper.LocatorHM`

```
class mpl_toolkits.axisartist.angle_helper.LocatorHM(den, include_last=True)
```

`mpl_toolkits.axisartist.angle_helper.LocatorHMS`

```
class mpl_toolkits.axisartist.angle_helper.LocatorHMS(den, include_last=True)
```

Examples using `mpl_toolkits.axisartist.angle_helper.LocatorHMS`

- `sphx_glr_gallery_axisartist_demo_floating_axes.py`

17.1.2 Functions

<code>select_step(v1, v2, nv[, hour, ...])</code>
<code>select_step24(v1, v2, nv[, include_last, ...])</code>
<code>select_step360(v1, v2, nv[, include_last, ...])</code>
<code>select_step_degree(dv)</code>
<code>select_step_hour(dv)</code>
<code>select_step_sub(dv)</code>

`mpl_toolkits.axisartist.angle_helper.select_step`

```
mpl_toolkits.axisartist.angle_helper.select_step(v1, v2, nv, hour=False, include_last=True,
                                                  threshold_factor=3600.0)
```



```
mpl_toolkits.axisartist.angle_helper.select_step24

mpl_toolkits.axisartist.angle_helper.select_step24(v1, v2, nv, include_last=True, thresh-
old_factor=3600)

mpl_toolkits.axisartist.angle_helper.select_step360

mpl_toolkits.axisartist.angle_helper.select_step360(v1, v2, nv, include_last=True, thresh-
old_factor=3600)

mpl_toolkits.axisartist.angle_helper.select_step_degree

mpl_toolkits.axisartist.angle_helper.select_step_degree(dv)

mpl_toolkits.axisartist.angle_helper.select_step_hour

mpl_toolkits.axisartist.angle_helper.select_step_hour(dv)

mpl_toolkits.axisartist.angle_helper.select_step_sub

mpl_toolkits.axisartist.angle_helper.select_step_sub(dv)
```

17.2 mpl_toolkits.axisartist.axes_divider

17.3 mpl_toolkits.axisartist.axes_grid

17.3.1 Classes

AxesGrid	alias of mpl_toolkits.axisartist.axes_grid. ImageGrid
CbarAxes(*kl, **kwargs)	
Grid(fig, rect, nrows_ncols[, ngrids, ...])	Build an Grid instance with a grid nrows*ncols Axes in Figure fig with rect=[left, bottom, width, height] (in Figure coordinates) or the subplot position code (e.g., "121").

Continued on next page

Table 4 – continued from previous page

<code>ImageGrid(fig, rect, nrows_ncols[, ngrids, ...])</code>	Build an <code>ImageGrid</code> instance with a grid <code>nrows*ncols</code> Axes in Figure <code>fig</code> with <code>rect=[left, bottom, width, height]</code> (in Figure coordinates) or the subplot position code (e.g., "121").
---	--

`mpl_toolkits.axisartist.axes_grid.AxesGrid`

`mpl_toolkits.axisartist.axes_grid.AxesGrid`
alias of `mpl_toolkits.axisartist.axes_grid.ImageGrid`

`mpl_toolkits.axisartist.axes_grid.CbarAxes`

`class mpl_toolkits.axisartist.axes_grid.CbarAxes(*kl, **kwargs)`

`cla()`
Clear the current axes.

`mpl_toolkits.axisartist.axes_grid.Grid`

`class mpl_toolkits.axisartist.axes_grid.Grid(fig, rect, nrows_ncols, ngrids=None, direction='row', axes_pad=0.02, add_all=True, share_all=False, share_x=True, share_y=True, label_mode='L', axes_class=None)`

Build an `Grid` instance with a grid `nrows*ncols` Axes in Figure `fig` with `rect=[left, bottom, width, height]` (in Figure coordinates) or the subplot position code (e.g., "121").

Optional keyword arguments:

Key-word	De-fault	Description
<code>direction</code>	"row"	["row" "column"]
<code>axes_pad</code>	0.02	float pad between axes given in inches or tuple-like of floats, (horizontal padding, vertical padding)
<code>add_all</code>	True	bool
<code>share_all</code>	False	bool
<code>share_x</code>	True	bool
<code>share_y</code>	True	bool
<code>label_mode</code>	"L"	["L" "1" "all"]
<code>axes_class</code>	None	a type object which must be a subclass of <code>Axes</code>

`mpl_toolkits.axisartist.axes_grid.ImageGrid`

```
class mpl_toolkits.axisartist.axes_grid.ImageGrid(fig,          rect,          nrows_ncols,
                                                    ngrids=None,        direction='row',
                                                    axes_pad=0.02,        add_all=True,
                                                    share_all=False,    aspect=True,    la-
                                                    bel_mode='L',        cbar_mode=None,
                                                    cbar_location='right', cbar_pad=None,
                                                    cbar_size='5%',    cbar_set_cax=True,
                                                    axes_class=None)
```

Build an `ImageGrid` instance with a grid `nrows*ncols` Axes in Figure `fig` with `rect`=[left, bottom, width, height] (in Figure coordinates) or the subplot position code (e.g., "121").

Optional keyword arguments:

Key-word	De-fault	Description
direc-tion	"row"	["row" "column"]
axes_pad	0.02	float pad between axes given in inches or tuple-like of floats, (horizontal padding, vertical padding)
add_all	True	bool
share_all	False	bool
aspect	True	bool
la-bel_mode	"L"	["L" "1" "all"]
cbar_mode	None	["each" "single" "edge"]
cbar_location	"right"	["left" "right" "bottom" "top"]
cbar_pad	None	
cbar_size	"5%"	
cbar_set_cax	True	bool
axes_class	None	a type object which must be a subclass of <code>axes_grid</code> 's subclass of Axes

`cbar_set_cax` [if True, each axes in the grid has a `cax`] attribute that is bind to associated `cbar_axes`.

17.4 `mpl_toolkits.axisartist.axes_rgb`

17.4.1 Classes

```
RGBAxes(*kl, **kwargs)
```

Parameters

```
mpl_toolkits.axisartist.axes_rgb.RGBAxes
```

```
class mpl_toolkits.axisartist.axes_rgb.RGBAxes(*kl, **kwargs)
```

Parameters

`pad` [float] fraction of the axes height to put as padding. defaults to 0.0

`add_all` [bool] True: Add the {rgb, r, g, b} axes to the figure defaults to True.

`axes_class` [matplotlib.axes.Axes]

`kl` : Unpacked into `axes_class()` init for RGB

`kwargs` : Unpacked into `axes_class()` init for RGB, R, G, B axes

17.5 mpl_toolkits.axisartist.axis_artist

`axis_artist.py` module provides axis-related artists. They are

- axis line
- tick lines
- tick labels
- axis label
- grid lines

The main artist class is a `AxisArtist` and a `GridlinesCollection`. The `GridlinesCollection` is responsible for drawing grid lines and the `AxisArtist` is responsible for all other artists. The `AxisArtist` class has attributes that are associated with each type of artists.

- `line` : axis line
- `major_ticks` : major tick lines
- `major_ticklabels` : major tick labels
- `minor_ticks` : minor tick lines
- `minor_ticklabels` : minor tick labels
- `label` : axis label

Typically, the `AxisArtist` associated with a `axes` will be accessed with the `axis` dictionary of the `axes`, i.e., the `AxisArtist` for the bottom axis is

```
ax.axis["bottom"]
```

where `ax` is an instance of axes (`mpl_toolkits.axislines.Axes`). Thus, `ax.axis["bottom"].line` is an artist associated with the axis line, and `ax.axis["bottom"].major_ticks` is an artist associated with the major tick lines.

You can change the colors, fonts, line widths, etc. of these artists by calling suitable set method. For example, to change the color of the major ticks of the bottom axis to red,

```
ax.axis["bottom"].major_ticks.set_color("r")
```

However, things like the locations of ticks, and their ticklabels need to be changed from the side of the `grid_helper`.

17.5.1 axis_direction

`AxisArtist`, `AxisLabel`, `TickLabels` have `axis_direction` attribute, which adjusts the location, angle, etc.,. The `axis_direction` must be one of `[left, right, bottom, top]` and they follow the matplotlib convention for the rectangle axis.

For example, for the bottom axis (the left and right is relative to the direction of the increasing coordinate),

- ticklabels and axislabel are on the right
- ticklabels and axislabel have text angle of 0
- ticklabels are baseline, center-aligned
- axislabel is top, center-aligned

The text angles are actually relative to $(90 + \text{angle of the direction to the ticklabel})$, which gives 0 for bottom axis.

Parameter	left	bottom	right	top	ticklabels location	left	right	right	left	axislabel location
	left	right	right	left	ticklabels angle	90	0	-90	180	axislabel angle
	180	0	0	180	ticklabel va	center	baseline	center	baseline	axislabel va
	center	top	center	bottom	ticklabel ha	right	center	right	center	

Ticks are by default direct opposite side of the ticklabels. To make ticks to the same side of the ticklabels,

```
ax.axis["bottom"].major_ticks.set_ticks_out(True)
```

Following attributes can be customized (use `set_xxx` method)

- Ticks : `ticksize`, `tick_out`
- TickLabels : `pad`
- AxisLabel : `pad`

17.5.2 Classes

<code>AttributeCopier(ref_artist[, klass])</code>	
<code>AxisArtist(axes, helper[, offset, ...])</code>	An artist which draws axis (a line along which the n-th axes coord is constant) line, ticks, tick-labels, and axis label.
<code>AxisLabel(*kl, **kwargs)</code>	Axis Label.
<code>BezierPath(path, *kl, **kw)</code>	
<code>GridlinesCollection(*kl, **kwargs)</code>	which : "major" or "minor" axis : "both", "x" or "y"
<code>LabelBase(*kl, **kwargs)</code>	A base class for AxisLabel and TickLabels.
<code>TickLabels(**kwargs)</code>	Tick Labels.
<code>Ticks(ticksizes[, tick_out])</code>	Ticks are derived from Line2D, and note that ticks themselves are markers.

`mpl_toolkits.axisartist.axis_artist.AttributeCopier`

```
class mpl_toolkits.axisartist.axis_artist.AttributeCopier(ref_artist, klass=<class 'matplotlib.artist.Artist'>)
```

```
    get_attribute_from_ref_artist(attr_name, default_value)
```

```
    get_ref_artist()
```

```
    set_ref_artist(artist)
```

`mpl_toolkits.axisartist.axis_artist.AxisArtist`

```
class mpl_toolkits.axisartist.axis_artist.AxisArtist(axes, helper, offset=None,
axis_direction='bottom', **kw)
```

An artist which draws axis (a line along which the n-th axes coord is constant) line, ticks, ticklabels, and axis label.

axes : axes helper : an AxisArtistHelper instance.

LABELPAD

ZORDER = 2.5

```
    draw(renderer)
```

Draw the axis lines, tick lines and labels

```
    get_axisline_style()
```

return the current axisline style.

```
    get_helper()
```

Return axis artist helper instance.

`get_tightbbox(renderer)`

`get_transform()`

Return the Transform instance used by this artist.

`invert_ticklabel_direction()`

`set_axis_direction(axis_direction)`

Adjust the direction, text angle, text alignment of ticklabels, labels following the matplotlib convention for the rectangle axes.

The `axis_direction` must be one of [left, right, bottom, top].

property	left	bottom	right	top
ticklabels location	"_"	"+"	"+"	"_"
axislabel location	"_"	"+"	"+"	"_"
ticklabels angle	90	0	-90	180
ticklabel va	center	baseline	center	baseline
ticklabel ha	right	center	right	center
axislabel angle	180	0	0	180
axislabel va	center	top	center	bottom
axislabel ha	right	center	right	center

Note that the direction "+" and "-" are relative to the direction of the increasing coordinate. Also, the text angles are actually relative to (90 + angle of the direction to the ticklabel), which gives 0 for bottom axis.

`set_axislabel_direction(label_direction)`

Adjust the direction of the axislabel.

ACCEPTS: ["+" | "-"]

Note that the `label_direction` '+' and '-' are relative to the direction of the increasing coordinate.

`set_axisline_style(axisline_style=None, **kw)`

Set the axisline style.

`axisline_style` can be a string with axisline style name with optional comma-separated attributes. Alternatively, the `attrs` can be provided as keywords.

`set_arrowstyle("->,size=1.5")` `set_arrowstyle("->", size=1.5)`

Old `attrs` simply are forgotten.

Without argument (or with `arrowstyle=None`), return available styles as a list of strings.

`set_label(s)`

Set the label to `s` for auto legend.

Parameters

s [object] s will be converted to a string by calling `str` (unicode on Py2).

`set_ticklabel_direction(tick_direction)`
Adjust the direction of the ticklabel.

ACCEPTS: ["+" | "-"]

Note that the label_direction '+' and '-' are relative to the direction of the increasing coordinate.

`toggle(all=None, ticks=None, ticklabels=None, label=None)`
Toggle visibility of ticks, ticklabels, and (axis) label. To turn all off,

```
axis.toggle(all=False)
```

To turn all off but ticks on

```
axis.toggle(all=False, ticks=True)
```

To turn all on but (axis) label off

```
axis.toggle(all=True, label=False))
```

`mpl_toolkits.axisartist.axis_artist.AxisLabel`

`class mpl_toolkits.axisartist.axis_artist.AxisLabel(*kl, **kwargs)`

Axis Label. Derived from Text. The position of the text is updated in the fly, so changing text position has no effect. Otherwise, the properties can be changed as a normal Text.

To change the pad between ticklabels and axis label, use `set_pad`.

`draw(renderer)`
Draws the Text object to the given renderer.

`get_color()`
Return the color of the text

`get_pad()`
return pad in points. See `set_pad` for more details.

`get_ref_artist()`

`get_text()`
Get the text as string

`get_window_extent(renderer)`
Return a Bbox object bounding the text, in display units.

In addition to being used internally, this is useful for specifying clickable regions in a png file on a web page.

renderer defaults to the `_renderer` attribute of the text object. This is not assigned until the first execution of `draw()`, so you must use this kwarg if you want to call

`get_window_extent()` prior to the first `draw()`. For getting web page regions, it is simpler to call the method after saving the figure.

`dpi` defaults to `self.figure.dpi`; the renderer `dpi` is irrelevant. For the web application, if `figure.dpi` is not the value used when saving the figure, then the value that was used must be specified as the `dpi` argument.

`set_axis_direction(d)`

Adjust the text angle and text alignment of axis label according to the matplotlib convention.

property	left	bottom	right	top
axislabel angle	180	0	0	180
axislabel va	center	top	center	bottom
axislabel ha	right	center	right	center

Note that the text angles are actually relative to $(90 + \text{angle of the direction to the ticklabel})$, which gives 0 for bottom axis.

`set_default_alignment(d)`

`set_default_angle(d)`

`set_pad(pad)`

Set the pad in points. Note that the actual pad will be the sum of the internal pad and the external pad (that are set automatically by the `AxisArtist`), and it only set the internal pad

`mpl_toolkits.axisartist.axis__artist.BezierPath`

`class mpl_toolkits.axisartist.axis__artist.BezierPath(path, *kl, **kw)`

`draw(renderer)`

draw the Line with renderer unless visibility is False

`recache()`

`set_path(path)`

`mpl_toolkits.axisartist.axis__artist.GridlinesCollection`

`class mpl_toolkits.axisartist.axis__artist.GridlinesCollection(*kl, **kwargs)`

which : "major" or "minor" axis : "both", "x" or "y"

```

draw(renderer)
    Derived classes drawing method

set_axis(axis)

set_grid_helper(grid_helper)

set_which(which)

```

`mpl_toolkits.axisartist.axis_artist.LabelBase`

```

class mpl_toolkits.axisartist.axis_artist.LabelBase(*kl, **kwargs)
    A base class for AxisLabel and TickLabels. The position and angle of the text are calculated
    by to_offset_ref_angle, text_ref_angle, and offset_radius attributes.

draw(renderer)
    Draws the Text object to the given renderer.

get_window_extent(renderer)
    Return a Bbox object bounding the text, in display units.

    In addition to being used internally, this is useful for specifying clickable regions in a
    png file on a web page.

    renderer defaults to the _renderer attribute of the text object. This is not assigned
    until the first execution of draw(), so you must use this kwarg if you want to call
    get_window_extent() prior to the first draw(). For getting web page regions, it is
    simpler to call the method after saving the figure.

    dpi defaults to self.figure.dpi; the renderer dpi is irrelevant. For the web application,
    if figure.dpi is not the value used when saving the figure, then the value that was used
    must be specified as the dpi argument.

```

`mpl_toolkits.axisartist.axis_artist.TickLabels`

```

class mpl_toolkits.axisartist.axis_artist.TickLabels(**kwargs)
    Tick Labels. While derived from Text, this single artist draws all ticklabels. As in AxisLabel,
    the position of the text is updated in the fly, so changing text position has no effect.
    Otherwise, the properties can be changed as a normal Text. Unlike the ticklabels of the
    mainline matplotlib, properties of single ticklabel alone cannot modified.

    To change the pad between ticks and ticklabels, use set_pad.

draw(renderer)
    Draws the Text object to the given renderer.

get_ref_artist()

```

`get_texts_widths_heights_descents(renderer)`
return a list of width, height, descent for ticklabels.

`get_window_extents(renderer)`

`invert_axis_direction()`

`set_axis_direction(label_direction)`

Adjust the text angle and text alignment of ticklabels according to the matplotlib convention.

The `label_direction` must be one of [left, right, bottom, top].

property	left	bottom	right	top
ticklabels angle	90	0	-90	180
ticklabel va	center	baseline	center	baseline
ticklabel ha	right	center	right	center

Note that the text angles are actually relative to (90 + angle of the direction to the ticklabel), which gives 0 for bottom axis.

`set_locs_angles_labels(locs_angles_labels)`

`mpl_toolkits.axisartist.axis_artist.Ticks`

`class mpl_toolkits.axisartist.axis_artist.Ticks(ticksize, tick_out=False, **kwargs)`

Ticks are derived from `Line2D`, and note that ticks themselves are markers. Thus, you should use `set_mec`, `set_mew`, etc.

To change the tick size (length), you need to use `set_ticksize`. To change the direction of the ticks (ticks are in opposite direction of ticklabels by default), use `set_tick_out(False)`.

`draw(renderer)`
draw the Line with `renderer` unless `visibility` is `False`

`get_color()`

`get_markedgedgecolor()`

`get_markedgedgewidth()`

`get_ref_artist()`

`get_tick_out()`
Return `True` if the tick will be rotated by 180 degree.

`get_ticksizes()`
 Return length of the ticks in points.

`set_locs_angles(locs_angles)`

`set_tick_out(b)`
 set True if tick need to be rotated by 180 degree.

`set_ticksizes(ticksizes)`
 set length of the ticks in points.

17.6 `mpl_toolkits.axisartist.axisline_style`

17.6.1 Classes

<code>AxislineStyle</code>	<code>AxislineStyle</code> is a container class which defines style classes for <code>AxisArtists</code> .
----------------------------	--

`mpl_toolkits.axisartist.axisline_style.AxislineStyle`

class `mpl_toolkits.axisartist.axisline_style.AxislineStyle`

`AxislineStyle` is a container class which defines style classes for `AxisArtists`.

An instance of any axisline style class is an callable object, whose call signature is

<code>__call__(self, axis_artist, path, transform)</code>

When called, this should return a mpl artist with following methods implemented.

<pre>def set_path(self, path): # set the path for axisline. def set_line_mutation_scale(self, scale): # set the scale def draw(self, renderer): # draw</pre>
--

return the instance of the subclass with the given style name.

class `FilledArrow(size=1)`

size size of the arrow as a fraction of the ticklabel size.

`ArrowAxisClass`

alias of `_FancyAxislineStyle.FilledArrow`

```
class SimpleArrow(size=1)
    A simple arrow.

    size  size of the arrow as a fraction of the ticklabel size.

    ArrowAxisClass
        alias of __FancyAxislineStyle.SimpleArrow

    new_line(axis_artist, transform)
```

17.7 mpl_toolkits.axisartist.axislines

Axislines includes modified implementation of the Axes class. The biggest difference is that the artists responsible for drawing the axis spine, ticks, ticklabels and axis labels are separated out from mpl's Axis class. Originally, this change was motivated to support curvilinear grid. Here are a few reasons that I came up with a new axes class:

- "top" and "bottom" x-axis (or "left" and "right" y-axis) can have different ticks (tick locations and labels). This is not possible with the current mpl, although some twin axes trick can help.
- Curvilinear grid.
- angled ticks.

In the new axes class, xaxis and yaxis is set to not visible by default, and new set of artist (AxisArtist) are defined to draw axis line, ticks, ticklabels and axis label. Axes.axis attribute serves as a dictionary of these artists, i.e., ax.axis["left"] is a AxisArtist instance responsible to draw left y-axis. The default Axes.axis contains "bottom", "left", "top" and "right".

AxisArtist can be considered as a container artist and has following children artists which will draw ticks, labels, etc.

- line
- major_ticks, major_ticklabels
- minor_ticks, minor_ticklabels
- offsetText
- label

Note that these are separate artists from Axis class of the original mpl, thus most of tick-related command in the original mpl won't work, although some effort has made to work with. For example, color and markerwidth of the ax.axis["bottom"].major_ticks will follow those of Axes.xaxis unless explicitly specified.

In addition to AxisArtist, the Axes will have gridlines attribute, which obviously draws grid lines. The gridlines needs to be separated from the axis as some gridlines can never pass any axis.

17.7.1 Classes

<code>Axes(*kl, **kw)</code>	
<code>AxesZero(*kl, **kw)</code>	
<code>AxisArtistHelper</code>	<code>AxisArtistHelper</code> should define following method with given APIs.
<code>AxisArtistHelperRectlinear</code>	
<code>GridHelperBase()</code>	
<code>GridHelperRectlinear(axes)</code>	
<code>SimpleChainedObjects(objects)</code>	

`mpl_toolkits.axisartist.axislines.Axes`

`class mpl_toolkits.axisartist.axislines.Axes(*kl, **kw)`

`class AxisDict()` -> new empty dictionary
`dict(mapping)` -> new dictionary initialized from a mapping object's (key, value) pairs
`dict(iterable)` -> new dictionary initialized as if via: `d = {}` for `k, v` in `iterable`: `d[k] = v`
`dict(**kwargs)` -> new dictionary initialized with the name=value pairs in the keyword argument list. For example: `dict(one=1, two=2)`

`axis`

Set axis properties.

Valid signatures:

```
xmin, xmax, ymin, ymax = axis()
xmin, xmax, ymin, ymax = axis(list_arg)
xmin, xmax, ymin, ymax = axis(string_arg)
xmin, xmax, ymin, ymax = axis(**kwargs)
```

Parameters

`v` [list of float or {'on', 'off', 'equal', 'tight', 'scaled', 'normal', 'auto', 'image', 'square'}] Optional positional argument

Axis data limits set from a list; or a command relating to axes:

Value	Description
'on'	Toggle axis lines and labels on
'off'	Toggle axis lines and labels off
'equal'	Equal scaling by changing limits
'scaled'	Equal scaling by changing box dimensions
'tight'	Limits set such that all data is shown
'auto'	Automatic scaling, fill rectangle with data
'normal'	Same as 'auto'; deprecated
'image'	'scaled' with axis limits equal to data limits
'square'	Square plot; similar to 'scaled', but initially forcing $x_{\max} - x_{\min} = y_{\max} - y_{\min}$

`emit` [bool, optional] Passed to `set_{x,y}lim` functions, if observers are notified of axis limit change

`xmin, ymin, xmax, ymax` [float, optional] The axis limits to be set

Returns

`xmin, xmax, ymin, ymax` [float] The axis limits

`cla()`

Clear the current axes.

`draw(renderer, inframe=False)`

Draw everything (plot lines, axes, labels)

`get_children()`

return a list of child artists

`get_grid_helper()`

`get_tightbbox(renderer, call_axes_locator=True)`

Return the tight bounding box of the axes. The dimension of the Bbox in canvas coordinate.

If `call_axes_locator` is False, it does not call the `_axes_locator` attribute, which is necessary to get the correct bounding box. `call_axes_locator==False` can be used if the caller is only interested in the relative size of the `tightbbox` compared to the axes `bbox`.

`grid(b=None, which='major', axis='both', **kwargs)`

Toggle the gridlines, and optionally set the properties of the lines.

`invalidate_grid_helper()`

`new_fixed_axis(loc, offset=None)`

```
new_floating_axis(nth_coord, value, axis_direction='bottom')
```

```
new_gridlines(grid_helper=None)
```

Create and return a new GridlineCollection instance.

which : "major" or "minor" axis : "both", "x" or "y"

```
toggle_axisline(b=None)
```

```
mpl_toolkits.axisartist.axislines.AxesZero
```

```
class mpl_toolkits.axisartist.axislines.AxesZero(*kl, **kw)
```

```
mpl_toolkits.axisartist.axislines.AxisArtistHelper
```

```
class mpl_toolkits.axisartist.axislines.AxisArtistHelper
```

AxisArtistHelper should define following method with given APIs. Note that the first axes argument will be axes attribute of the caller artist.:

```
# LINE (spinal line?)

def get_line(self, axes):
    # path : Path
    return path

def get_line_transform(self, axes):
    # ...
    # trans : transform
    return trans

# LABEL

def get_label_pos(self, axes):
    # x, y : position
    return (x, y), trans

def get_label_offset_transform(self, axes,
    pad_points, fontprops, renderer,
    bboxes,
    ):
    # va : vertical alignment
    # ha : horizontal alignment
    # a : angle
    return trans, va, ha, a
```

(continues on next page)

(continued from previous page)

```
# TICK

def get_tick_transform(self, axes):
    return trans

def get_tick_iterators(self, axes):
    # iter : iterable object that yields (c, angle, l) where
    # c, angle, l is position, tick angle, and label

    return iter_major, iter_minor
```

```
class Fixed(loc, nth_coord=None)
    Helper class for a fixed (in the axes coordinate) axis.

    nth_coord = along which coordinate value varies in 2d, nth_coord = 0 -> x axis,
    nth_coord = 1 -> y axis

    get_axislabel_pos_angle(axes)
        label reference position in transAxes.

        get_label_transform() returns a transform of (transAxes+offset)

    get_axislabel_transform(axes)

    get_line(axes)

    get_line_transform(axes)

    get_nth_coord()

    get_tick_transform(axes)

class Floating(nth_coord, value)

    get_line(axes)

    get_nth_coord()
```

`mpl_toolkits.axisartist.axislines.AxisArtistHelperRectlinear`

`class mpl_toolkits.axisartist.axislines.AxisArtistHelperRectlinear`

```

class Fixed(axes, loc, nth_coord=None)
    nth_coord = along which coordinate value varies in 2d, nth_coord = 0 -> x axis,
    nth_coord = 1 -> y axis

    get_tick_iterators(axes)
        tick_loc, tick_angle, tick_label

class Floating(axes, nth_coord, passingthrough_point, axis_direction='bottom')

    get_axislabel_pos_angle(axes)
        label reference position in transAxes.

        get_label_transform() returns a transform of (transAxes+offset)

    get_axislabel_transform(axes)

    get_line(axes)

    get_line_transform(axes)

    get_tick_iterators(axes)
        tick_loc, tick_angle, tick_label

    get_tick_transform(axes)

```

`mpl_toolkits.axisartist.axislines.GridHelperBase`

```
class mpl_toolkits.axisartist.axislines.GridHelperBase
```

```

    get_gridlines(which, axis)
        Return list of grid lines as a list of paths (list of points).

        which : "major" or "minor" axis : "both", "x" or "y"

    invalidate()

    new_gridlines(ax)
        Create and return a new GridlineCollection instance.

        which : "major" or "minor" axis : "both", "x" or "y"

    update_lim(axes)

    valid()

```

`mpl_toolkits.axisartist.axislines.GridHelperRectlinear`

`class mpl_toolkits.axisartist.axislines.GridHelperRectlinear(axes)`

```
    get_gridlines(which='major', axis='both')
        return list of gridline coordinates in data coordinates.
        which : "major" or "minor" axis : "both", "x" or "y"
    new_fixed_axis(loc, nth_coord=None, axis_direction=None, offset=None,
                  axes=None)
    new_floating_axis(nth_coord, value, axis_direction='bottom', axes=None)
```

`mpl_toolkits.axisartist.axislines.SimpleChainedObjects`

`class mpl_toolkits.axisartist.axislines.SimpleChainedObjects(objects)`

17.8 `mpl_toolkits.axisartist.clip_path`

17.8.1 Functions

`atan2(dy, dx)`

`clip(xlines, ylines, x0[, clip, xdir, ydir])`

`clip_line_to_rect(xline, yline, bbox)`

`mpl_toolkits.axisartist.clip_path.atan2`

`mpl_toolkits.axisartist.clip_path.atan2(dy, dx)`

`mpl_toolkits.axisartist.clip_path.clip`

`mpl_toolkits.axisartist.clip_path.clip(xlines, ylines, x0, clip='right', xdir=True, ydir=True)`

`mpl_toolkits.axisartist.clip_path.clip_line_to_rect`

`mpl_toolkits.axisartist.clip_path.clip_line_to_rect(xline, yline, bbox)`

17.9 mpl_toolkits.axisartist.floating_axes

An experimental support for curvilinear grid.

17.9.1 Classes

ExtremeFinderFixed(extremes)	
FixedAxisArtistHelper(grid_helper, side[, ...])	nth_coord = along which coordinate value varies.
FloatingAxes	alias of mpl_toolkits.axisartist.floating_axes.Floating AxesHostAxes
FloatingAxesBase(*kl, **kwargs)	
FloatingAxisArtistHelper(grid_helper, ...[, ...])	nth_coord = along which coordinate value varies.
GridHelperCurveLinear(aux_trans, extremes[, ...])	aux_trans : a transform from the source (curved) coordinate to target (rectilinear) coordinate.

`mpl_toolkits.axisartist.floating_axes.ExtremeFinderFixed`

```
class mpl_toolkits.axisartist.floating_axes.ExtremeFinderFixed(extremes)
```

`mpl_toolkits.axisartist.floating_axes.FixedAxisArtistHelper`

```
class mpl_toolkits.axisartist.floating_axes.FixedAxisArtistHelper(grid_helper, side,
                                                                    nth_coord_ticks=None)

    nth_coord = along which coordinate value varies.  nth_coord = 0 -> x axis, nth_coord =
        1 -> y axis
    get_axislabel_pos_angle(axes)

    get_line(axes)

    get_line_transform(axes)

    get_tick_iterators(axes)
        tick_loc, tick_angle, tick_label, (optionally) tick_label
    get_tick_transform(axes)

    update_lim(axes)
```

`mpl_toolkits.axisartist.floating_axes.FloatingAxes`

`mpl_toolkits.axisartist.floating_axes.FloatingAxes`
alias of `mpl_toolkits.axisartist.floating_axes.FloatingAxesHostAxes`

`mpl_toolkits.axisartist.floating_axes.FloatingAxesBase`

`class mpl_toolkits.axisartist.floating_axes.FloatingAxesBase(*kl, **kwargs)`

`adjust_axes_lim()`

`cla()`

`mpl_toolkits.axisartist.floating_axes.FloatingAxisArtistHelper`

`class mpl_toolkits.axisartist.floating_axes.FloatingAxisArtistHelper(grid_helper,
nth_coord, value,
axis_direction=None)`

`nth_coord` = along which coordinate value varies. `nth_coord = 0` -> x axis, `nth_coord = 1` -> y axis

`mpl_toolkits.axisartist.floating_axes.GridHelperCurveLinear`

`class mpl_toolkits.axisartist.floating_axes.GridHelperCurveLinear(aux_trans, extremes,
grid_locator1=None,
grid_locator2=None,
tick_formatter1=None,
tick_formatter2=None)`

`aux_trans` : a transform from the source (curved) coordinate to target (rectilinear) coordinate. An instance of MPL's `Transform` (inverse transform should be defined) or a tuple of two callable objects which defines the transform and its inverse. The callables need take two arguments of array of source coordinates and should return two target coordinates: e.g., `x2, y2 = trans(x1, y1)`

`get_boundary()`
return Nx2 array of x,y coordinate of the boundary

`get_data_boundary(side)`
return `v = 0` , `nth=1`

`get_gridlines(which='major', axis='both')`
Return list of grid lines as a list of paths (list of points).

`which` : "major" or "minor" axis : "both", "x" or "y"

```
new_fixed_axis(loc, nth_coord=None, axis_direction=None, offset=None,
               axes=None)
```

Examples using `mpl_toolkits.axisartist.floating_axes.GridHelperCurveLinear`

- `sphinx_glr_gallery_axisartist_demo_floating_axes.py`

17.9.2 Functions

```
floatingaxes_class_factory(axes_class)
```

`mpl_toolkits.axisartist.floating_axes.floatingaxes_class_factory`

`mpl_toolkits.axisartist.floating_axes.floatingaxes_class_factory(axes_class)`

17.10 `mpl_toolkits.axisartist.grid_finder`

17.10.1 Classes

<code>DictFormatter(format_dict[, formatter])</code>	<code>format_dict</code> : dictionary for format strings to be used.
<code>ExtremeFinderSimple(nx, ny)</code>	
<code>FixedLocator(locs)</code>	
<code>FormatterPrettyPrint([useMathText])</code>	
<code>GridFinder(transform[, extreme_finder, ...])</code>	<code>transform</code> : transform from the image coordinate (which will be the <code>transData</code> of the axes) to the world coordinate.
<code>GridFinderBase(extreme_finder, ...[, ...])</code>	the <code>transData</code> of the axes to the world coordinate.
<code>MaxNLocator([nbins, steps, trim, integer, ...])</code>	

`mpl_toolkits.axisartist.grid_finder.DictFormatter`

```
class mpl_toolkits.axisartist.grid_finder.DictFormatter(format_dict, formatter=None)
    format_dict : dictionary for format strings to be used. formatter : fall-back formatter
```

Examples using `mpl_toolkits.axisartist.grid_finder.DictFormatter`

- `sphinx_glr_gallery_axisartist_demo_floating_axes.py`

`mpl_toolkits.axisartist.grid_finder.ExtremeFinderSimple`

```
class mpl_toolkits.axisartist.grid_finder.ExtremeFinderSimple(nx, ny)
```

`mpl_toolkits.axisartist.grid_finder.FixedLocator`

```
class mpl_toolkits.axisartist.grid_finder.FixedLocator(locs)
```

```
    set_factor(f)
```

Examples using `mpl_toolkits.axisartist.grid_finder.FixedLocator`

- `sphx_glr_gallery_axisartist_demo_floating_axes.py`

`mpl_toolkits.axisartist.grid_finder.FormatterPrettyPrint`

```
class mpl_toolkits.axisartist.grid_finder.FormatterPrettyPrint(useMathText=True)
```

`mpl_toolkits.axisartist.grid_finder.GridFinder`

```
class mpl_toolkits.axisartist.grid_finder.GridFinder(transform, extreme_finder=None,
                                                    grid_locator1=None,
                                                    grid_locator2=None,
                                                    tick_formatter1=None,
                                                    tick_formatter2=None)
    transform : transform from the image coordinate (which will be the transData of the axes to
    the world coordinate.
    or transform = (transform_xy, inv_transform_xy)
    locator1, locator2 : grid locator for 1st and 2nd axis.
```

`mpl_toolkits.axisartist.grid_finder.GridFinderBase`

```
class mpl_toolkits.axisartist.grid_finder.GridFinderBase(extreme_finder,
                                                         grid_locator1, grid_locator2,
                                                         tick_formatter1=None,
                                                         tick_formatter2=None)
    the transData of the axes to the world coordinate. locator1, locator2 : grid locator for 1st
    and 2nd axis.
    Derived must define "transform_xy, inv_transform_xy" (may use update_transform)
```

`get_grid_info(x1, y1, x2, y2)`

`lon_values, lat_values` [list of grid values. if integer is given,] rough number of grids in each direction.

`update(**kw)`

`update_transform(aux_trans)`

`mpl_toolkits.axisartist.grid_finder.MaxNLocator`

```
class mpl_toolkits.axisartist.grid_finder.MaxNLocator(nbins=10, steps=None, trim=True,
                                                         integer=False, symmetric=False,
                                                         prune=None)
```

`set_factor(f)`

Examples using `mpl_toolkits.axisartist.grid_finder.MaxNLocator`

- `sphx_glr_gallery_axisartist_demo_axis_direction.py`
- `sphx_glr_gallery_axisartist_simple_axis_pad.py`
- `sphx_glr_gallery_axisartist_demo_floating_axes.py`

17.11 `mpl_toolkits.axisartist.grid_helper_curvilinear`

An experimental support for curvilinear grid.

17.11.1 Classes

<code>FixedAxisArtistHelper(grid_helper, side[, ...])</code>	Helper class for a fixed axis.
<code>FloatingAxisArtistHelper(grid_helper, ...[, nth_coord = along which coordinate value varies.])</code>	
<code>GridHelperCurveLinear(aux_trans[, ...])</code>	<code>aux_trans</code> : a transform from the source (curved) coordinate to target (rectilinear) coordinate.

`mpl_toolkits.axisartist.grid_helper_curvilinear.FixedAxisArtistHelper`

```
class mpl_toolkits.axisartist.grid_helper_curvilinear.FixedAxisArtistHelper(grid_helper,
                                                                              side,
                                                                              nth_coord_ticks=None)
    Helper class for a fixed axis.

    nth_coord = along which coordinate value varies.  nth_coord = 0 -> x axis, nth_coord =
        1 -> y axis
    change_tick_coord(coord_number=None)

    get_tick_iterators(axes)
        tick_loc, tick_angle, tick_label
    get_tick_transform(axes)

    update_lim(axes)
```

`mpl_toolkits.axisartist.grid_helper_curvilinear.FloatingAxisArtistHelper`

```
class mpl_toolkits.axisartist.grid_helper_curvilinear.FloatingAxisArtistHelper(grid_helper,
                                                                              nth_coord,
                                                                              value,
                                                                              axis_direction=None)

    nth_coord = along which coordinate value varies.  nth_coord = 0 -> x axis, nth_coord =
        1 -> y axis
    get_axislabel_pos_angle(axes)

    get_axislabel_transform(axes)

    get_line(axes)

    get_line_transform(axes)

    get_tick_iterators(axes)
        tick_loc, tick_angle, tick_label, (optionally) tick_label
    get_tick_transform(axes)

    set_extremes(e1, e2)
```

`update_lim(axes)`

`mpl_toolkits.axisartist.grid_helper_curvilinear.GridHelperCurveLinear`

```
class mpl_toolkits.axisartist.grid_helper_curvilinear.GridHelperCurveLinear(aux_trans,
                                                                              ex-
                                                                              treme_finder=None,
                                                                              grid_locator1=None,
                                                                              grid_locator2=None,
                                                                              tick_formatter1=None,
                                                                              tick_formatter2=None)
```

`aux_trans` : a transform from the source (curved) coordinate to target (rectilinear) coordinate. An instance of MPL's Transform (inverse transform should be defined) or a tuple of two callable objects which defines the transform and its inverse. The callables need take two arguments of array of source coordinates and should return two target coordinates.

e.g., `x2, y2 = trans(x1, y1)`

`get_gridlines(which='major', axis='both')`

Return list of grid lines as a list of paths (list of points).

`which` : "major" or "minor" `axis` : "both", "x" or "y"

`get_tick_iterator(nth_coord, axis_side, minor=False)`

`new_fixed_axis(loc, nth_coord=None, axis_direction=None, offset=None, axes=None)`

`new_floating_axis(nth_coord, value, axes=None, axis_direction='bottom')`

`update_grid_finder(aux_trans=None, **kw)`

17.12 `mpl_toolkits.axisartist.parasite_axes`

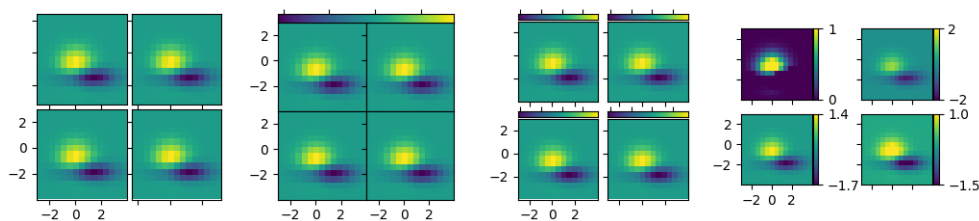
MATPLOTLIB AXES_GRID TOOLKIT

Note: AxesGrid toolkit has been a part of matplotlib since v 0.99. Originally, the toolkit had a single namespace of `axes_grid`. In more recent version, the toolkit has divided into two separate namespace (`axes_grid1` and `axisartist`). While `axes_grid` namespace is maintained for the backward compatibility, use of `axes_grid1` and `axisartist` is recommended. For the documentation on `axes_grid`, see the [previous version of the docs](#).

18.1 Matplotlib `axes_grid1` Toolkit

The matplotlib `mpl_toolkits.axes_grid1` toolkit is a collection of helper classes to ease displaying multiple images in matplotlib. While the `aspect` parameter in matplotlib adjust the position of the single axes, `axes_grid1` toolkit provides a framework to adjust the position of multiple axes according to their aspects.

See [What is `axes_grid1` toolkit?](#) for a guide on the usage of `axes_grid1`.



The submodules of the `axes_grid1` API are:

`axes_grid1.anchored_artists`

`axes_grid1.axes_divider`

The `axes_divider` module provides helper classes to adjust the positions of multiple axes at drawing time.

`axes_grid1.axes_grid`

`axes_grid1.axes_rgb`

Continued on next page

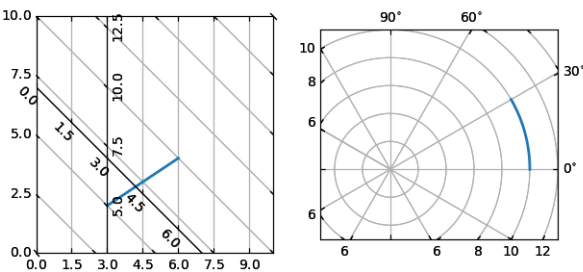
Table 1 – continued from previous page

<code>axes_grid1.axes_size</code>	provides a classes of simple units that will be used with <code>AxesDivider</code> class (or others) to determine the size of each axes.
<code>axes_grid1.colorbar</code>	Colorbar toolkit with two classes and a function:
<code>axes_grid1.inset_locator</code>	A collection of functions and objects for creating or placing inset axes.
<code>axes_grid1.mpl_axes</code>	
<code>axes_grid1.parasite_axes</code>	

18.2 Matplotlib axisartist Toolkit

The `axisartist` namespace includes a derived `Axis` implementation (`mpl_toolkits.axisartist.Axes`). The biggest difference is that the artists that are responsible for drawing axis lines, ticks, ticklabels, and axis labels are separated out from the `mpl`’s `Axis` class. This change was strongly motivated to support curvilinear grid.

You can find a tutorial describing usage of `axisartist` at the [axisartist](#) user guide.



The submodules of the `axisartist` API are:

<code>axisartist.angle_helper</code>	
<code>axisartist.axes_divider</code>	
<code>axisartist.axes_grid</code>	
<code>axisartist.axes_rgb</code>	
<code>axisartist.axis_artist</code>	<code>axis_artist.py</code> module provides axis-related artists.
<code>axisartist.axisline_style</code>	
<code>axisartist.axislines</code>	<code>Axislines</code> includes modified implementation of the <code>Axis</code> class.
<code>axisartist.clip_path</code>	
<code>axisartist.floating_axes</code>	An experimental support for curvilinear grid.
<code>axisartist.grid_finder</code>	

Continued on next page

Table 2 – continued from previous page

<code>axisartist.grid_helper_curvilinear</code>	An experimental support for curvilinear grid.
<code>axisartist.parasite_axes</code>	

Part IV

External Resources

BOOKS, CHAPTERS AND ARTICLES

- [Mastering matplotlib](#) by Duncan M. McGregor
- [Interactive Applications Using Matplotlib](#) by Benjamin Root
- [Matplotlib for Python Developers](#) by Sandro Tosi
- [Matplotlib chapter](#) by John Hunter and Michael Droettboom in *The Architecture of Open Source Applications*
- [Graphics with Matplotlib](#) by David J. Raymond
- [Ten Simple Rules for Better Figures](#) by Nicolas P. Rougier, Michael Droettboom and Philip E. Bourne
- [Learning Scientific Programming with Python chapter 7](#) by Christian Hill

CHAPTER TWENTY

VIDEOS

- [Plotting with matplotlib](#) by Mike Müller
- [Introduction to NumPy and Matplotlib](#) by Eric Jones
- [Anatomy of Matplotlib](#) by Benjamin Root
- [Data Visualization Basics with Python \(O'Reilly\)](#) by Randal S. Olson

CHAPTER TWENTYONE

TUTORIALS

- [Matplotlib tutorial](#) by Nicolas P. Rougier
- [Anatomy of Matplotlib - IPython Notebooks](#) by Benjamin Root

Part V

Third party packages

Several external packages that extend or build on Matplotlib functionality are listed below. They are maintained and distributed separately from Matplotlib and thus need to be installed individually.

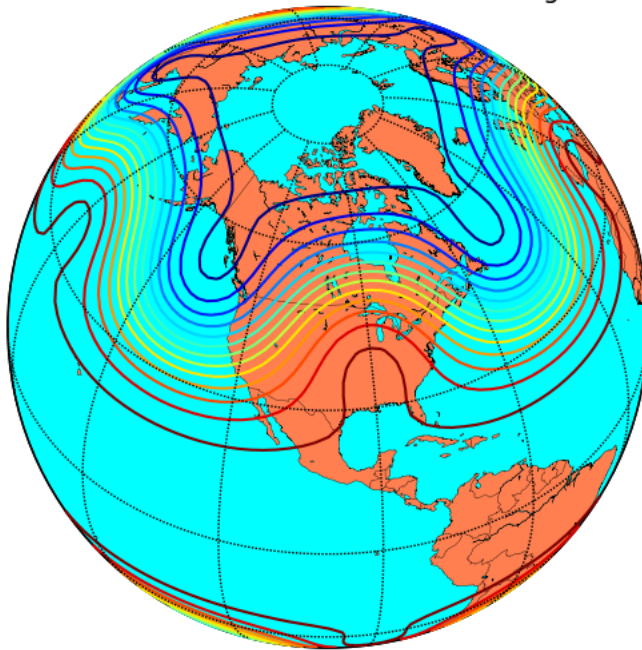
Please submit an issue or pull request on Github if you have created a package that you would like to have included. We are also happy to host third party packages within the [Matplotlib Github Organization](#).

MAPPING TOOLKITS

22.1 Basemap

Basemap plots data on map projections, with continental and political boundaries.

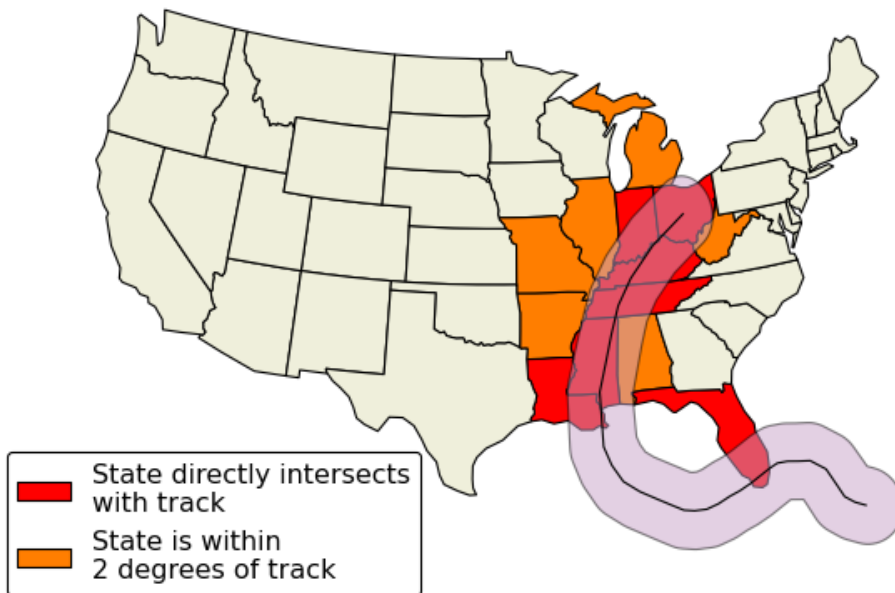
contour lines over filled continent background



22.2 Cartopy

Cartopy builds on top of Matplotlib to provide object oriented map projection definitions and close integration with Shapely for powerful yet easy-to-use vector data processing tools. An example plot from the Cartopy gallery:

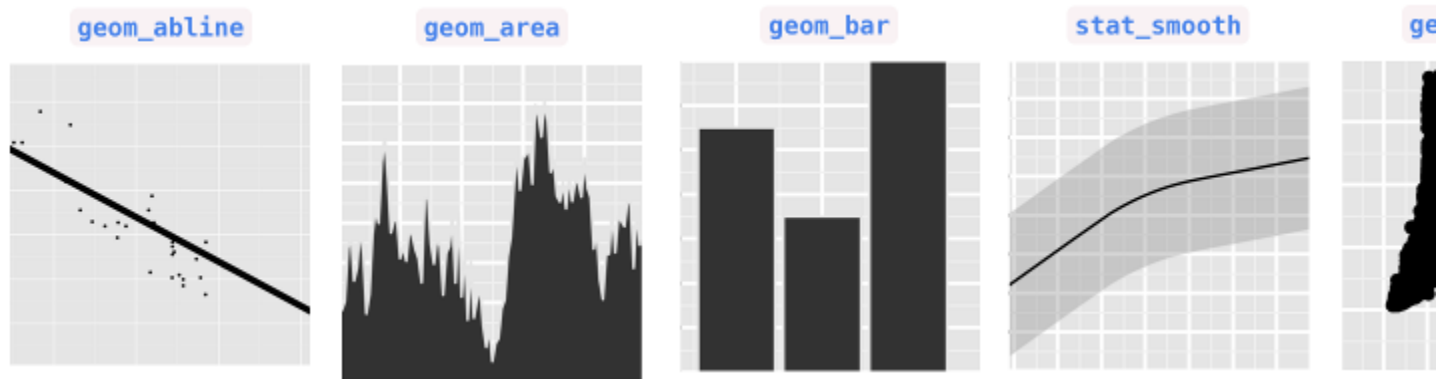
US States which intersect the track of Hurricane Katrina (2005)



DECLARATIVE LIBRARIES

23.1 ggplot

`ggplot` is a port of the R `ggplot2` package to python based on Matplotlib.

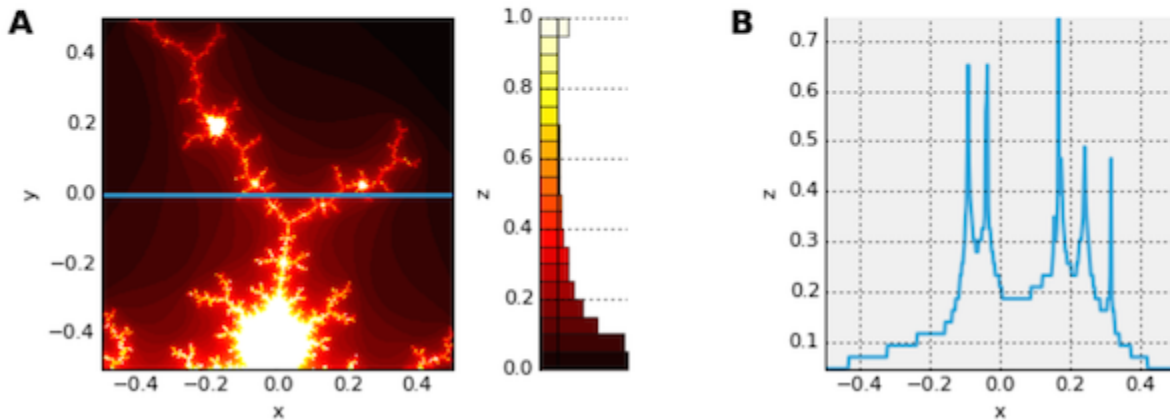


23.2 holoviews

`holoviews` makes it easier to visualize data interactively, especially in a [Jupyter notebook](#), by providing a set of declarative plotting objects that store your data and associated metadata. Your data is then immediately visualizable alongside or overlaid with other data, either statically or with automatically provided widgets for parameter exploration.

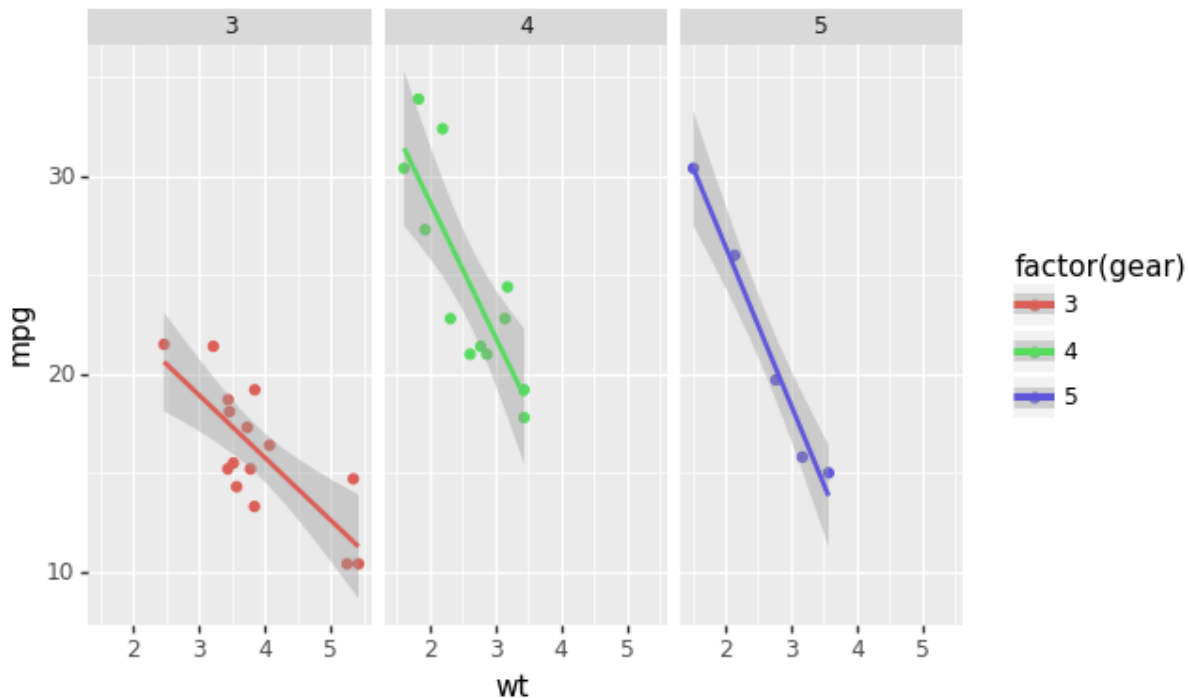
```
import numpy as np
import holoviews as hv
hv.notebook_extension('matplotlib')
fractal = hv.Image(np.load('mandelbrot.npy'))

((fractal * hv.HLine(y=0)).hist() + fractal.sample(y=0))
```



23.3 plotnine

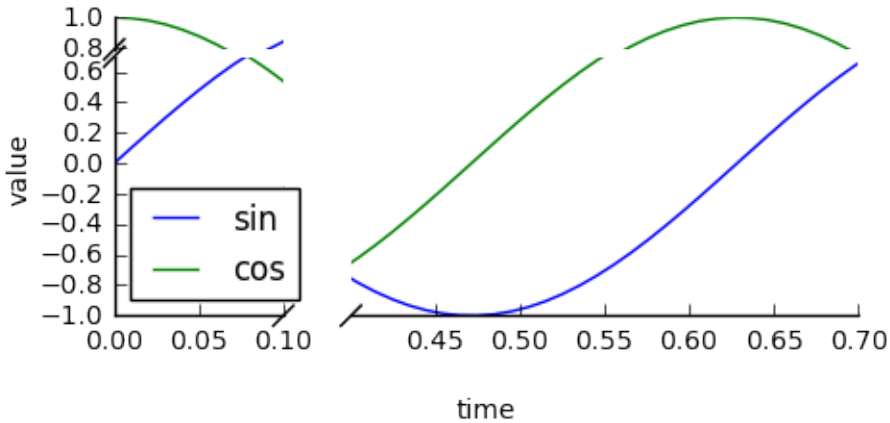
`plotnine` implements a grammar of graphics, similar to R's `ggplot2`. The grammar allows users to compose plots by explicitly mapping data to the visual objects that make up the plot.



SPECIALTY PLOTS

24.1 Broken Axes

`brokenaxes` supplies an axes class that can have a visual break to indicate a discontinuous range.



24.2 DeCiDa

DeCiDa is a library of functions and classes for electron device characterization, electronic circuit design and general data visualization and analysis.

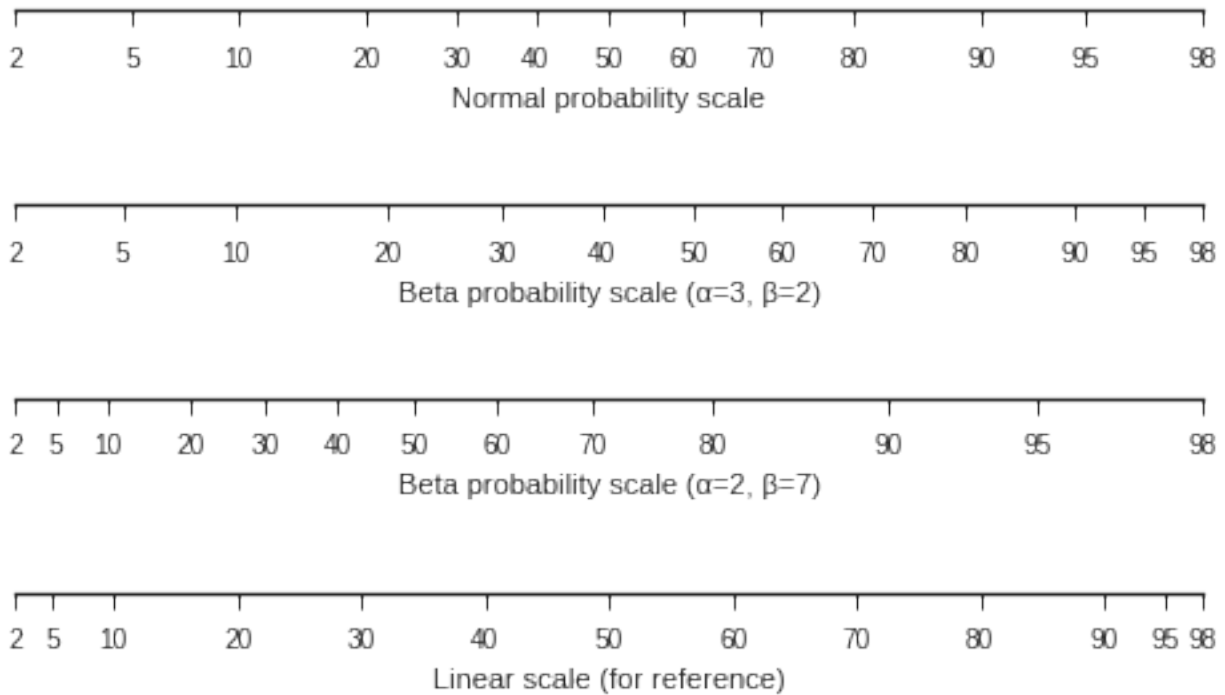
24.3 Matplotlib-Venn

Matplotlib-Venn provides a set of functions for plotting 2- and 3-set area-weighted (or unweighted) Venn diagrams.

24.4 mpl-probscale

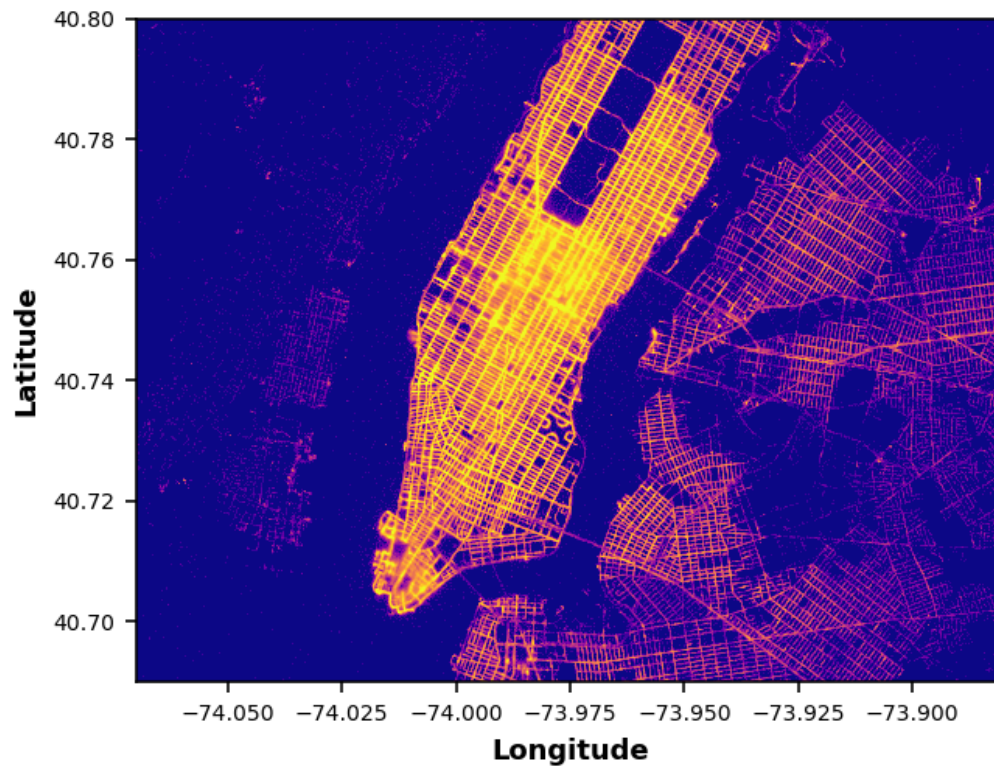
`mpl-probscale` is a small extension that allows Matplotlib users to specify probability scales. Simply importing the `probscale` module registers the scale with Matplotlib, making it accessible via e.g.,

`ax.set_xscale('prob')` or `plt.yscale('prob')`.



24.5 mpl-scatter-density

`mpl-scatter-density` is a small package that makes it easy to make scatter plots of large numbers of points using a density map. The following example contains around 13 million points and the plotting (excluding reading in the data) took less than a second on an average laptop:



When used in interactive mode, the density map is downsampled on-the-fly while panning/zooming in order to provide a smooth interactive experience.

24.6 `mplstereonet`

`mplstereonet` provides stereonets for plotting and analyzing orientation data in Matplotlib.

24.7 `Natgrid`

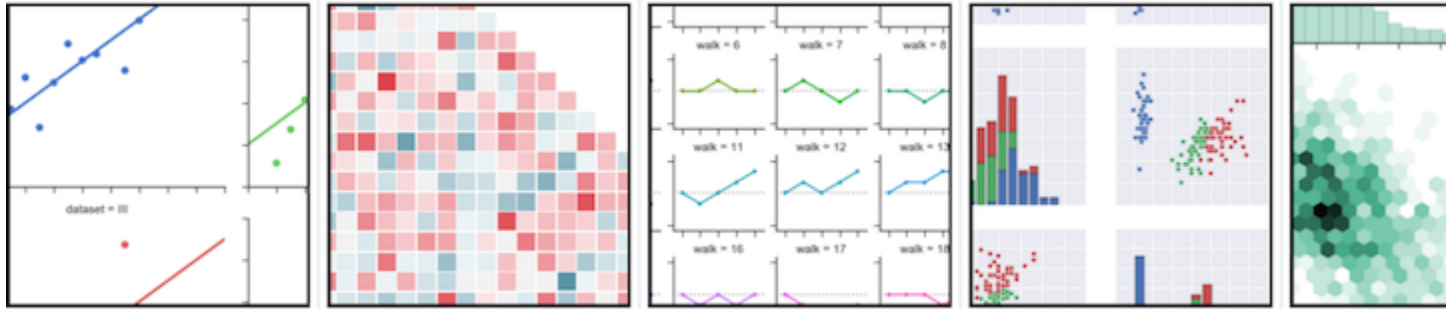
`mpl_toolkits.natgrid` is an interface to the `natgrid` C library for gridding irregularly spaced data.

24.8 `pyUpSet`

`pyUpSet` is a static Python implementation of the `UpSet` suite by Lex et al. to explore complex intersections of sets and data frames.

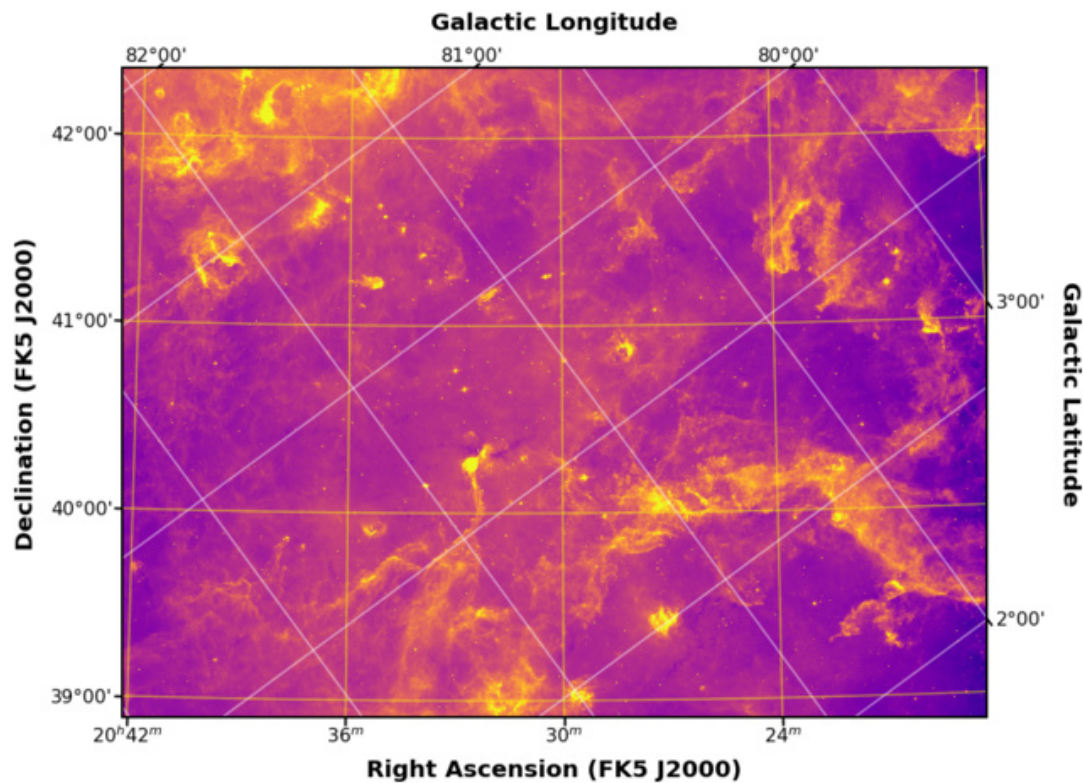
24.9 `seaborn`

`seaborn` is a high level interface for drawing statistical graphics with Matplotlib. It aims to make visualization a central part of exploring and understanding complex datasets.



24.10 WCSAxes

The `Astropy` core package includes a submodule called `WCSAxes` (available at `astropy.visualization.wcsaxes`) which adds Matplotlib projections for Astronomical image data. The following is an example of a plot made with `WCSAxes` which includes the original coordinate system of the image and an overlay of a different coordinate system:



24.11 Windrose

`Windrose` is a Python Matplotlib, Numpy library to manage wind data, draw windroses (also known as polar rose plots), draw probability density functions and fit Weibull distributions.

INTERACTIVITY

25.1 `mplcursors`

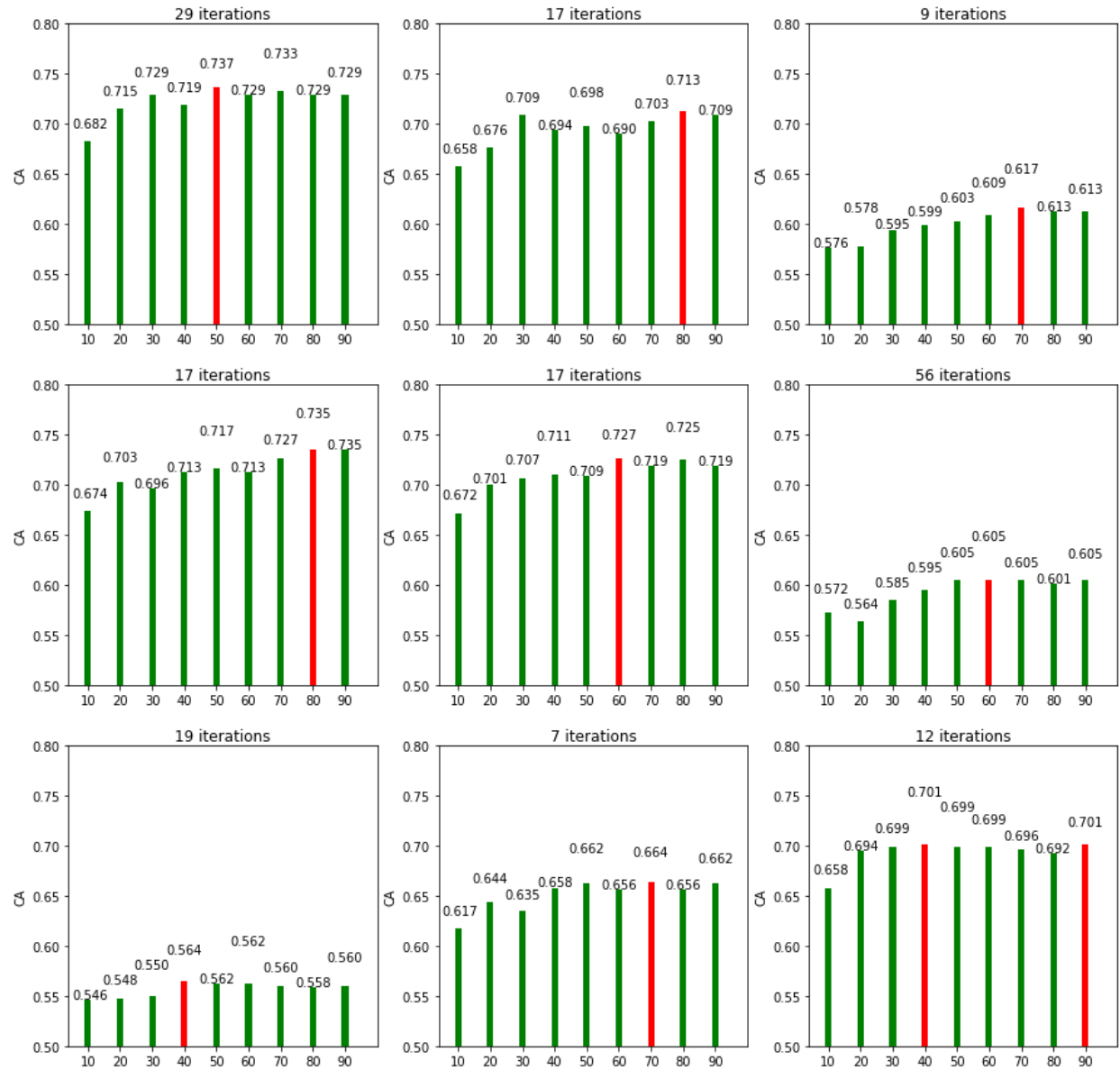
`mplcursors` provides interactive data cursors for Matplotlib.

25.2 `MplDataCursor`

`MplDataCursor` is a toolkit written by Joe Kington to provide interactive "data cursors" (clickable annotation boxes) for Matplotlib.

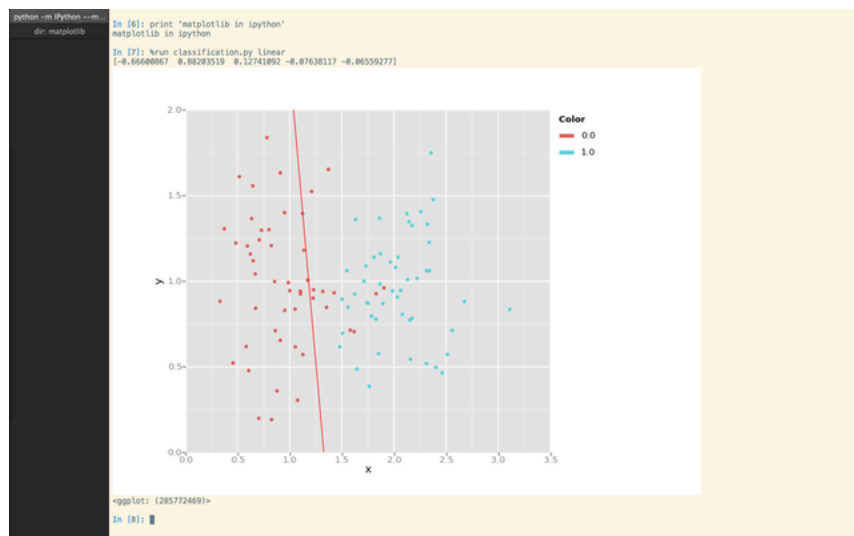
26.1 `adjustText`

`adjustText` is a small library for automatically adjusting text position in Matplotlib plots to minimize overlaps between them, specified points and other objects.



26.2 iTerm2 terminal backend

`matplotlib_itehm2` is an external Matplotlib backend using the iTerm2 nightly build inline image display feature.

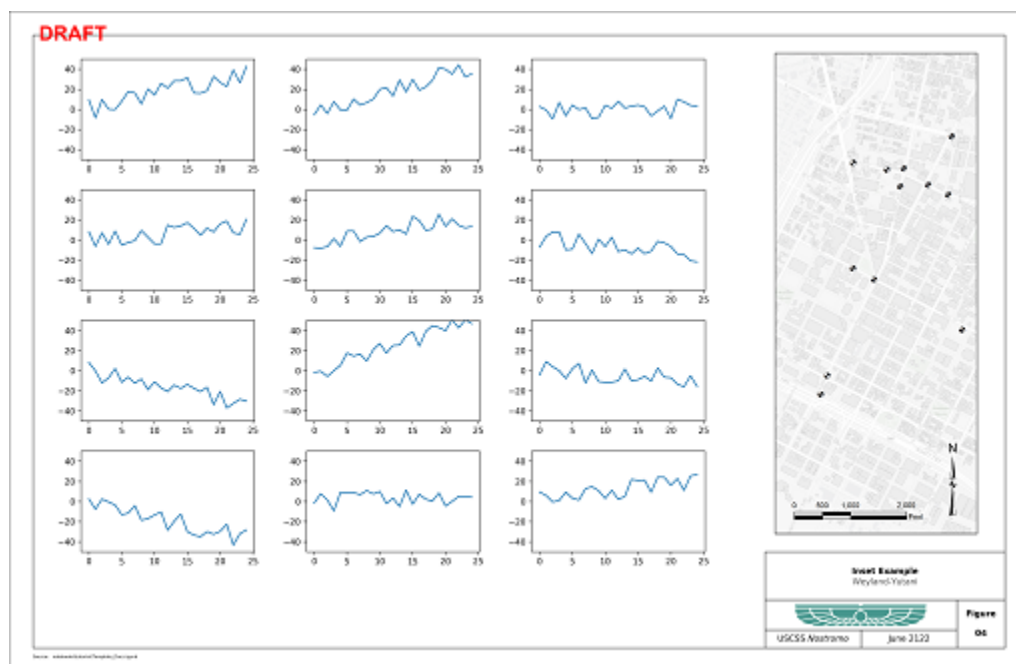


26.3 mplcairo

`mplcairo` is a cairo backend for Matplotlib, with faster and more accurate marker drawing, support for a wider selection of font formats and complex text layout, and various other features.

26.4 mpl-template

`mpl-template` provides a customizable way to add engineering figure elements such as a title block, border, and logo.



Part VI

The Matplotlib API

Below we describe several common approaches to plotting with Matplotlib.

Contents

- The Pyplot API
- The Object-Oriented API
- Colors in Matplotlib

THE PYPLOT API

The `matplotlib.pyplot` module contains functions that allow you to generate many kinds of plots quickly. For examples that showcase the use of the `matplotlib.pyplot` module, see the [Pyplot tutorial](#) or the `pyplots_examples`. We also recommend that you look into the object-oriented approach to plotting, described below.

`matplotlib.pyplot.plotting()`

	Function	Description
<code>acorr</code>		Plot the autocorrelation of <code>x</code> .
<code>angle_spectrum</code>		Plot the angle spectrum.
<code>annotate</code>		Annotate the point <code>xy</code> with text <code>s</code> .
<code>arrow</code>		Add an arrow to the axes.
<code>autoscale</code>		Autoscale the axis view to the data (toggle).
<code>axes</code>		Add an axes to the current figure and make it the current axes.
<code>axhline</code>		Add a horizontal line across the axis.
<code>axhspan</code>		Add a horizontal span (rectangle) across the axis.
<code>axis</code>		Convenience method to get or set axis properties.
<code>axvline</code>		Add a vertical line across the axes.
<code>axvspan</code>		Add a vertical span (rectangle) across the axes.
<code>bar</code>		Make a bar plot.
<code>barbs</code>		Plot a 2-D field of barbs.
<code>barh</code>		Make a horizontal bar plot.
<code>box</code>		Turn the axes box on or off on the current axes.
<code>boxplot</code>		Make a box and whisker plot.
<code>broken_barh</code>		Plot a horizontal sequence of rectangles.
<code>cla</code>		Clear the current axes.
<code>clabel</code>		Label a contour plot.
<code>clf</code>		Clear the current figure.
<code>clim</code>		Set the color limits of the current image.
<code>close</code>		Close a figure window.
<code>cohere</code>		Plot the coherence between <code>x</code> and <code>y</code> .
<code>colorbar</code>		Add a colorbar to a plot.
<code>contour</code>		Plot contours.

Continued on next page

Table 1 – continued from previous page

	Function	Description
	contourf	Plot contours.
	csd	Plot the cross-spectral density.
	delaxes	Remove the given Axes ax from the current figure.
	draw	Redraw the current figure.
	errorbar	Plot y versus x as lines and/or markers with attached errorbars.
	eventplot	Plot identical parallel lines at the given positions.
	figimage	Add a non-resampled image to the figure.
	figlegend	Place a legend in the figure.
	fignum_exists	
	figtext	Add text to figure.
	figure	Creates a new figure.
	fill	Plot filled polygons.
	fill_between	Fill the area between two horizontal curves.
	fill_betweenx	Fill the area between two vertical curves.
	findobj	Find artist objects.
gca		Get the current Axes instance on the current figure matching the given keyword arguments.
gcf		Get a reference to the current figure.
gci		Get the current colorable artist.
get_figlabels		Return a list of existing figure labels.
get_fignums		Return a list of existing figure numbers.
grid		Turn the axes grids on or off.
hexbin		Make a hexagonal binning plot.
hist		Plot a histogram.
hist2d		Make a 2D histogram plot.
hlines		Plot horizontal lines at each y from xmin to xmax.
hold		
imread		Read an image from a file into an array.
imsave		Save an array as in image file.
imshow		Display an image on the axes.
install_repl_displayhook		Install a repl display hook so that any stale figure are automatically redrawn when called.
ioff		Turn interactive mode off.
ion		Turn interactive mode on.
ishold		
isinteractive		Return status of interactive mode.
legend		Places a legend on the axes.
locator_params		Control behavior of tick locators.
loglog		Make a plot with log scaling on both the x and y axis.
magnitude_spectrum		Plot the magnitude spectrum.
margins		Set or retrieve autoscaling margins.
matshow		Display an array as a matrix in a new figure window.
minorticks_off		Remove minor ticks from the current plot.
minorticks_on		Display minor ticks on the current plot.
over		

Continued on next page

Table 1 – continued from previous page

	Function	Description
pause		Pause for interval seconds.
pcolor		Create a pseudocolor plot with a non-regular rectangular grid.
pcolormesh		Create a pseudocolor plot with a non-regular rectangular grid.
phase_spectrum		Plot the phase spectrum.
pie		Plot a pie chart.
plot		Plot y versus x as lines and/or markers.
plot_date		Plot data that contains dates.
plotfile		Plot the data in a file.
polar		Make a polar plot.
psd		Plot the power spectral density.
quiver		Plot a 2-D field of arrows.
quiverkey		Add a key to a quiver plot.
rc		Set the current rc params.
rc_context		Return a context manager for managing rc settings.
rcdefaults		Restore the rc params from Matplotlib's internal defaults.
rgrids		Get or set the radial gridlines on a polar plot.
savefig		Save the current figure.
sca		Set the current Axes instance to ax.
scatter		A scatter plot of y vs x with varying marker size and/or color.
sci		Set the current image.
semilogx		Make a plot with log scaling on the x axis.
semilogy		Make a plot with log scaling on the y axis.
set_cmap		Set the default colormap.
setp		Set a property on an artist object.
show		Display a figure.
specgram		Plot a spectrogram.
spectral		Set the colormap to "spectral".
spy		Plot the sparsity pattern on a 2-D array.
stackplot		Draws a stacked area plot.
stem		Create a stem plot.
step		Make a step plot.
streamplot		Draws streamlines of a vector flow.
subplot		Return a subplot axes at the given grid position.
subplot2grid		Create an axis at specific location inside a regular grid.
subplot_tool		Launch a subplot tool window for a figure.
subplots		Create a figure and a set of subplots This utility wrapper makes it convenient to create
subplots_adjust		Tune the subplot layout.
suptitle		Add a centered title to the figure.
switch_backend		Switch the default backend.
table		Add a table to the current axes.
text		Add text to the axes.
thetagrids		Get or set the theta locations of the gridlines in a polar plot.
tick_params		Change the appearance of ticks, tick labels, and gridlines.

Continued on next page

Table 1 – continued from previous page

	Function	Description
ticklabel_format		Change the ScalarFormatter used by default for linear axes.
tight_layout		Automatically adjust subplot parameters to give specified padding.
title		Set a title of the current axes.
tricontour		Draw contours on an unstructured triangular grid.
tricontourf		Draw contours on an unstructured triangular grid.
tripcolor		Create a pseudocolor plot of an unstructured triangular grid.
triplot		Draw a unstructured triangular grid as lines and/or markers.
twinx		Make a second axes that shares the x-axis.
twiny		Make a second axes that shares the y-axis.
uninstall_repl_displayhook		Uninstalls the matplotlib display hook.
violinplot		Make a violin plot.
vlines		Plot vertical lines.
xcorr		Plot the cross correlation between x and y.
xkcd		Turns on <code>xkcd</code> sketch-style drawing mode.
xlabel		Set the x-axis label of the current axes.
xlim		Get or set the x limits of the current axes.
xscale		Set the scaling of the x-axis.
xticks		Get or set the current tick locations and labels of the x-axis.
ylabel		Set the y-axis label of the current axes.
ylim		Get or set the y-limits of the current axes.
yscale		Set the scaling of the y-axis.
yticks		Get or set the current tick locations and labels of the y-axis.

THE OBJECT-ORIENTED API

Most of these functions also exist as methods in the `matplotlib.axes.Axes` class. You can use them with the "Object Oriented" approach to Matplotlib.

While it is easy to quickly generate plots with the `matplotlib.pyplot` module, we recommend using the object-oriented approach for more control and customization of your plots. See the methods in the `matplotlib.axes.Axes()` class for many of the same plotting functions. For examples of the OO approach to Matplotlib, see the API Examples.

COLORS IN MATPLOTLIB

There are many colormaps you can use to map data onto color values. Below we list several ways in which color can be utilized in Matplotlib.

For a more in-depth look at colormaps, see the [Colormaps in Matplotlib](#) tutorial.

`matplotlib.pyplot.colormaps()`

Matplotlib provides a number of colormaps, and others can be added using `register_cmap()`. This function documents the built-in colormaps, and will also return a list of all registered colormaps if called.

You can set the colormap for an image, `pcolor`, `scatter`, etc, using a keyword argument:

```
imshow(X, cmap=cm.hot)
```

or using the `set_cmap()` function:

```
imshow(X)
pyplot.set_cmap('hot')
pyplot.set_cmap('jet')
```

In interactive mode, `set_cmap()` will update the colormap post-hoc, allowing you to see which one works best for your data.

All built-in colormaps can be reversed by appending `_r`: For instance, `gray_r` is the reverse of `gray`.

There are several common color schemes used in visualization:

Sequential schemes for unipolar data that progresses from low to high

Diverging schemes for bipolar data that emphasizes positive or negative deviations from a central value

Cyclic schemes meant for plotting values that wrap around at the endpoints, such as phase angle, wind direction, or time of day

Qualitative schemes for nominal data that has no inherent ordering, where color is used only to distinguish categories

Matplotlib ships with 4 perceptually uniform color maps which are the recommended color maps for sequential data:

Colormap	Description
inferno	perceptually uniform shades of black-red-yellow
magma	perceptually uniform shades of black-red-white
plasma	perceptually uniform shades of blue-red-yellow
viridis	perceptually uniform shades of blue-green-yellow

The following colormaps are based on the [ColorBrewer](#) color specifications and designs developed by Cynthia Brewer:

ColorBrewer Diverging (luminance is highest at the midpoint, and decreases towards differently-colored endpoints):

Colormap	Description
BrBG	brown, white, blue-green
PiYG	pink, white, yellow-green
PRGn	purple, white, green
PuOr	orange, white, purple
RdBu	red, white, blue
RdGy	red, white, gray
RdYlBu	red, yellow, blue
RdYlGn	red, yellow, green
Spectral	red, orange, yellow, green, blue

ColorBrewer Sequential (luminance decreases monotonically):

Colormap	Description
Blues	white to dark blue
BuGn	white, light blue, dark green
BuPu	white, light blue, dark purple
GnBu	white, light green, dark blue
Greens	white to dark green
Greys	white to black (not linear)
Oranges	white, orange, dark brown
OrRd	white, orange, dark red
PuBu	white, light purple, dark blue
PuBuGn	white, light purple, dark green
PuRd	white, light purple, dark red
Purples	white to dark purple
RdPu	white, pink, dark purple
Reds	white to dark red
YlGn	light yellow, dark green
YlGnBu	light yellow, light green, dark blue
YlOrBr	light yellow, orange, dark brown
YlOrRd	light yellow, orange, dark red

ColorBrewer Qualitative:

(For plotting nominal data, ListedColormap is used, not LinearSegmentedColormap. Different sets of colors are recommended for different numbers of categories.)

- Accent
- Dark2
- Paired
- Pastel1
- Pastel2
- Set1
- Set2
- Set3

A set of colormaps derived from those of the same name provided with Matlab are also included:

Colormap	Description
autumn	sequential linearly-increasing shades of red-orange-yellow
bone	sequential increasing black-white color map with a tinge of blue, to emulate X-ray film
cool	linearly-decreasing shades of cyan-magenta
copper	sequential increasing shades of black-copper
flag	repetitive red-white-blue-black pattern (not cyclic at endpoints)
gray	sequential linearly-increasing black-to-white grayscale
hot	sequential black-red-yellow-white, to emulate blackbody radiation from an object at increasing temperatures
hsv	cyclic red-yellow-green-cyan-blue-magenta-red, formed by changing the hue component in the HSV color space
jet	a spectral map with dark endpoints, blue-cyan-yellow-red; based on a fluid-jet simulation by NCSA ¹
pink	sequential increasing pastel black-pink-white, meant for sepia tone colorization of photographs
prism	repetitive red-yellow-green-blue-purple-...-green pattern (not cyclic at endpoints)
spring	linearly-increasing shades of magenta-yellow
summer	sequential linearly-increasing shades of green-yellow
winter	linearly-increasing shades of blue-green

¹ Rainbow colormaps, jet in particular, are considered a poor choice for scientific visualization by many researchers: [Rainbow Color Map \(Still\) Considered Harmful](#)

A set of palettes from the [Yorick scientific visualisation package](#), an evolution of the GIST package, both by David H. Munro are included:

Colormap	Description
<code>gist_earth</code>	mapmaker's colors from dark blue deep ocean to green lowlands to brown highlands to white mountains
<code>gist_heat</code>	sequential increasing black-red-orange-white, to emulate blackbody radiation from an iron bar as it grows hotter
<code>gist_ncar</code>	pseudo-spectral black-blue-green-yellow-red-purple-white colormap from National Center for Atmospheric Research ²
<code>gist_rainbow</code>	goes through the colors in spectral order from red to violet at full saturation (like hsv but not cyclic)
<code>gist_sern</code>	"Stern special" color table from Interactive Data Language software

Other miscellaneous schemes:

² Resembles "BkBlAqGrYeOrReViWh200" from NCAR Command Language. See [Color Table Gallery](#)

Colormap	Description
afmhot	sequential black-orange-yellow-white blackbody spectrum, commonly used in atomic force microscopy
brg	blue-red-green
bwr	diverging blue-white-red
coolwarm	diverging blue-gray-red, meant to avoid issues with 3D shading, color blindness, and ordering of colors ³
CMRmap	"Default colormaps on color images often reproduce to confusing grayscale images. The proposed colormap maintains an aesthetically pleasing color image that automatically reproduces to a monotonic grayscale with discrete, quantifiable saturation levels." ⁴
cubehelix	Unlike most other color schemes cubehelix was designed by D.A. Green to be monotonically increasing in terms of perceived brightness. Also, when printed on a black and white postscript printer, the scheme results in a greyscale with monotonically increasing brightness. This color scheme is named cubehelix because the r,g,b values produced can be visualised as a squashed helix around the diagonal in the r,g,b color cube.
gnuplot	gnuplot's traditional pm3d scheme (black-blue-red-yellow)
gnuplot2	sequential color printable as gray (black-blue-violet-yellow-white)
ocean	green-blue-white
rainbow	spectral purple-blue-green-yellow-orange-red colormap with diverging luminance
seismic	diverging blue-white-red
nipy_spectral	black-purple-blue-green-yellow-red-white spectrum, originally from the Neuroimaging in Python project
terrain	mapmaker's colors, blue-green-yellow-brown-white, originally from IGOR Pro

The following colormaps are redundant and may be removed in future versions. It's recommended to use the names in the descriptions instead, which produce identical output:

Colormap	Description
gist_gray	identical to gray
gist_yarg	identical to gray_r
binary	identical to gray_r
spectral	identical to nipy_spectral ⁵

³ See [Diverging Color Maps for Scientific Visualization](#) by Kenneth Moreland.

⁴ See [A Color Map for Effective Black-and-White Rendering of Color-Scale Images](#) by Carey Rappaport

⁵ Changed to distinguish from ColorBrewer's Spectral map. `spectral()` still works, but `set_cmap('nipy_spectral')` is recommended for clarity.

API CHANGES

Log of changes to Matplotlib that affect the outward-facing API. If updating Matplotlib breaks your scripts, this list may help you figure out what caused the breakage and how to fix it by updating your code.

For new features that were added to Matplotlib, please see [What's new in Matplotlib](#).

30.1 API Changes in 2.2.0

30.1.1 New dependency

[kiwisolver](#) is now a required dependency to support the new `constrained_layout`, see [Constrained Layout Guide](#) for more details.

30.1.2 Deprecations

Classes, functions, and methods

The unused and untested `Artist.onRemove` and `Artist.hitlist` methods have been deprecated.

The now unused `mlab.less_simple_linear_interpolation` function is deprecated.

The unused `ContourLabeler.get_real_label_width` method is deprecated.

The unused `FigureManagerBase.show_popup` method is deprecated. This introduced in e945059b327d42a99938b939a1be867fa023e7ba in 2005 but never built out into any of the backends.

`backend_tkagg.AxisMenu` is deprecated, as it has become unused since the removal of "classic" toolbars.

Changed function signatures

kwarg `fig` to `GridSpec.get_subplot_params` is deprecated, use `figure` instead.

Using `pyplot.axes` with an `Axes` as argument is deprecated. This sets the current axes, i.e. it has the same effect as `pyplot.sca`. For clarity `plt.sca(ax)` should be preferred over `plt.axes(ax)`.

Using strings instead of booleans to control grid and tick visibility is deprecated. Using "on", "off", "true", or "false" to control grid and tick visibility has been deprecated. Instead, use normal booleans (True/False) or boolean-likes. In the future, all non-empty strings may be interpreted as True.

When given 2D inputs with non-matching numbers of columns, plot currently cycles through the columns of the narrower input, until all the columns of the wider input have been plotted. This behavior is deprecated; in the future, only broadcasting (1 column to n columns) will be performed.

rcparams

The `rcParams["backend.qt4"]` and `rcParams["backend.qt5"]` `rcParams` were deprecated in version 2.2. In order to force the use of a specific Qt binding, either import that binding first, or set the `QT_API` environment variable.

Deprecation of the `nbagg.transparent` `rcParam`. To control transparency of figure patches in the `nbagg` (or any other) backend, directly set `figure.patch.facecolor`, or the `figure.facecolor` `rcParam`.

Deprecated `Axis.unit_data`

Use `Axis.units` (which has long existed) instead.

30.1.3 Removals

Function Signatures

Contouring no longer supports legacy corner masking. The deprecated `ContourSet.vmin` and `ContourSet.vmax` properties have been removed.

Passing `None` instead of "none" as format to `errorbar` is no longer supported.

The `bgcolor` keyword argument to `Axes` has been removed.

Modules, methods, and functions

The `matplotlib.finance`, `mpl_toolkits.exceltools` and `mpl_toolkits.gtktools` modules have been removed. `matplotlib.finance` remains available at https://github.com/matplotlib/mpl_finance.

The `mpl_toolkits.mplot3d.art3d.iscolor` function has been removed.

The `Axes.get_axis_bgcolor`, `Axes.set_axis_bgcolor`, `Bbox.update_from_data`, `Bbox.update_datalim_numerix`, `MaxNLocator.bin_boundaries` methods have been removed.

`mencoder` can no longer be used to encode animations.

The unused `FONT_SCALE` and `fontd` attributes of the `RendererSVG` class have been removed.

color maps

The spectral colormap has been removed. The Vega* colormaps, which were aliases for the tab* colormaps, have been removed.

rcparams

The following deprecated rcParams have been removed:

- `axes.color_cycle` (see `axes.prop_cycle`),
- `legend.isaxes`,
- `svg.embed_char_paths` (see `svg.fonttype`),
- `text.fontstyle`, `text.fontangle`, `text.fontvariant`, `text.fontweight`, `text.fontsize` (renamed to `text.style`, etc.),
- `tick.size` (renamed to `tick.major.size`).

30.1.4 Only accept string-like for Categorical input

Do not accept mixed string / float / int input, only strings are valid categoricals.

30.1.5 Removal of unused imports

Many unused imports were removed from the codebase. As a result, trying to import certain classes or functions from the "wrong" module (e.g. `Figure` from `matplotlib.backends.backend_agg` instead of `matplotlib.figure`) will now raise an `ImportError`.

30.1.6 Axes3D.get_xlim, get_ylim and get_zlim now return a tuple

They previously returned an array. Returning a tuple is consistent with the behavior for 2D axes.

30.1.7 Exception type changes

If `MovieWriterRegistry` can't find the requested `MovieWriter`, a more helpful `RuntimeError` message is now raised instead of the previously raised `KeyError`.

`auto_adjust_subplots` now raises `ValueError` instead of `RuntimeError` when sizes of input lists don't match

30.1.8 Figure.set_figwidth and Figure.set_figheight default forward to True

`matplotlib.figure.Figure.set_figwidth` and `matplotlib.figure.Figure.set_figheight` had the kwarg `forward=False` by default, but `Figure.set_size_inches` now defaults to `forward=True`. This makes these functions consistent.

30.1.9 Do not truncate svg sizes to nearest point

There is no reason to size the SVG out put in integer points, change to out putting floats for the height, width, and viewBox attributes of the svg element.

30.1.10 Fontsizes less than 1 pt are clipped to be 1 pt.

FreeType doesn't allow fonts to get smaller than 1 pt, so all Agg backends were silently rounding up to 1 pt. PDF (other vector backends?) were letting us write fonts that were less than 1 pt, but they could not be placed properly because position information comes from FreeType. This change makes it so no backends can use fonts smaller than 1 pt, consistent with FreeType and ensuring more consistent results across backends.

30.1.11 Changes to Qt backend class MRO

To support both Agg and cairo rendering for Qt backends all of the non-Agg specific code previously in `backend_qt5agg.FigureCanvasQTAggBase` has been moved to `backend_qt5.FigureCanvasQT` so it can be shared with the cairo implementation. The `FigureCanvasQTAggBase.paintEvent()`, `FigureCanvasQTAggBase.blit()`, and `FigureCanvasQTAggBase.print_figure()` methods have moved to `FigureCanvasQTAgg.paintEvent()`, `FigureCanvasQTAgg.blit()`, and `FigureCanvasQTAgg.print_figure()`. The first two methods assume that the instance is also a `QWidget` so to use `FigureCanvasQTAggBase` it was required to multiple inherit from a `QWidget` sub-class.

Having moved all of its methods either up or down the class hierarchy `FigureCanvasQTAggBase` has been deprecated. To do this with out warning and to preserve as much API as possible, `backend_qt5.FigureCanvasQTAggBase` now inherits from `backend_qt5.FigureCanvasQTAgg`.

The MRO for `FigureCanvasQTAgg` and `FigureCanvasQTAggBase` used to be

```
[matplotlib.backends.backend_qt5agg.FigureCanvasQTAgg,
matplotlib.backends.backend_qt5agg.FigureCanvasQTAggBase,
matplotlib.backends.backend_agg.FigureCanvasAgg,
matplotlib.backends.backend_qt5.FigureCanvasQT,
PyQt5.QtWidgets.QWidget,
PyQt5.QtCore.QObject,
sip.wrapper,
PyQt5.QtGui.QPaintDevice,
sip.simplewrapper,
matplotlib.backend_bases.FigureCanvasBase,
object]
```

and

```
[matplotlib.backends.backend_qt5agg.FigureCanvasQTAggBase,
matplotlib.backends.backend_agg.FigureCanvasAgg,
matplotlib.backend_bases.FigureCanvasBase,
object]
```

respectively. They are now

```
[matplotlib.backends.backend_qt5agg.FigureCanvasQTAgg,
matplotlib.backends.backend_agg.FigureCanvasAgg,
matplotlib.backends.backend_qt5.FigureCanvasQT,
PyQt5.QtWidgets.QWidget,
PyQt5.QtCore.QObject,
sip.wrapper,
PyQt5.QtGui.QPaintDevice,
sip.simplewrapper,
matplotlib.backend_bases.FigureCanvasBase,
object]
```

and

```
[matplotlib.backends.backend_qt5agg.FigureCanvasQTAggBase,
matplotlib.backends.backend_qt5agg.FigureCanvasQTAgg,
matplotlib.backends.backend_agg.FigureCanvasAgg,
matplotlib.backends.backend_qt5.FigureCanvasQT,
PyQt5.QtWidgets.QWidget,
PyQt5.QtCore.QObject,
sip.wrapper,
PyQt5.QtGui.QPaintDevice,
sip.simplewrapper,
matplotlib.backend_bases.FigureCanvasBase,
object]
```

30.1.12 Axes.imshow clips RGB values to the valid range

When `Axes.imshow` is passed an RGB or RGBA value with out-of-range values, it now logs a warning and clips them to the valid range. The old behaviour, wrapping back in to the range, often hid outliers and made interpreting RGB images unreliable.

30.1.13 GTKAgg and GTKCairo backends deprecated

The `GTKAgg` and `GTKCairo` backends have been deprecated. These obsolete backends allow figures to be rendered via the GTK+ 2 toolkit. They are untested, known to be broken, will not work with Python 3, and their use has been discouraged for some time. Instead, use the `GTK3Agg` and `GTK3Cairo` backends for rendering to GTK+ 3 windows.

30.2 API Changes in 2.1.2

30.2.1 Figure.legend no longer checks for repeated lines to ignore

matplotlib.figure.Figure.legend used to check if a line had the same label as an existing legend entry. If it also had the same line color or marker color legend didn't add a new entry for that line. However, the list of conditions was incomplete, didn't handle RGB tuples, didn't handle linewidths or linestyle etc.

This logic did not exist in Axes.legend. It was included (erroneously) in Matplotlib 2.1.1 when the legend argument parsing was unified [#9324](<https://github.com/matplotlib/matplotlib/pull/9324>). This change removes that check in Axes.legend again to restore the old behavior.

This logic has also been dropped from Figure.legend, where it was previously undocumented. Repeated lines with the same label will now each have an entry in the legend. If you do not want the duplicate entries, don't add a label to the line, or prepend the label with an underscore.

30.3 API Changes in 2.1.1

30.3.1 Default behavior of log scales reverted to clip ≤ 0 values

The change in 2.1.0 to mask in logscale by default had more disruptive changes than anticipated and has been reverted, however the clipping is now done in a way that fixes the issues that motivated changing the default behavior to 'mask'.

As a side effect of this change, error bars which go negative now work as expected on log scales.

30.4 API Changes in 2.1.0

30.4.1 Default behavior of log scales changed to mask ≤ 0 values

Calling matplotlib.axes.Axes.set_xscale or matplotlib.axes.Axes.set_yscale now uses 'mask' as the default method to handle invalid values (as opposed to 'clip'). This means that any values ≤ 0 on a log scale will not be shown.

Previously they were clipped to a very small number and shown.

30.4.2 matplotlib.cbook.CallbackRegistry.process() suppresses exceptions by default

Matplotlib uses instances of CallbackRegistry as a bridge between user input event from the GUI and user callbacks. Previously, any exceptions raised in a user call back would bubble out of the process method, which is typically in the GUI event loop. Most GUI frameworks simply print the traceback to the screen and continue as there is not always a clear method of getting the exception back to the user. However PyQt5 now exits the process when it receives an un-handled python

exception in the event loop. Thus, `process()` now suppresses and prints tracebacks to `stderr` by default.

What `process()` does with exceptions is now user configurable via the `exception_handler` attribute and kwarg. To restore the previous behavior pass `None`

```
cb = CallbackRegistry(exception_handler=None)
```

A function which take and `Exception` as its only argument may also be passed

```
def maybe_reraise(exc):
    if isinstance(exc, RuntimeError):
        pass
    else:
        raise exc

cb = CallbackRegistry(exception_handler=maybe_reraise)
```

30.4.3 Improved toggling of the axes grids

The `g` key binding now switches the states of the `x` and `y` grids independently (by cycling through all four on/off combinations).

The new `G` key binding switches the states of the minor grids.

Both bindings are disabled if only a subset of the grid lines (in either direction) is visible, to avoid making irreversible changes to the figure.

30.4.4 Ticklabels are turned off instead of being invisible

Internally, the `Tick`'s `label1On()` attribute is now used to hide tick labels instead of setting the visibility on the tick label objects. This improves overall performance and fixes some issues. As a consequence, in case those labels ought to be shown, `tick_params()` needs to be used, e.g.

```
ax.tick_params(labelbottom=True)
```

30.4.5 Removal of warning on empty legends

`plt.legend` used to issue a warning when no labeled artist could be found. This warning has been removed.

30.4.6 More accurate legend autopositioning

Automatic positioning of legends now prefers using the area surrounded by a `Line2D` rather than placing the legend over the line itself.

30.4.7 Cleanup of stock sample data

The sample data of stocks has been cleaned up to remove redundancies and increase portability. The AAPL.dat.gz, INTC.dat.gz and aapl.csv files have been removed entirely and will also no longer be available from `matplotlib.cbook.get_sample_data`. If a CSV file is required, we suggest using the `msft.csv` that continues to be shipped in the sample data. If a NumPy binary file is acceptable, we suggest using one of the following two new files. The `aapl.npy.gz` and `goog.npy` files have been replaced by `aapl.npz` and `goog.npz`, wherein the first column's type has changed from `datetime.date` to `np.datetime64` for better portability across Python versions. Note that Matplotlib does not fully support `np.datetime64` as yet.

30.4.8 Updated qhull to 2015.2

The version of qhull shipped with Matplotlib, which is used for Delaunay triangulation, has been updated from version 2012.1 to 2015.2.

30.4.9 Improved Delaunay triangulations with large offsets

Delaunay triangulations now deal with large x,y offsets in a better way. This can cause minor changes to any triangulations calculated using Matplotlib, i.e. any use of `matplotlib.tri.Triangulation` that requests that a Delaunay triangulation is calculated, which includes `matplotlib.pyplot.tricontour`, `matplotlib.pyplot.tricontourf`, `matplotlib.pyplot.tripcolor`, `matplotlib.pyplot.triplot`, `matplotlib.mlab.griddata` and `mpl_toolkits.mplot3d.axes3d.Axes3D.plot_trisurf`.

30.4.10 Use `backports.functools_lru_cache` instead of `functools32`

It's better maintained and more widely used (by `pylint`, `jaraco`, etc).

30.4.11 `cbook.is_numlike` only performs an instance check

`is_numlike()` now only checks that its argument is an instance of `(numbers.Number, np.Number)`. In particular, this means that arrays are now not num-like.

30.4.12 Elliptical arcs now drawn between correct angles

The `matplotlib.patches.Arc` patch is now correctly drawn between the given angles.

Previously a circular arc was drawn and then stretched into an ellipse, so the resulting arc did not lie between `theta1` and `theta2`.

30.4.13 `-d$backend` no longer sets the backend

It is no longer possible to set the backend by passing `-d$backend` at the command line. Use the `MPLBACKEND` environment variable instead.

30.4.14 Path.intersects_bbox always treats the bounding box as filled

Previously, when `Path.intersects_bbox` was called with `filled` set to `False`, it would treat both the path and the bounding box as unfilled. This behavior was not well documented and it is usually not the desired behavior, since bounding boxes are used to represent more complex shapes located inside the bounding box. This behavior has now been changed: when `filled` is `False`, the path will be treated as unfilled, but the bounding box is still treated as filled. The old behavior was arguably an implementation bug.

When `Path.intersects_bbox` is called with `filled` set to `True` (the default value), there is no change in behavior. For those rare cases where `Path.intersects_bbox` was called with `filled` set to `False` and where the old behavior is actually desired, the suggested workaround is to call `Path.intersects_path` with a rectangle as the path:

```
from matplotlib.path import Path
from matplotlib.transforms import Bbox, BboxTransformTo
rect = Path.unit_rectangle().transformed(BboxTransformTo(bbox))
result = path.intersects_path(rect, filled=False)
```

30.4.15 WX no longer calls generates IdleEvent events or calls idle_event

Removed unused private method `__onIdle` from `FigureCanvasWx`.

The `IdleEvent` class and `FigureCanvasBase.idle_event` method will be removed in 2.2

30.4.16 Correct scaling of `magnitude_spectrum()`

The functions `matplotlib.mlab.magnitude_spectrum()` and `matplotlib.pyplot.magnitude_spectrum()` implicitly assumed the sum of windowing function values to be one. In Matplotlib and Numpy the standard windowing functions are scaled to have maximum value of one, which usually results in a sum of the order of $n/2$ for a n -point signal. Thus the amplitude scaling `magnitude_spectrum()` was off by that amount when using standard windowing functions (Bug 8417). Now the behavior is consistent with `matplotlib.pyplot.psd()` and `scipy.signal.welch()`. The following example demonstrates the new and old scaling:

```
import matplotlib.pyplot as plt
import numpy as np

tau, n = 10, 1024 # 10 second signal with 1024 points
T = tau/n # sampling interval
t = np.arange(n)*T

a = 4 # amplitude
x = a*np.sin(40*np.pi*t) # 20 Hz sine with amplitude a

# New correct behavior: Amplitude at 20 Hz is a/2
```

(continues on next page)

(continued from previous page)

```
plt.magnitude_spectrum(x, Fs=1/T, sides='onesided', scale='linear')

# Original behavior: Amplitude at 20 Hz is (a/2)*(n/2) for a Hanning window
w = np.hanning(n) # default window is a Hanning window
plt.magnitude_spectrum(x*np.sum(w), Fs=1/T, sides='onesided', scale='linear')
```

30.4.17 Change to signatures of `bar()` & `barh()`

For 2.0 the default value of `*align*` changed to `'center'`. However this caused the signature of `bar()` and `barh()` to be misleading as the first parameters were still left and bottom respectively:

```
bar(left, height, *, align='center', **kwargs)
barh(bottom, width, *, align='center', **kwargs)
```

despite behaving as the center in both cases. The methods now take `*args`, `**kwargs` as input and are documented to have the primary signatures of:

```
bar(x, height, *, align='center', **kwargs)
barh(y, width, *, align='center', **kwargs)
```

Passing left and bottom as keyword arguments to `bar()` and `barh()` respectively will warn. Support will be removed in Matplotlib 3.0.

30.4.18 Font cache as json

The font cache is now saved as json, rather than a pickle.

30.4.19 Invalid (Non-finite) Axis Limit Error

When using `set_xlim()` and `set_ylim()`, passing non-finite values now results in a `ValueError`. The previous behavior resulted in the limits being erroneously reset to `(-0.001, 0.001)`.

30.4.20 `scatter` and Collection offsets are no longer implicitly flattened

Collection (and thus both 2D scatter and 3D scatter) no longer implicitly flattens its offsets. As a consequence, `scatter`'s `x` and `y` arguments can no longer be 2+-dimensional arrays.

30.4.21 Deprecations

`GraphicsContextBase`'s `linestyle` property.

The `GraphicsContextBase.get_linestyle` and `GraphicsContextBase.set_linestyle` methods, which had no effect, have been deprecated. All of the backends Matplotlib ships use `GraphicsContextBase`.

`get_dashes` and `GraphicsContextBase.set_dashes` which are more general. Third-party backends should also migrate to the `*_dashes` methods.

`NavigationToolbar2.dynamic_update`

Use `draw_idle()` method on the Canvas instance instead.

Testing

`matplotlib.testing.noseclasses` is deprecated and will be removed in 2.3

`EngFormatter` num arg as string

Passing a string as num argument when calling an instance of `matplotlib.ticker.EngFormatter` is deprecated and will be removed in 2.3.

`mpl_toolkits.axes_grid` module

All functionally from `mpl_toolkits.axes_grid` can be found in either `mpl_toolkits.axes_grid1` or `mpl_toolkits.axisartist`. Axes classes from `mpl_toolkits.axes_grid` based on `Axis` from `mpl_toolkits.axisartist` can be found in `mpl_toolkits.axisartist`.

Axes collision in `Figure.add_axes`

Adding an axes instance to a figure by using the same arguments as for a previous axes instance currently reuses the earlier instance. This behavior has been deprecated in Matplotlib 2.1. In a future version, a new instance will always be created and returned. Meanwhile, in such a situation, a deprecation warning is raised by `AxesStack`.

This warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance. See the docstring of `add_axes()` for more information.

Additional details on the rationale behind this deprecation can be found in [#7377](#) and [#9024](#).

Former validators for `contour.negative_linestyle`

The former public validation functions `validate_negative_linestyle` and `validate_negative_linestyle_legacy` will be deprecated in 2.1 and may be removed in 2.3. There are no public functions to replace them.

cbook

Many unused or near-unused matplotlib.cbook functions and classes have been deprecated: `converter`, `tostr`, `todatetime`, `today`, `tofloat`, `toint`, `unique`, `is_string_like`, `is_sequence_of_strings`, `is_scalar`, `Sorter`, `Xlator`, `soundex`, `Null`, `dict_delall`, `RingBuffer`, `get_split_ind`, `wrap`, `get_recursive_filelist`, `pieces`, `exception_to_str`, `allequal`, `alltrue`, `onetrue`, `allpairs`, `finddir`, `reverse_dict`, `restrict_dict`, `issubclass_safe`, `recursive_remove`, `unmasked_index_ranges`.

30.4.22 Code Removal

qt4_compat.py

Moved to `qt_compat.py`. Renamed because it now handles Qt5 as well.

Previously Deprecated methods

The `GraphicsContextBase.set_graylevel`, `FigureCanvasBase.onHilite` and `mpl_toolkits.axes_grid1.mpl_axes.Axes.toggle_axisline` methods have been removed.

The `ArtistInspector.findobj` method, which was never working due to the lack of a `get_children` method, has been removed.

The deprecated `point_in_path`, `get_path_extents`, `point_in_path_collection`, `path_intersects_path`, `convert_path_to_polygons`, `cleanup_path` and `clip_path_to_rect` functions in the `matplotlib.path` module have been removed. Their functionality remains exposed as methods on the `Path` class.

The deprecated `Artist.get_axes` and `Artist.set_axes` methods have been removed

The `matplotlib.backends.backend_ps.seq_allequal` function has been removed. Use `np.array_equal` instead.

The deprecated `matplotlib.rcsetup.validate_maskedarray`, `matplotlib.rcsetup.deprecate_savefig_extension` and `matplotlib.rcsetup.validate_tkpythoninspect` functions, and associated `savefig.extension` and `tk.pythoninspect.rcparams` entries have been removed.

The kwarg resolution of `matplotlib.projections.polar.PolarAxes` has been removed. It has deprecation with no effect from version 0.98.x.

`Axes.set_aspect("normal")`

Support for setting an Axes's aspect to "normal" has been removed, in favor of the synonym "auto".

shading kwarg to pcolor

The shading kwarg to `pcolor` has been removed. Set `edgecolors` appropriately instead.

Functions removed from the lines module

The `matplotlib.lines` module no longer imports the `pts_to_prestep`, `pts_to_midstep` and `pts_to_poststep` functions from `matplotlib.cbook`.

PDF backend functions

The methods `embedTeXFont` and `tex_font_mapping` of `matplotlib.backend_pdf.PdfFile` have been removed. It is unlikely that external users would have called these methods, which are related to the font system internal to the PDF backend.

`matplotlib.delaunay`

Remove the `delaunay` triangulation code which is now handled by `Qhull` via `matplotlib.tri`.

30.5 API Changes in 2.0.1

30.5.1 Extensions to `matplotlib.backend_bases.GraphicsContextBase`

To better support controlling the color of hatches, the method `matplotlib.backend_bases.GraphicsContextBase.set_hatch_color` was added to the expected API of `GraphicsContext` classes. Calls to this method are currently wrapped with a `try:...except AttributeError` block to preserve back-compatibility with any third-party backends which do not extend `GraphicsContextBase`.

This value can be accessed in the backends via `matplotlib.backend_bases.GraphicsContextBase.get_hatch_color` (which was added in 2.0 see Extension to `matplotlib.backend_bases.GraphicsContextBase`) and should be used to color the hatches.

In the future there may also be `hatch_linewidth` and `hatch_density` related methods added. It is encouraged, but not required that third-party backends extend `GraphicsContextBase` to make adapting to these changes easier.

30.5.2 `afm.get_fontconfig_fonts` returns a list of paths and does not check for existence

`afm.get_fontconfig_fonts` used to return a set of paths encoded as a `{key: 1, ...}` dict, and checked for the existence of the paths. It now returns a list and dropped the existence check, as the same check is performed by the caller (`afm.findSystemFonts`) as well.

30.5.3 `bar` now returns rectangles of negative height or width if the corresponding input is negative

`plt.bar` used to normalize the coordinates of the rectangles that it created, to keep their height and width positives, even if the corresponding input was negative. This normalization has been removed to permit a simpler computation of the correct `sticky_edges` to use.

30.5.4 Do not clip line width when scaling dashes

The algorithm to scale dashes was changed to no longer clip the scaling factor: the dash patterns now continue to shrink at thin line widths. If the line width is smaller than the effective pixel size, this may result in dashed lines turning into solid gray-ish lines. This also required slightly tweaking the default patterns for '-', ':', and '-.' so that with the default line width the final patterns would not change.

There is no way to restore the old behavior.

30.5.5 Deprecate 'Vega' color maps

The "Vega" colormaps are deprecated in Matplotlib 2.0.1 and will be removed in Matplotlib 2.2. Use the "tab" colormaps instead: "tab10", "tab20", "tab20b", "tab20c".

30.6 API Changes in 2.0.0

30.6.1 Deprecation and removal

Color of Axes

The `axisbg` and `axis_bgcolor` properties on Axes have been deprecated in favor of `facecolor`.

GTK and GDK backends deprecated

The GDK and GTK backends have been deprecated. These obsolete backends allow figures to be rendered via the GDK API to files and GTK2 figures. They are untested and known to be broken, and their use has been discouraged for some time. Instead, use the `GTKAgg` and `GTKCairo` backends for rendering to GTK2 windows.

WX backend deprecated

The WX backend has been deprecated. It is untested, and its use has been discouraged for some time. Instead, use the `WXAgg` backend for rendering figures to WX windows.

CocoaAgg backend removed

The deprecated and not fully functional `CocoaAgg` backend has been removed.

round removed from TkAgg Backend

The `TkAgg` backend had its own implementation of the `round` function. This was unused internally and has been removed. Instead, use either the `round` builtin function or `numpy.round`.

'hold' functionality deprecated

The 'hold' keyword argument and all functions and methods related to it are deprecated, along with the 'axes.hold' rcParams entry. The behavior will remain consistent with the default hold=True state that has long been in place. Instead of using a function or keyword argument (hold=False) to change that behavior, explicitly clear the axes or figure as needed prior to subsequent plotting commands.

30.6.2 Artist.update has return value

The methods matplotlib.artist.Artist.set, matplotlib.Artist.update, and the function matplotlib.artist.setp now use a common codepath to look up how to update the given artist properties (either using the setter methods or an attribute/property).

The behavior of matplotlib.Artist.update is slightly changed to return a list of the values returned from the setter methods to avoid changing the API of matplotlib.Artist.set and matplotlib.artist.setp.

The keys passed into matplotlib.Artist.update are now converted to lower case before being processed, to match the behavior of matplotlib.Artist.set and matplotlib.artist.setp. This should not break any user code because there are no set methods with capitals in their names, but this puts a constraint on naming properties in the future.

30.6.3 Legend initializers gain edgecolor and facecolor kwargs

The Legend background patch (or 'frame') can have its edgecolor and facecolor determined by the corresponding keyword arguments to the matplotlib.legend.Legend initializer, or to any of the methods or functions that call that initializer. If left to their default values of None, their values will be taken from matplotlib.rcParams. The previously-existing framealpha kwarg still controls the alpha transparency of the patch.

30.6.4 Qualitative colormaps

Colorbrewer's qualitative/discrete colormaps ("Accent", "Dark2", "Paired", "Pastel1", "Pastel2", "Set1", "Set2", "Set3") are now implemented as ListedColormap instead of LinearSegmentedColormap.

To use these for images where categories are specified as integers, for instance, use:

```
plt.imshow(x, cmap='Dark2', norm=colors.NoNorm())
```

30.6.5 Change in the draw_image backend API

The draw_image method implemented by backends has changed its interface.

This change is only relevant if the backend declares that it is able to transform images by returning True from option_scale_image. See the draw_image docstring for more information.

30.6.6 matplotlib.ticker.LinearLocator algorithm update

The `matplotlib.ticker.LinearLocator` is used to define the range and location of axis ticks when the user wants an exact number of ticks. `LinearLocator` thus differs from the default locator `MaxNLocator`, for which the user specifies a maximum number of intervals rather than a precise number of ticks.

The view range algorithm in `matplotlib.ticker.LinearLocator` has been changed so that more convenient tick locations are chosen. The new algorithm returns a plot view range that is a multiple of the user-requested number of ticks. This ensures tick marks will be located at whole integers more consistently. For example, when both y-axes of a “twinx” plot use `matplotlib.ticker.LinearLocator` with the same number of ticks, their y-tick locations and grid lines will coincide.

30.6.7 matplotlib.ticker.LogLocator gains numticks kwarg

The maximum number of ticks generated by the `LogLocator` can now be controlled explicitly via setting the new ‘`numticks`’ kwarg to an integer. By default the kwarg is `None` which internally sets it to the ‘`auto`’ string, triggering a new algorithm for adjusting the maximum according to the axis length relative to the ticklabel font size.

30.6.8 matplotlib.ticker.LogFormatter: two new kwargs

Previously, minor ticks on log-scaled axes were not labeled by default. An algorithm has been added to the `LogFormatter` to control the labeling of ticks between integer powers of the base. The algorithm uses two parameters supplied in a kwarg tuple named ‘`minor_thresholds`’. See the docstring for further explanation.

To improve support for axes using `SymmetricLogLocator`, a ‘`linthresh`’ kwarg was added.

30.6.9 New defaults for 3D quiver function in `mpl_toolkits.mplot3d.axes3d.py`

Matplotlib has both a 2D and a 3D quiver function. These changes affect only the 3D function and make the default behavior of the 3D function match the 2D version. There are two changes:

- 1) The 3D quiver function previously normalized the arrows to be the same length, which makes it unusable for situations where the arrows should be different lengths and does not match the behavior of the 2D function. This normalization behavior is now controlled with the `normalize` keyword, which defaults to `False`.
- 2) The `pivot` keyword now defaults to `tail` instead of `tip`. This was done in order to match the default behavior of the 2D quiver function.

To obtain the previous behavior with the 3D quiver function, one can call the function with

```
ax.quiver(x, y, z, u, v, w, normalize=True, pivot='tip')
```

where “ax” is an `Axes3d` object created with something like

```
import mpl_toolkits.mplot3d.axes3d
ax = plt.subplot(111, projection='3d')
```

30.6.10 Stale figure behavior

Attempting to draw the figure will now mark it as not stale (independent if the draw succeeds). This change is to prevent repeatedly trying to re-draw a figure which is raising an error on draw. The previous behavior would only mark a figure as not stale after a full re-draw succeeded.

30.6.11 The spectral colormap is now nipy_spectral

The colormaps formerly known as `spectral` and `spectral_r` have been replaced by `nipy_spectral` and `nipy_spectral_r` since Matplotlib 1.3.0. Even though the colormap was deprecated in Matplotlib 1.3.0, it never raised a warning. As of Matplotlib 2.0.0, using the old names raises a deprecation warning. In the future, using the old names will raise an error.

30.6.12 Default install no longer includes test images

To reduce the size of wheels and source installs, the tests and baseline images are no longer included by default.

To restore installing the tests and images, use a `setup.cfg` with

```
[packages]
tests = True
toolkits_tests = True
```

in the source directory at build/install time.

30.7 Changes in 1.5.3

30.7.1 `ax.plot(..., marker=None)` gives default marker

Prior to 1.5.3 kwargs passed to `plot` were handled in two parts – default kwargs generated internal to `plot` (such as the cycled styles) and user supplied kwargs. The internally generated kwargs were passed to the `matplotlib.lines.Line2D.__init__` and the user kwargs were passed to `ln.set(**kwargs)` to update the artist after it was created. Now both sets of kwargs are merged and passed to `__init__`. This change was made to allow `None` to be passed in via the user kwargs to mean 'do the default thing' as is the convention through out `mpl` rather than raising an exception.

Unlike most `Line2D` setter methods `set_marker` did accept `None` as a valid input which was mapped to 'no marker'. Thus, by routing this `marker=None` through `__init__` rather than `set(...)` the meaning of `ax.plot(..., marker=None)` changed from 'no markers' to 'default markers from `rcparams`'.

This change is only evident if `mpl.rcParams['lines.marker']` has a value other than 'None' (which is string 'None' which means 'no marker').

30.8 Changes in 1.5.2

30.8.1 Default Behavior Changes

Changed default autorange behavior in boxplots

Prior to v1.5.2, the whiskers of boxplots would extend to the minimum and maximum values if the quartiles were all equal (i.e., $Q1 = \text{median} = Q3$). This behavior has been disabled by default to restore consistency with other plotting packages.

To restore the old behavior, simply set `autorange=True` when calling `plt.boxplot`.

30.9 Changes in 1.5.0

30.9.1 Code Changes

Reversed `matplotlib.cbook.ls_mapper`, added `ls_mapper_r`

Formerly, `matplotlib.cbook.ls_mapper` was a dictionary with the long-form line-style names ("solid") as keys and the short forms ("-") as values. This long-to-short mapping is now done by `ls_mapper_r`, and the short-to-long mapping is done by the `ls_mapper`.

Prevent moving artists between Axes, Property-ify `Artist.axes`, deprecate `Artist.{get,set}_axes`

This was done to prevent an `Artist` that is already associated with an `Axes` from being moved/added to a different `Axes`. This was never supported as it causes havoc with the transform stack. The apparent support for this (as it did not raise an exception) was the source of multiple bug reports and questions on SO.

For almost all use-cases, the assignment of the axes to an artist should be taken care of by the axes as part of the `Axes.add_*` method, hence the deprecation of `{get,set}_axes`.

Removing the `set_axes` method will also remove the 'axes' line from the `ACCEPTS` kwarg tables (assuming that the removal date gets here before that gets overhauled).

Tightened input validation on 'pivot' kwarg to quiver

Tightened validation so that only {'tip', 'tail', 'mid', and 'middle'} (but any capitalization) are valid values for the 'pivot' kwarg in the `Quiver.__init__` (and hence `Axes.quiver` and `plt.quiver` which both fully delegate to `Quiver`). Previously any input matching 'mid.*' would be interpreted as 'middle', 'tip.*' as 'tip' and any string not matching one of those patterns as 'tail'.

The value of `Quiver.pivot` is normalized to be in the set `{'tip', 'tail', 'middle'}` in `Quiver.__init__`.

Reordered `Axes.get_children`

The artist order returned by `Axes.get_children` did not match the one used by `Axes.draw`. They now use the same order, as `Axes.draw` now calls `Axes.get_children`.

Changed behaviour of contour plots

The default behaviour of `contour()` and `contourf()` when using a masked array is now determined by the new keyword argument `corner_mask`, or if this is not specified then the new `rcParam` `contour.corner_mask` instead. The new default behaviour is equivalent to using `corner_mask=True`; the previous behaviour can be obtained using `corner_mask=False` or by changing the `rcParam`. The example http://matplotlib.org/examples/pylab_examples/contour_corner_mask.html demonstrates the difference. Use of the old contouring algorithm, which is obtained with `corner_mask='legacy'`, is now deprecated.

Contour labels may now appear in different places than in earlier versions of Matplotlib.

In addition, the keyword argument `nchunk` now applies to `contour()` as well as `contourf()`, and it subdivides the domain into subdomains of exactly `nchunk` by `nchunk` quads, whereas previously it was only roughly `nchunk` by `nchunk` quads.

The C/C++ object that performs contour calculations used to be stored in the public attribute `QuadContourSet.Cntr`, but is now stored in a private attribute and should not be accessed by end users.

Added `set_params` function to all Locator types

This was a bug fix targeted at making the api for Locators more consistent.

In the old behavior, only locators of type `MaxNLocator` have `set_params()` defined, causing its use on any other Locator to raise an `AttributeError` (aside: `set_params(args)` is a function that sets the parameters of a Locator instance to be as specified within `args`). The fix involves moving `set_params()` to the Locator class such that all subtypes will have this function defined.

Since each of the Locator subtypes have their own modifiable parameters, a universal `set_params()` in Locator isn't ideal. Instead, a default no-operation function that raises a warning is implemented in Locator. Subtypes extending Locator will then override with their own implementations. Subtypes that do not have a need for `set_params()` will fall back onto their parent's implementation, which raises a warning as intended.

In the new behavior, Locator instances will not raise an `AttributeError` when `set_params()` is called. For Locators that do not implement `set_params()`, the default implementation in Locator is used.

Disallow None as x or y value in ax.plot

Do not allow None as a valid input for the x or y args in ax.plot. This may break some user code, but this was never officially supported (ex documented) and allowing None objects through can lead to confusing exceptions downstream.

To create an empty line use

```
ln1, = ax.plot([], [], ...)  
ln2, = ax.plot([], ...)
```

In either case to update the data in the Line2D object you must update both the x and y data.

Removed args and kwargs from MicrosecondLocator.__call__

The call signature of `__call__()` has changed from `__call__(self, *args, **kwargs)` to `__call__(self)`. This is consistent with the superclass `Locator` and also all the other `Locators` derived from this superclass.

No ValueError for the MicrosecondLocator and YearLocator

The `MicrosecondLocator` and `YearLocator` objects when called will return an empty list if the axes have no data or the view has no interval. Previously, they raised a `ValueError`. This is consistent with all the `Date Locators`.

'OffsetBox.DrawingArea' respects the 'clip' keyword argument

The call signature was `OffsetBox.DrawingArea(..., clip=True)` but nothing was done with the clip argument. The object did not do any clipping regardless of that parameter. Now the object can and does clip the child Artists if they are set to be clipped.

You can turn off the clipping on a per-child basis using `child.set_clip_on(False)`.

Add salt to clipPath id

Add salt to the hash used to determine the id of the clipPath nodes. This is to avoid conflicts when two svg documents with the same clip path are included in the same document (see <https://github.com/ipython/ipython/issues/8133> and <https://github.com/matplotlib/matplotlib/issues/4349>), however this means that the svg output is no longer deterministic if the same figure is saved twice. It is not expected that this will affect any users as the current ids are generated from an md5 hash of properties of the clip path and any user would have a very difficult time anticipating the value of the id.

Changed snap threshold for circle markers to inf

When drawing circle markers above some marker size (previously 6.0) the path used to generate the marker was snapped to pixel centers. However, this ends up distorting the marker away from a circle. By setting the snap threshold to inf snapping is never done on circles.

This change broke several tests, but is an improvement.

Preserve units with Text position

Previously the `'get_position'` method on `Text` would strip away unit information even though the units were still present. There was no inherent need to do this, so it has been changed so that unit data (if present) will be preserved. Essentially a call to `'get_position'` will return the exact value from a call to `'set_position'`.

If you wish to get the old behaviour, then you can use the new method called `'get_unitless_position'`.

New API for custom Axes view changes

Interactive pan and zoom were previously implemented using a Cartesian-specific algorithm that was not necessarily applicable to custom Axes. Three new private methods, `_get_view()`, `_set_view()`, and `_set_view_from_bbox()`, allow for custom Axes classes to override the pan and zoom algorithms. Implementors of custom Axes who override these methods may provide suitable behaviour for both pan and zoom as well as the view navigation buttons on the interactive toolbars.

30.9.2 MathTex visual changes

The spacing commands in `mathtext` have been changed to more closely match vanilla TeX.

Improved spacing in `mathtext`

The extra space that appeared after subscripts and superscripts has been removed.

No annotation coordinates wrap

In #2351 for 1.4.0 the behavior of `'axes points'`, `'axes pixel'`, `'figure points'`, `'figure pixel'` as coordinates was change to no longer wrap for negative values. In 1.4.3 this change was reverted for `'axes points'` and `'axes pixel'` and in addition caused `'axes fraction'` to wrap. For 1.5 the behavior has been reverted to as it was in 1.4.0-1.4.2, no wrapping for any type of coordinate.

30.9.3 Deprecation

Deprecated GraphicsContextBase.set__graylevel

The GraphicsContextBase.set__graylevel function has been deprecated in 1.5 and will be removed in 1.6. It has been unused. The GraphicsContextBase.set__foreground could be used instead.

deprecated idle_event

The idle_event was broken or missing in most backends and causes spurious warnings in some cases, and its use in creating animations is now obsolete due to the animations module. Therefore code involving it has been removed from all but the wx backend (where it partially works), and its use is deprecated. The animations module may be used instead to create animations.

color_cycle deprecated

In light of the new property cycling feature, the Axes method set_color_cycle is now deprecated. Calling this method will replace the current property cycle with one that cycles just the given colors.

Similarly, the rc parameter axes.color_cycle is also deprecated in lieu of the new axes.prop_cycle parameter. Having both parameters in the same rc file is not recommended as the result cannot be predicted. For compatibility, setting axes.color_cycle will replace the cycler in axes.prop_cycle with a color cycle. Accessing axes.color_cycle will return just the color portion of the property cycle, if it exists.

Timeline for removal has not been set.

30.9.4 Bundled jquery

The version of jquery bundled with the webagg backend has been upgraded from 1.7.1 to 1.11.3. If you are using the version of jquery bundled with webagg you will need to update your html files as such

```
- <script src="_static/jquery/js/jquery-1.7.1.min.js"></script>
+ <script src="_static/jquery/js/jquery-1.11.3.min.js"></script>
```

30.9.5 Code Removed

Removed Image from main namespace

Image was imported from PIL/pillow to test if PIL is available, but there is no reason to keep Image in the namespace once the availability has been determined.

Removed lod from Artist

Removed the method `set_lod` and all references to the attribute `_lod` as they are not used anywhere else in the code base. It appears to be a feature stub that was never built out.

Removed threading related classes from cbook

The classes `Scheduler`, `Timeout`, and `Idle` were in `cbook`, but are not used internally. They appear to be a prototype for the idle event system which was not working and has recently been pulled out.

Removed Lena images from sample_data

The `lena.png` and `lena.jpg` images have been removed from Matplotlib's `sample_data` directory. The images are also no longer available from `matplotlib.cbook.get_sample_data`. We suggest using `matplotlib.cbook.get_sample_data('grace_hopper.png')` or `matplotlib.cbook.get_sample_data('grace_hopper.jpg')` instead.

Legend

Removed handling of `loc` as a positional argument to `Legend`

Legend handlers

Remove code to allow legend handlers to be callable. They must now implement a method `legend_artist`.

Axis

Removed method `set_scale`. This is now handled via a private method which should not be used directly by users. It is called via `Axes.set_{x,y}scale` which takes care of ensuring the related changes are also made to the `Axes` object.

finance.py

Removed functions with ambiguous argument order from `finance.py`

Annotation

Removed `textcoords` and `xytext` properties from `Annotation` objects.

`sphinxext.ipython_*.py`

Both `ipython_console_highlighting` and `ipython_directive` have been moved to IPython.

Change your import from `'matplotlib.sphinxext.ipython_directive'` to `'IPython.sphinxext.ipython_directive'` and from `'matplotlib.sphinxext.ipython_directive'` to `'IPython.sphinxext.ipython_directive'`

`LineCollection.color`

Deprecated in 2005, use `set_color`

remove 'faceted' as a valid value for shading in `tri.tripcolor`

Use `edgecolor` instead. Added validation on shading to only be valid values.

Remove `faceted` kwarg from `scatter`

Remove support for the `faceted` kwarg. This was deprecated in `d48b34288e9651ff95c3b8a071ef5ac5cf50bae7` (2008-04-18!) and replaced by `edgecolor`.

Remove `set_colorbar` method from `ScalarMappable`

Remove `set_colorbar` method, use `colorbar` attribute directly.

`patheffects.svg`

- remove `get_proxy_renderer` method from `AbstractPathEffect` class
- remove `patch_alpha` and `offset_xy` from `SimplePatchShadow`

Remove `testing.image_util.py`

Contained only a no-longer used port of functionality from PIL

Remove `mlab.FIFOBuffer`

Not used internally and not part of core mission of `mpl`.

Remove `mlab.prepca`

Deprecated in 2009.

Remove NavigationToolbar2QTAgg

Added no functionality over the base NavigationToolbar2Qt

mpl.py

Remove the module matplotlib.mpl. Deprecated in 1.3 by PR #1670 and commit 78ce67d161625833cacff23cfe5d74920248c5b2

30.10 Changes in 1.4.x

30.10.1 Code changes

- A major refactoring of the axes module was made. The axes module has been split into smaller modules:
 - the `_base` module, which contains a new private `_AxesBase` class. This class contains all methods except plotting and labelling methods.
 - the `axes` module, which contains the `Axes` class. This class inherits from `_AxesBase`, and contains all plotting and labelling methods.
 - the `_subplot` module, with all the classes concerning subplotting.

There are a couple of things that do not exists in the axes module's namespace anymore. If you use them, you need to import them from their original location:

- `math` -> `import math`
- `ma` -> `from numpy import ma`
- `cbook` -> `from matplotlib import cbook`
- `docstring` -> `from matplotlib import docstring`
- `is_sequence_of_strings` -> `from matplotlib.cbook import is_sequence_of_strings`
- `is_string_like` -> `from matplotlib.cbook import is_string_like`
- `iterable` -> `from matplotlib.cbook import iterable`
- `itertools` -> `import itertools`
- `martist` -> `from matplotlib import artist as martist`
- `matplotlib` -> `import matplotlib`
- `mcoll` -> `from matplotlib import collections as mcoll`
- `mcolors` -> `from matplotlib import colors as mcolors`
- `mcontour` -> `from matplotlib import contour as mcontour`
- `mpatches` -> `from matplotlib import patches as mpatches`

- `mpath` -> from `matplotlib` import `path` as `mpath`
- `mquiver` -> from `matplotlib` import `quiver` as `mquiver`
- `mstack` -> from `matplotlib` import `stack` as `mstack`
- `mstream` -> from `matplotlib` import `stream` as `mstream`
- `mtable` -> from `matplotlib` import `table` as `mtable`
- As part of the refactoring to enable Qt5 support, the module `matplotlib.backends.qt4_compat` was renamed to `matplotlib.qt_compat`. `qt4_compat` is deprecated in 1.4 and will be removed in 1.5.
- The `errorbar()` method has been changed such that the upper and lower limits (`lolims`, `uplims`, `xlolims`, `xuplims`) now point in the correct direction.
- The `fmt` kwarg for `plot()` defaults.
- A bug has been fixed in the path effects rendering of fonts, which now means that the font size is consistent with non-path effect fonts. See <https://github.com/matplotlib/matplotlib/issues/2889> for more detail.
- The Sphinx extensions `ipython_directive` and `ipython_console_highlighting` have been moved to the IPython project itself. While they remain in Matplotlib for this release, they have been deprecated. Update your extensions in `conf.py` to point to `IPython.sphinxext.ipython_directive` instead of `matplotlib.sphinxext.ipython_directive`.
- In finance, almost all functions have been deprecated and replaced with a pair of functions name `*_ochl` and `*_ohlc`. The former is the 'open-close-high-low' order of quotes used previously in this module, and the latter is the 'open-high-low-close' order that is standard in finance.
- For consistency the `face_alpha` keyword to `matplotlib.patheffects.SimplePatchShadow` has been deprecated in favour of the `alpha` keyword. Similarly, the keyword `offset_xy` is now named `offset` across all `_Base`` has been renamed to `matplotlib.patheffects.AbstractPathEffect`. `matplotlib.patheffect.ProxyRenderer` has been renamed to `matplotlib.patheffects.PathEffectRenderer` and is now a full `RendererBase` subclass.
- The artist used to draw the outline of a colorbar has been changed from a `matplotlib.lines.Line2D` to `matplotlib.patches.Polygon`, thus `colorbar.ColorbarBase.outline` is now a `matplotlib.patches.Polygon` object.
- The legend handler interface has changed from a callable, to any object which implements the `legend_artists` method (a deprecation phase will see this interface be maintained for v1.4). See [Legend guide](#) for further details. Further legend changes include:
 - `matplotlib.axes.Axes._get_legend_handles()` now returns a generator of handles, rather than a list.
 - The `legend()` function's "loc" positional argument has been deprecated. Use the "loc" keyword instead.
- The `rcParams.savefig.transparent` has been added to control default transparency when saving figures.

- Slightly refactored the Annotation family. The text location in Annotation is now handled entirely handled by the underlying Text object so `set_position` works as expected. The attributes `xytext` and `textcoords` have been deprecated in favor of `xyann` and `anncoords` so that Annotation and AnnotationBbox can share a common sensibly named api for getting/setting the location of the text or box.
 - `xyann` -> set the location of the annotation
 - `xy` -> set where the arrow points to
 - `anncoords` -> set the units of the annotation location
 - `xycoords` -> set the units of the point location
 - `set_position()` -> Annotation only set location of annotation
- `matplotlib.mlab.specgram`, `matplotlib.mlab.psd`, `matplotlib.mlab.csd`, `matplotlib.mlab.cohere`, `matplotlib.mlab.cohere_pairs`, `matplotlib.pyplot.specgram`, `matplotlib.pyplot.psd`, `matplotlib.pyplot.csd`, and `matplotlib.pyplot.cohere` now raise `ValueError` where they previously raised `AssertionError`.
- For `matplotlib.mlab.psd`, `matplotlib.mlab.csd`, `matplotlib.mlab.cohere`, `matplotlib.mlab.cohere_pairs`, `matplotlib.pyplot.specgram`, `matplotlib.pyplot.psd`, `matplotlib.pyplot.csd`, and `matplotlib.pyplot.cohere`, in cases where a shape `(n, 1)` array is returned, this is now converted to a `(n,)` array. Previously, `(n, m)` arrays were averaged to an `(n,)` array, but `(n, 1)` arrays were returned unchanged. This change makes the dimensions consistent in both cases.
- Added the `rcParam axes.formatter.useoffset` to control the default value of `useOffset` in `ticker.ScalarFormatter`
- Added `Formatter` sub-class `StrMethodFormatter` which does the exact same thing as `FormatStrFormatter`, but for new-style formatting strings.
- Deprecated `matplotlib.testing.image_util` and the only function within, `matplotlib.testing.image_util.autocontrast`. These will be removed completely in v1.5.0.
- The `fnt` argument of `plot_date()` has been changed from `bo` to just `o`, so color cycling can happen by default.
- Removed the class `FigureManagerQTAgg` and deprecated `NavigationToolbar2QTAgg` which will be removed in 1.5.
- Removed formerly public (non-prefixed) attributes `rect` and `drawRect` from `FigureCanvasQTAgg`; they were always an implementation detail of the (preserved) `drawRectangle()` function.
- The function signatures of `tight_bbox.adjust_bbox` and `tight_bbox.process_figure_for_rasterizing` have been changed. A new `fixed_dpi` parameter allows for overriding the `figure.dpi` setting instead of trying to deduce the intended behaviour from the file format.
- Added support for horizontal/vertical axes padding to `mpl_toolkits.axes_grid1.ImageGrid` — argument `axes_pad` can now be tuple-like if separate axis padding is required. The original behavior is preserved.

- Added support for skewed transforms to `matplotlib.transforms.Affine2D`, which can be created using the `skew` and `skew_deg` methods.
- Added clockwise parameter to control sectors direction in `axes.pie`
- In `matplotlib.lines.Line2D` the `markevery` functionality has been extended. Previously an integer start-index and stride-length could be specified using either a two-element-list or a two-element-tuple. Now this can only be done using a two-element-tuple. If a two-element-list is used then it will be treated as numpy fancy indexing and only the two markers corresponding to the given indexes will be shown.
- removed prop kwarg from `mpl_toolkits.axes_grid1.anchored_artists.AnchoredSizeBar` call. It was passed through to the base-class `__init__` and is only used for setting padding. Now `fontproperties` (which is what is really used to set the font properties of `AnchoredSizeBar`) is passed through in place of `prop`. If `fontproperties` is not passed in, but `prop` is, then `prop` is used in place of `fontproperties`. If both are passed in, `prop` is silently ignored.
- The use of the index 0 in `plt.subplot` and related commands is deprecated. Due to a lack of validation calling `plt.subplots(2, 2, 0)` does not raise an exception, but puts an axes in the `__last__` position. This is due to the indexing in subplot being 1-based (to mirror MATLAB) so before indexing into the `GridSpec` object used to determine where the axes should go, 1 is subtracted off. Passing in 0 results in passing -1 to `GridSpec` which results in getting the last position back. Even though this behavior is clearly wrong and not intended, we are going through a deprecation cycle in an abundance of caution that any users are exploiting this 'feature'. The use of 0 as an index will raise a warning in 1.4 and an exception in 1.5.
- Clipping is now off by default on offset boxes.
- Matplotlib now uses a less-aggressive call to `gc.collect(1)` when closing figures to avoid major delays with large numbers of user objects in memory.
- The default clip value of all pie artists now defaults to `False`.

30.10.2 Code removal

- Removed `mlab.levypdf`. The code raised a numpy error (and has for a long time) and was not the standard form of the Levy distribution. `scipy.stats.levy` should be used instead

30.11 Changes in 1.3.x

30.11.1 Changes in 1.3.1

It is rare that we make an API change in a bugfix release, however, for 1.3.1 since 1.3.0 the following change was made:

- `text.Text.cached` (used to cache font objects) has been made into a private variable. Among the obvious encapsulation benefit, this removes this confusing-looking member from the documentation.

- The method `hist()` now always returns bin occupancies as an array of type `float`. Previously, it was sometimes an array of type `int`, depending on the call.

30.11.2 Code removal

- The following items that were deprecated in version 1.2 or earlier have now been removed completely.
 - The Qt 3.x backends (`qt` and `qtagg`) have been removed in favor of the Qt 4.x backends (`qt4` and `qt4agg`).
 - The `FltkAgg` and `Emf` backends have been removed.
 - The `matplotlib.nxutils` module has been removed. Use the functionality on `matplotlib.path.Path.contains_point` and `friends` instead.
 - Instead of `axes.Axes.get_frame`, use `axes.Axes.patch`.
 - The following kwargs to the legend function have been renamed:
 - * `pad` -> `borderpad`
 - * `labelsep` -> `labelspacing`
 - * `handlelen` -> `handlelength`
 - * `handletextsep` -> `handletextpad`
 - * `axespad` -> `borderaxespad`

Related to this, the following `rcParams` have been removed:

- * `legend.pad`, `legend.labelsep`, `legend.handlelen`, `legend.handletextsep` and `legend.axespad`
- For the `hist` function, instead of `width`, use `rwidth` (relative width).
- On `patches.Circle`, the `resolution` kwarg has been removed. For a circle made up of line segments, use `patches.CirclePolygon`.
- The printing functions in the `Wx` backend have been removed due to the burden of keeping them up-to-date.
- `mlab.liaupunov` has been removed.
- `mlab.save`, `mlab.load`, `pylab.save` and `pylab.load` have been removed. We recommend using `numpy.savetxt` and `numpy.loadtxt` instead.
- `widgets.HorizontalSpanSelector` has been removed. Use `widgets.SpanSelector` instead.

30.11.3 Code deprecation

- The `CocoaAgg` backend has been deprecated, with the possibility for deletion or resurrection in a future release.

- The top-level functions in `matplotlib.path` that are implemented in C++ were never meant to be public. Instead, users should use the Pythonic wrappers for them in the `path.Path` and `collections.Collection` classes. Use the following mapping to update your code:
 - `point_in_path` -> `path.Path.contains_point`
 - `get_path_extents` -> `path.Path.get_extents`
 - `point_in_path_collection` -> `collection.Collection.contains`
 - `path_in_path` -> `path.Path.contains_path`
 - `path_intersects_path` -> `path.Path.intersects_path`
 - `convert_path_to_polygons` -> `path.Path.to_polygons`
 - `cleanup_path` -> `path.Path.cleaned`
 - `points_in_path` -> `path.Path.contains_points`
 - `clip_path_to_rect` -> `path.Path.clip_to_bbox`
- `matplotlib.colors.normalize` and `matplotlib.colors.no_norm` have been deprecated in favour of `matplotlib.colors.Normalize` and `matplotlib.colors.NoNorm` respectively.
- The `ScalarMappable` class' `set_colorbar` is now deprecated. Instead, the `matplotlib.cm.ScalarMappable.colorbar` attribute should be used. In previous Matplotlib versions this attribute was an undocumented tuple of (`colorbar_instance`, `colorbar_axes`) but is now just `colorbar_instance`. To get the colorbar axes it is possible to just use the `ax` attribute on a `colorbar` instance.
- The `mpl` module is now deprecated. Those who relied on this module should transition to simply using `import matplotlib as mpl`.

30.11.4 Code changes

- `Patch` now fully supports using `RGBA` values for its `facecolor` and `edgecolor` attributes, which enables faces and edges to have different alpha values. If the `Patch` object's `alpha` attribute is set to anything other than `None`, that value will override any alpha-channel value in both the face and edge colors. Previously, if `Patch` had `alpha=None`, the alpha component of `edgecolor` would be applied to both the edge and face.
- The optional `isRGB` argument to `set_foreground()` (and the other `GraphicsContext` classes that descend from it) has been renamed to `isRGBA`, and should now only be set to `True` if the `fg` color argument is known to be an `RGBA` tuple.
- For `Patch`, the `capstyle` used is now `butt`, to be consistent with the default for most other objects, and to avoid problems with non-solid linestyle appearing solid when using a large linewidth. Previously, `Patch` used `capstyle='projecting'`.
- `Path` objects can now be marked as `readonly` by passing `readonly=True` to its constructor. The built-in `path` singletons, obtained through `Path.unit*` class methods return `readonly` paths. If you have code that modified these, you will need to make a `deepcopy` first, using either:

```
import copy
path = copy.deepcopy(Path.unit_circle())

# or

path = Path.unit_circle().deepcopy()
```

Deep copying a Path always creates an editable (i.e. non-readonly) Path.

- The list at Path.NUM_VERTICES was replaced by a dictionary mapping Path codes to the number of expected vertices at NUM_VERTICES_FOR_CODE.
- To support XKCD style plots, the matplotlib.path.cleanup_path() method's signature was updated to require a sketch argument. Users of matplotlib.path.cleanup_path() are encouraged to use the new cleaned() Path method.
- Data limits on a plot now start from a state of having "null" limits, rather than limits in the range (0, 1). This has an effect on artists that only control limits in one direction, such as axvline and axhline, since their limits will not longer also include the range (0, 1). This fixes some problems where the computed limits would be dependent on the order in which artists were added to the axes.
- Fixed a bug in setting the position for the right/top spine with data position type. Previously, it would draw the right or top spine at +1 data offset.
- In FancyArrow, the default arrow head width, head_width, has been made larger to produce a visible arrow head. The new value of this kwarg is head_width = 20 * width.
- It is now possible to provide number of levels + 1 colors in the case of extend='both' for contourf (or just number of levels colors for an extend value min or max) such that the resulting colormap's set_under and set_over are defined appropriately. Any other number of colors will continue to behave as before (if more colors are provided than levels, the colors will be unused). A similar change has been applied to contour, where extend='both' would expect number of levels + 2 colors.
- A new keyword extendrect in colorbar() and ColorbarBase allows one to control the shape of colorbar extensions.
- The extension of MultiCursor to both vertical (default) and/or horizontal cursor implied that self.line is replaced by self.vline for vertical cursors lines and self.hline is added for the horizontal cursors lines.
- On POSIX platforms, the report_memory() function raises NotImplementedError instead of OSError if the ps command cannot be run.
- The matplotlib.cbook.check_output() function has been moved to matplotlib.compat.subprocess().

30.11.5 Configuration and rcParams

- On Linux, the user-specific matplotlibrc configuration file is now located in config/matplotlib/matplotlibrc to conform to the [XDG Base Directory Specification](#).
- The font.* rcParams now affect only text objects created after the rcParam has been set, and will not retroactively affect already existing text objects. This brings their behavior in line with most other rcParams.
- Removed call of grid() in plotfile(). To draw the axes grid, set the axes.grid rcParam to True, or explicitly call grid().

30.12 Changes in 1.2.x

- The classic option of the rc parameter toolbar is deprecated and will be removed in the next release.
- The isvector() method has been removed since it is no longer functional.
- The rasterization_zorder property on Axes a zorder below which artists are rasterized. This has defaulted to -30000.0, but it now defaults to [None](#), meaning no artists will be rasterized. In order to rasterize artists below a given zorder value, set_rasterization_zorder must be explicitly called.
- In scatter(), and scatter, when specifying a marker using a tuple, the angle is now specified in degrees, not radians.
- Using twinx() or twiny() no longer overrides the current locaters and formatters on the axes.
- In contourf(), the handling of the extend kwarg has changed. Formerly, the extended ranges were mapped after to 0, 1 after being normed, so that they always corresponded to the extreme values of the colormap. Now they are mapped outside this range so that they correspond to the special colormap values determined by the set_under() and set_over() methods, which default to the colormap end points.
- The new rc parameter savefig.format replaces cairo.format and savefig.extension, and sets the default file format used by matplotlib.figure.Figure.savefig().
- In pie() and pie(), one can now set the radius of the pie; setting the radius to 'None' (the default value), will result in a pie with a radius of 1 as before.
- Use of projection_factory() is now deprecated in favour of axes class identification using process_projection_requirements() followed by direct axes class invocation (at the time of writing, functions which do this are: add_axes(), add_subplot() and gca()). Therefore:

```
key = figure._make_key(*args, **kwargs)
ispolar = kwargs.pop('polar', False)
projection = kwargs.pop('projection', None)
if ispolar:
    if projection is not None and projection != 'polar':
        raise ValueError('polar and projection args are inconsistent')
```

(continues on next page)

(continued from previous page)

```

    projection = 'polar'
    ax = projection_factory(projection, self, rect, **kwargs)
    key = self._make_key(*args, **kwargs)

    # is now

    projection_class, kwargs, key = \
        process_projection_requirements(self, *args, **kwargs)
    ax = projection_class(self, rect, **kwargs)

```

This change means that third party objects can expose themselves as Matplotlib axes by providing a `_as_mpl_axes` method. See Developer's guide for creating scales and transformations for more detail.

- A new keyword `extendfrac` in `colorbar()` and `ColorbarBase` allows one to control the size of the triangular minimum and maximum extensions on colorbars.
- A new keyword `capthick` in `errorbar()` has been added as an intuitive alias to the `markeredgewidth` and `mew` keyword arguments, which indirectly controlled the thickness of the caps on the errorbars. For backwards compatibility, specifying either of the original keyword arguments will override any value provided by `capthick`.
- Transform subclassing behaviour is now subtly changed. If your transform implements a non-affine transformation, then it should override the `transform_non_affine` method, rather than the generic `transform` method. Previously transforms would define `transform` and then copy the method into `transform_non_affine`:

```

class MyTransform(mtransforms.Transform):
    def transform(self, xy):
        ...
    transform_non_affine = transform

```

This approach will no longer function correctly and should be changed to:

```

class MyTransform(mtransforms.Transform):
    def transform_non_affine(self, xy):
        ...

```

- Artists no longer have `x_isdata` or `y_isdata` attributes; instead any artist's transform can be interrogated with `artist_instance.get_transform().contains_branch(ax.transData)`
- Lines added to an axes now take into account their transform when updating the data and view limits. This means transforms can now be used as a pre-transform. For instance:

```

>>> import matplotlib.pyplot as plt
>>> import matplotlib.transforms as mtransforms
>>> ax = plt.axes()
>>> ax.plot(range(10), transform=mtransforms.Affine2D().scale(10) + ax.transData)

```

(continues on next page)

(continued from previous page)

```
>>> print(ax.viewLim)
Bbox('array([[ 0.,  0.],\n      [ 90.,  90.]])')
```

- One can now easily get a transform which goes from one transform's coordinate system to another, in an optimized way, using the new `subtract` method on a transform. For instance, to go from data coordinates to axes coordinates:

```
>>> import matplotlib.pyplot as plt
>>> ax = plt.axes()
>>> data2ax = ax.transData - ax.transAxes
>>> print(ax.transData.depth, ax.transAxes.depth)
3, 1
>>> print(data2ax.depth)
2
```

for versions before 1.2 this could only be achieved in a sub-optimal way, using `ax.transData + ax.transAxes.inverted()` (depth is a new concept, but had it existed it would return 4 for this example).

- `twinx` and `twiny` now returns an instance of `SubplotBase` if parent axes is an instance of `SubplotBase`.
- All Qt3-based backends are now deprecated due to the lack of py3k bindings. Qt and QtAgg backends will continue to work in v1.2.x for py2.6 and py2.7. It is anticipated that the Qt3 support will be completely removed for the next release.
- `ColorConverter`, `Colormap` and `Normalize` now subclasses object
- `ContourSet` instances no longer have a transform attribute. Instead, access the transform with the `get_transform` method.

30.13 Changes in 1.1.x

- Added new `matplotlib.sankey.Sankey` for generating Sankey diagrams.
- In `imshow()`, setting interpolation to 'nearest' will now always mean that the nearest-neighbor interpolation is performed. If you want the no-op interpolation to be performed, choose 'none'.
- There were errors in how the tri-functions were handling input parameters that had to be fixed. If your tri-plots are not working correctly anymore, or you were working around apparent mistakes, please see issue #203 in the github tracker. When in doubt, use `kwargs`.
- The 'symlog' scale had some bad behavior in previous versions. This has now been fixed and users should now be able to use it without frustrations. The fixes did result in some minor changes in appearance for some users who may have been depending on the bad behavior.
- There is now a common set of markers for all plotting functions. Previously, some markers existed only for `scatter()` or just for `plot()`. This is now no longer the case. This merge did

result in a conflict. The string 'd' now means "thin diamond" while 'D' will mean "regular diamond".

30.14 Changes beyond 0.99.x

- The default behavior of `matplotlib.axes.Axes.set_xlim()`, `matplotlib.axes.Axes.set_ylim()`, and `matplotlib.axes.Axes.axis()`, and their corresponding pyplot functions, has been changed: when view limits are set explicitly with one of these methods, autoscaling is turned off for the matching axis. A new auto kwarg is available to control this behavior. The limit kwargs have been renamed to left and right instead of xmin and xmax, and bottom and top instead of ymin and ymax. The old names may still be used, however.
- There are five new Axes methods with corresponding pyplot functions to facilitate autoscaling, tick location, and tick label formatting, and the general appearance of ticks and tick labels:
 - `matplotlib.axes.Axes.autoscale()` turns autoscaling on or off, and applies it.
 - `matplotlib.axes.Axes.margins()` sets margins used to autoscale the `matplotlib.axes.Axes.viewLim` based on the `matplotlib.axes.Axes.dataLim`.
 - `matplotlib.axes.Axes.locator_params()` allows one to adjust axes locator parameters such as nbins.
 - `matplotlib.axes.Axes.ticklabel_format()` is a convenience method for controlling the `matplotlib.ticker.ScalarFormatter` that is used by default with linear axes.
 - `matplotlib.axes.Axes.tick_params()` controls direction, size, visibility, and color of ticks and their labels.
- The `matplotlib.axes.Axes.bar()` method accepts a `error_kw` kwarg; it is a dictionary of kwargs to be passed to the errorbar function.
- The `matplotlib.axes.Axes.hist()` color kwarg now accepts a sequence of color specs to match a sequence of datasets.
- The `EllipseCollection` has been changed in two ways:
 - There is a new units option, 'xy', that scales the ellipse with the data units. This matches the `:class:'~matplotlib.patches.Ellipse'` scaling.
 - The height and width kwargs have been changed to specify the height and width, again for consistency with `Ellipse`, and to better match their names; previously they specified the half-height and half-width.
- There is a new rc parameter `axes.color_cycle`, and the color cycle is now independent of the rc parameter `lines.color`. `matplotlib.Axes.set_default_color_cycle()` is deprecated.
- You can now print several figures to one pdf file and modify the document information dictionary of a pdf file. See the docstrings of the class `matplotlib.backends.backend_pdf.PdfPages` for more information.
- Removed `configobj` and `enthought.traits` packages, which are only required by the experimental traitled config and are somewhat out of date. If needed, install them independently.

- The new rc parameter `savefig.extension` sets the filename extension that is used by `matplotlib.figure.Figure.savefig()` if its `fname` argument lacks an extension.
- In an effort to simplify the backend API, all clipping rectangles and paths are now passed in using `GraphicsContext` objects, even on collections and images. Therefore:

```
draw_path_collection(self, master_transform, cliprect, clippath,
                    clippath_trans, paths, all_transforms, offsets,
                    offsetTrans, facecolors, edgecolors, linewidths,
                    linestyle, antialiaseds, urls)

# is now

draw_path_collection(self, gc, master_transform, paths, all_transforms,
                    offsets, offsetTrans, facecolors, edgecolors,
                    linewidths, linestyle, antialiaseds, urls)

draw_quad_mesh(self, master_transform, cliprect, clippath,
               clippath_trans, meshWidth, meshHeight, coordinates,
               offsets, offsetTrans, facecolors, antialiased,
               showedges)

# is now

draw_quad_mesh(self, gc, master_transform, meshWidth, meshHeight,
               coordinates, offsets, offsetTrans, facecolors,
               antialiased, showedges)

draw_image(self, x, y, im, bbox, clippath=None, clippath_trans=None)

# is now

draw_image(self, gc, x, y, im)
```

- There are four new `Axes` methods with corresponding pyplot functions that deal with unstructured triangular grids:
 - `matplotlib.axes.Axes.tricontour()` draws contour lines on a triangular grid.
 - `matplotlib.axes.Axes.tricontourf()` draws filled contours on a triangular grid.
 - `matplotlib.axes.Axes.tripcolor()` draws a pseudocolor plot on a triangular grid.
 - `matplotlib.axes.Axes.triplot()` draws a triangular grid as lines and/or markers.

30.15 Changes in 0.99

- `pylab` no longer provides a load and save function. These are available in `matplotlib.mlab`, or you can use `numpy.loadtxt` and `numpy.savetxt` for text files, or `np.save` and `np.load` for binary numpy arrays.
- User-generated colormaps can now be added to the set recognized by `matplotlib.cm.get_cmap()`. Colormaps can be made the default and applied to the current image using `matplotlib.pyplot.set_cmap()`.
- changed `use_mrecords` default to `False` in `mlab.csv2rec` since this is partially broken
- Axes instances no longer have a "frame" attribute. Instead, use the new "spines" attribute. Spines is a dictionary where the keys are the names of the spines (e.g., 'left', 'right' and so on) and the values are the artists that draw the spines. For normal (rectilinear) axes, these artists are `Line2D` instances. For other axes (such as polar axes), these artists may be `Patch` instances.
- Polar plots no longer accept a resolution kwarg. Instead, each `Path` must specify its own number of interpolation steps. This is unlikely to be a user-visible change – if interpolation of data is required, that should be done before passing it to `Matplotlib`.

30.16 Changes for 0.98.x

- `psd()`, `csd()`, and `cohere()` will now automatically wrap negative frequency components to the beginning of the returned arrays. This is much more sensible behavior and makes them consistent with `specgram()`. The previous behavior was more of an oversight than a design decision.
- Added new keyword parameters `nonposx`, `nonposy` to `matplotlib.axes.Axes` methods that set log scale parameters. The default is still to mask out non-positive values, but the kwargs accept 'clip', which causes non-positive values to be replaced with a very small positive value.
- Added new `matplotlib.pyplot.fignum_exists()` and `matplotlib.pyplot.get_fignums()`; they merely expose information that had been hidden in `matplotlib.__pylab_helpers`.
- Deprecated `numerix` package.
- Added new `matplotlib.image.imsave()` and exposed it to the `matplotlib.pyplot` interface.
- Remove support for `pyExcelerator` in `exceltools` – use `xlwt` instead
- Changed the defaults of `acorr` and `xcorr` to use `usevlines=True`, `maxlags=10` and `normed=True` since these are the best defaults
- Following keyword parameters for `matplotlib.label.Label` are now deprecated and new set of parameters are introduced. The new parameters are given as a fraction of the font-size. Also, `scatteryoffsets`, `fancybox` and `columnspacing` are added as keyword parameters.

Deprecated	New
<code>pad</code>	<code>borderpad</code>
<code>labelsep</code>	<code>labelspacing</code>
<code>handlelen</code>	<code>handlelength</code>
<code>handletextsep</code>	<code>handletextpad</code>
<code>axespadd</code>	<code>borderaxespad</code>

- Removed the `configobj` and experimental traits rc support
- Modified `matplotlib.mlab.psd()`, `matplotlib.mlab.csd()`, `matplotlib.mlab.cohere()`, and `matplotlib.mlab.spectrogram()` to scale one-sided densities by a factor of 2. Also, optionally scale the densities by the sampling frequency, which gives true values of densities that can be integrated by the returned frequency values. This also gives better MATLAB compatibility. The corresponding `matplotlib.axes.Axes` methods and `matplotlib.pyplot` functions were updated as well.
- Font lookup now uses a nearest-neighbor approach rather than an exact match. Some fonts may be different in plots, but should be closer to what was requested.
- `matplotlib.axes.Axes.set_xlim()`, `matplotlib.axes.Axes.set_ylim()` now return a copy of the `viewlim` array to avoid modify-in-place surprises.
- `matplotlib.afm.AFM.get_fullname()` and `matplotlib.afm.AFM.get_familyname()` no longer raise an exception if the AFM file does not specify these optional attributes, but returns a guess based on the required `FontName` attribute.
- Changed precision kwarg in `matplotlib.pyplot.spy()`; default is 0, and the string value 'present' is used for sparse arrays only to show filled locations.
- `matplotlib.collections.EllipseCollection` added.
- Added `angles` kwarg to `matplotlib.pyplot.quiver()` for more flexible specification of the arrow angles.
- Deprecated (raise `NotImplementedError`) all the `mlab2` functions from `matplotlib.mlab` out of concern that some of them were not clean room implementations.
- Methods `matplotlib.collections.Collection.get_offsets()` and `matplotlib.collections.Collection.set_offsets()` added to `Collection` base class.
- `matplotlib.figure.Figure.figurePatch` renamed `matplotlib.figure.Figure.patch`; `matplotlib.axes.Axes.axesPatch` renamed `matplotlib.axes.Axes.patch`; `matplotlib.axes.Axes.axesFrame` renamed `matplotlib.axes.Axes.frame`. `matplotlib.axes.Axes.get_frame()`, which returns `matplotlib.axes.Axes.patch`, is deprecated.
- Changes in the `matplotlib.contour.ContourLabeler` attributes (`matplotlib.pyplot.clabel()` function) so that they all have a form like `.labelAttribute`. The three attributes that are most likely to be used by end users, `.cl`, `.cl_xy` and `.cl_cvalues` have been maintained for the moment (in addition to their renamed versions), but they are deprecated and will eventually be removed.

- Moved several functions in `matplotlib.mlab` and `matplotlib.cbook` into a separate module `matplotlib.numerical_methods` because they were unrelated to the initial purpose of `mlab` or `cbook` and appeared more coherent elsewhere.

30.17 Changes for 0.98.1

- Removed broken `matplotlib.axes3d` support and replaced it with a non-implemented error pointing to 0.91.x

30.18 Changes for 0.98.0

- `matplotlib.image.imread()` now no longer always returns RGBA data—if the image is luminance or RGB, it will return a `MxN` or `MxNx3` array if possible. Also `uint8` is no longer always forced to float.
- Rewrote the `matplotlib.cm.ScalarMappable` callback infrastructure to use `matplotlib.cbook.CallbackRegistry` rather than custom callback handling. Any users of `matplotlib.cm.ScalarMappable.add_observer()` of the `ScalarMappable` should use the `matplotlib.cm.ScalarMappable.callbacks.CallbackRegistry` instead.
- New axes function and Axes method provide control over the plot color cycle: `matplotlib.axes.set_default_color_cycle()` and `matplotlib.axes.Axes.set_color_cycle()`.
- Matplotlib now requires Python 2.4, so `matplotlib.cbook` will no longer provide `set`, `enumerate()`, `reversed()` or `izip()` compatibility functions.
- In Numpy 1.0, bins are specified by the left edges only. The axes method `matplotlib.axes.Axes.hist()` now uses future Numpy 1.3 semantics for histograms. Providing `binedges`, the last value gives the upper-right edge now, which was implicitly set to `+infinity` in Numpy 1.0. This also means that the last bin doesn't contain upper outliers any more by default.
- New axes method and pyplot function, `hexbin()`, is an alternative to `scatter()` for large datasets. It makes something like a `pcolor()` of a 2-D histogram, but uses hexagonal bins.
- New kwarg, `symmetric`, in `matplotlib.ticker.MaxNLocator` allows one require an axis to be centered around zero.
- Toolkits must now be imported from `mpl_toolkits` (not `matplotlib.toolkits`)

30.18.1 Notes about the transforms refactoring

A major new feature of the 0.98 series is a more flexible and extensible transformation infrastructure, written in Python/Numpy rather than a custom C extension.

The primary goal of this refactoring was to make it easier to extend matplotlib to support new kinds of projections. This is mostly an internal improvement, and the possible user-visible changes it allows are yet to come.

See `matplotlib.transforms` for a description of the design of the new transformation framework.

For efficiency, many of these functions return views into Numpy arrays. This means that if you hold on to a reference to them, their contents may change. If you want to store a snapshot of their current values, use the Numpy array method `copy()`.

The view intervals are now stored only in one place – in the `matplotlib.axes.Axes` instance, not in the locator instances as well. This means locators must get their limits from their `matplotlib.axis.Axis`, which in turn looks up its limits from the `Axes`. If a locator is used temporarily and not assigned to an `Axis` or `Axes`, (e.g., in `matplotlib.contour`), a dummy axis must be created to store its bounds. Call `matplotlib.ticker.Locator.create_dummy_axis()` to do so.

The functionality of `Pbox` has been merged with `Bbox`. Its methods now all return copies rather than modifying in place.

The following lists many of the simple changes necessary to update code from the old transformation framework to the new one. In particular, methods that return a copy are named with a verb in the past tense, whereas methods that alter an object in place are named with a verb in the present tense.

matplotlib.transforms

Old method	New method
Bbox.get_bounds()	transforms.Bbox.bounds
Bbox.width()	transforms.Bbox.width
Bbox.height()	transforms.Bbox.height
Bbox.intervalx(). get_bounds()	transforms.Bbox.intervalx
Bbox.intervalx(). set_bounds()	[Bbox.intervalx is now a property.]
Bbox.intervaly(). get_bounds()	transforms.Bbox.intervaly
Bbox.intervaly(). set_bounds()	[Bbox.intervaly is now a property.]
Bbox.xmin()	transforms.Bbox.x0 or transforms.Bbox.xmin ¹
Bbox.ymin()	transforms.Bbox.y0 or transforms.Bbox.ymin ¹
Bbox.xmax()	transforms.Bbox.x1 or transforms.Bbox.xmax ¹
Bbox.ymax()	transforms.Bbox.y1 or transforms.Bbox.ymax ¹
Bbox.overlaps(bboxes)	Bbox.count_overlaps(bboxes)
bbox_all(bboxes)	Bbox.union(bboxes) [transforms.Bbox.union() is a staticmethod.]
lbwh_to_bbox(l, b, w, h)	Bbox.from_bounds(x0, y0, w, h) [transforms.Bbox.from_bounds() is a staticmethod.]
inverse_transform_bbox(trans, bbox)	Bbox.inverse_transformed(trans)
Interval.contains_open(v)	interval_contains_open(tuple, v)
Interval.contains(v)	interval_contains(tuple, v)
identity_transform()	matplotlib.transforms.IdentityTransform
blend_xy_sep_transform(xtrans, ytrans)	blended_transform_factory(xtrans, ytrans)
scale_transform(xs, ys)	Affine2D().scale(xs[, ys])
get_bbox_transform(boxin, boxout)	BboxTransform(boxin, boxout) or BboxTransformFrom(boxin) or BboxTransformTo(boxout)
Transform. seq_xy_tup(points)	Transform.transform(points)
Transform. inverse_xy_tup(points)	Transform.inverted().transform(points)

¹ The Bbox is bound by the points (x0, y0) to (x1, y1) and there is no defined order to these points, that is, x0 is not necessarily the left edge of the box. To get the left edge of the Bbox, use the read-only property xmin.

matplotlib.axes

Old method	New method
<code>Axes.get_position()</code>	<code>matplotlib.axes.Axes.get_position()</code> ²
<code>Axes.set_position()</code>	<code>matplotlib.axes.Axes.set_position()</code> ³
<code>Axes.toggle_log_lineary()</code>	<code>matplotlib.axes.Axes.set_yscale()</code> ⁴
Subplot class	removed.

The Polar class has moved to `matplotlib.projections.polar`.

matplotlib.artist

Old method	New method
<code>Artist.set_clip_path(path)</code>	<code>Artist.set_clip_path(path, transform)</code> ⁵

matplotlib.collections

Old method	New method
<code>linestyle</code>	<code>linestyles</code> ⁶

matplotlib.colors

Old method	New method
<code>ColorConverter.to_rgba_list(c)</code>	<code>ColorConverter.to_rgba_array(c)</code> [matplotlib.colors.ColorConverter.to_rgba_array() returns an Nx4 Numpy array of RGBA color quadruples.]

matplotlib.contour

Old method	New method
<code>Contour.__segments</code>	<code>matplotlib.contour.Contour.get_paths`()</code> [Returns a list of matplotlib.path.Path instances.]

² `matplotlib.axes.Axes.get_position()` used to return a list of points, now it returns a `matplotlib.transforms.Bbox` instance.

³ `matplotlib.axes.Axes.set_position()` now accepts either four scalars or a `matplotlib.transforms.Bbox` instance.

⁴ Since the refactoring allows for more than two scale types ('log' or 'linear'), it no longer makes sense to have a `toggle`. `Axes.toggle_log_lineary()` has been removed.

⁵ `matplotlib.artist.Artist.set_clip_path()` now accepts a `matplotlib.path.Path` instance and a `matplotlib.transforms.Transform` that will be applied to the path immediately before clipping.

⁶ Linestyles are now treated like all other collection attributes, i.e. a single value or multiple values may be provided.

matplotlib.figure

Old method	New method
Figure.dpi.get() / Figure.dpi.set()	matplotlib.figure.Figure.dpi (a property)

matplotlib.patches

Old method	New method
Patch.get_verts()	matplotlib.patches.Patch.get_path() [Returns a matplotlib.path.Path instance]

matplotlib.backend_bases

Old method	New method
GraphicsContext.set_clip_rectangle(tuple)	GraphicsContext.set_clip_rectangle(bbox)
GraphicsContext.get_clip_path()	GraphicsContext.get_clip_path() ⁷
GraphicsContext.set_clip_path()	GraphicsContext.set_clip_path() ⁸

RendererBase

New methods:

- draw_path(self, gc, path, transform, rgbFace)
- draw_markers(self, gc, marker_path, marker_trans, path, trans, rgbFace)
- draw_path_collection(self, master_transform, cliprect, clippath, clippath_trans, paths, all_transforms, offsets, offsetTrans, facecolors, edgecolors, linewidths, linestyle, antialiaseds) [optional]

Changed methods:

- draw_image(self, x, y, im, bbox) is now draw_image(self, x, y, im, bbox, clippath, clippath_trans)

Removed methods:

- draw_arc
- draw_line_collection

⁷ matplotlib.backend_bases.GraphicsContext.get_clip_path() returns a tuple of the form (path, affine_transform), where path is a matplotlib.path.Path instance and affine_transform is a matplotlib.transforms.Affine2D instance.

⁸ matplotlib.backend_bases.GraphicsContext.set_clip_path() now only accepts a matplotlib.transforms.TransformPath instance.

- `draw_line`
- `draw_lines`
- `draw_point`
- `draw_quad_mesh`
- `draw_poly_collection`
- `draw_polygon`
- `draw_rectangle`
- `draw_regpoly_collection`

30.19 Changes for 0.91.2

- For `csv2rec()`, `checkrows=0` is the new default indicating all rows will be checked for type inference
- A warning is issued when an image is drawn on log-scaled axes, since it will not log-scale the image data.
- Moved `rec2gtk()` to `matplotlib.toolkits.gtktools`
- Moved `rec2excel()` to `matplotlib.toolkits.exceltools`
- Removed, dead/experimental `ExampleInfo`, `Namespace` and `Importer` code from `matplotlib.__init__`

30.20 Changes for 0.91.1

30.21 Changes for 0.91.0

- Changed `cbook.is_file_like()` to `cbook.is_writable_file_like()` and corrected behavior.
- Added `ax` kwarg to `pyplot.colorbar()` and `Figure.colorbar()` so that one can specify the axes object from which space for the colorbar is to be taken, if one does not want to make the colorbar axes manually.
- Changed `cbook.reversed()` so it yields a tuple rather than a (index, tuple). This agrees with the python `reversed` builtin, and `cbook` only defines `reversed` if python doesn't provide the builtin.
- Made `skiprows=1` the default on `csv2rec()`
- The `gd` and `paint` backends have been deleted.
- The `errorbar` method and function now accept additional kwargs so that upper and lower limits can be indicated by capping the bar with a caret instead of a straight line segment.

- The `matplotlib.dviread` file now has a parser for files like `psfonts.map` and `pdftex.map`, to map TeX font names to external files.
- The file `matplotlib.type1font` contains a new class for Type 1 fonts. Currently it simply reads pfa and pfb format files and stores the data in a way that is suitable for embedding in pdf files. In the future the class might actually parse the font to allow e.g., subsetting.
- `matplotlib.FT2Font` now supports `FT_Attach_File()`. In practice this can be used to read an afm file in addition to a pfa/pfb file, to get metrics and kerning information for a Type 1 font.
- The AFM class now supports querying `CapHeight` and stem widths. The `get_name_char` method now has an `isord` kwarg like `get_width_char`.
- Changed `pcolor()` default to `shading='flat'`; but as noted now in the docstring, it is preferable to simply use the `edgecolor` kwarg.
- The `mathtext` font commands (`\cal`, `\rm`, `\it`, `\tt`) now behave as TeX does: they are in effect until the next font change command or the end of the grouping. Therefore uses of `\cal{R}` should be changed to `{\cal R}`. Alternatively, you may use the new LaTeX-style font commands (`\mathcal`, `\mathrm`, `\mathit`, `\mathtt`) which do affect the following group, e.g., `\mathcal{R}`.
- Text creation commands have a new default `linespacing` and a new `linespacing` kwarg, which is a multiple of the maximum vertical extent of a line of ordinary text. The default is 1.2; `linespacing=2` would be like ordinary double spacing, for example.
- Changed default kwarg in `matplotlib.colors.Normalize.__init__()` to `clip=False`; clipping silently defeats the purpose of the special over, under, and bad values in the colormap, thereby leading to unexpected behavior. The new default should reduce such surprises.
- Made the `emit` property of `set_xlim()` and `set_ylim()` True by default; removed the Axes custom callback handling into a `'callbacks'` attribute which is a `CallbackRegistry` instance. This now supports the `'xlim_changed'` and `'ylim_changed'` Axes events.

30.22 Changes for 0.90.1

The file `dviread.py` has a (very limited and fragile) dvi reader for usetex support. The API might change in the future so don't depend on it yet.

Removed deprecated support for a float value as a gray-scale; now it must be a string, like `'0.5'`. Added alpha kwarg to `ColorConverter.to_rgba_list`.

New method `set_bounds(vmin, vmax)` for formatters, locators sets the `viewInterval` and `dataInterval` from floats.

Removed deprecated `colorbar_classic`.

(continues on next page)

(continued from previous page)

Line2D.get_xdata and get_ydata valid_only=False kwarg is replaced by orig=True. When True, it returns the original data, otherwise the processed data (masked, converted)

Some modifications to the units interface.

units.ConversionInterface.tickers renamed to

units.ConversionInterface.axisinfo and it now returns a

units.AxisInfo object rather than a tuple. This will make it

easier to add axis info functionality (e.g., I added a default label on this iteration) w/o having to change the tuple length and hence the API of the client code every time new functionality is added.

Also, units.ConversionInterface.convert_to_value is now simply named units.ConversionInterface.convert.

Axes.errorbar uses Axes.vlines and Axes.hlines to draw its error limits int he vertical and horizontal direction. As you'll see in the changes below, these functions now return a LineCollection rather than a list of lines. The new return signature for errorbar is ylines, caplines, errorcollections where errorcollections is a xerrcollection, yerrcollection

Axes.vlines and Axes.hlines now create and returns a LineCollection, not a list of lines. This is much faster. The kwarg signature has changed, so consult the docs

MaxNLocator accepts a new Boolean kwarg ('integer') to force ticks to integer locations.

Commands that pass an argument to the Text constructor or to Text.set_text() now accept any object that can be converted with '%s'. This affects xlabel(), title(), etc.

Barh now takes a **kwargs dict instead of most of the old arguments. This helps ensure that bar and barh are kept in sync, but as a side effect you can no longer pass e.g., color as a positional argument.

ft2font.get_charmap() now returns a dict that maps character codes to glyph indices (until now it was reversed)

Moved data files into lib/matplotlib so that setuptools' develop mode works. Re-organized the mpl-data layout so that this source structure is maintained in the installation. (i.e., the 'fonts' and 'images' sub-directories are maintained in site-packages.).

(continues on next page)

(continued from previous page)

Suggest removing site-packages/matplotlib/mpl-data and
~/.matplotlib/ttfont.cache before installing

30.23 Changes for 0.90.0

All artists now implement a "pick" method which users should **not** call. Rather, **set** the "picker" property of **any** artist you want to pick on (the epsilon distance **in** points **for** a hit test) **and** register **with** the "pick_event" callback. See examples/pick_event_demo.py **for** details

Bar, barh, **and** hist have "log" binary kwarg: log=**True** sets the ordinate to a log scale.

Boxplot can handle a **list** of vectors instead of just an array, so vectors can have different lengths.

Plot can handle 2-D x **and/or** y; it plots the columns.

Added linewidth kwarg to bar **and** barh.

Made the default Artist._transform **None** (rather than invoking identity_transform **for** each artist only to have it overridden later). Use artist.get_transform() rather than artist._transform, even **in** derived classes, so that the default transform will be created lazily **as** needed

New LogNorm subclass of Normalize added to colors.py. All Normalize subclasses have new inverse() method, **and** the `__call__()` method has a new clip kwarg.

Changed **class names** **in** colors.py to match convention: normalize -> Normalize, no_norm -> NoNorm. Old names are still available **for** now.

Removed obsolete pcolor_classic command **and** method.

Removed lineprops **and** markerprops **from** the Annotation code **and** replaced them **with** an arrow configurable **with** kwarg arrowprops. See examples/annotation_demo.py - JDH

30.24 Changes for 0.87.7

Completely reworked the annotations API because I found the old API cumbersome. The new design is much more legible and easy to read. See `matplotlib.text.Annotation` and `examples/annotation_demo.py`

`markeredgecolor` and `markerfacecolor` cannot be configured in `matplotlibrc` any more. Instead, markers are generally colored automatically based on the color of the line, unless marker colors are explicitly set as kwargs - NN

Changed default comment character for load to '#' - JDH

`math_parse_s_ft2font_svg` from `mathtext.py` & `mathtext2.py` now returns width, height, `svg_elements`. `svg_elements` is an instance of `Bunch` (`cmbook.py`) and has the attributes `svg_glyphs` and `svg_lines`, which are both lists.

`Renderer.draw_arc` now takes an additional parameter, `rotation`. It specifies to draw the artist rotated in degrees anti-clockwise. It was added for rotated ellipses.

Renamed `Figure.set_figsize_inches` to `Figure.set_size_inches` to better match the get method, `Figure.get_size_inches`.

Removed the `copy_bbox_transform` from `transforms.py`; added `shallowcopy` methods to all transforms. All transforms already had `deepcopy` methods.

`FigureManager.resize(width, height)`: resize the window specified in pixels

`barh`: `x` and `y` args have been renamed to `width` and `bottom` respectively, and their order has been swapped to maintain a (position, value) order.

`bar` and `barh`: now accept kwarg `'edgecolor'`.

`bar` and `barh`: The left, height, width and bottom args can now all be scalars or sequences; see docstring.

`barh`: now defaults to edge aligned instead of center aligned bars

`bar`, `barh` and `hist`: Added a keyword arg `'align'` that

(continues on next page)

(continued from previous page)

controls between edge **or** center bar alignment.

Collections: PolyCollection **and** LineCollection now accept vertices **or** segments either **in** the original form [(x,y), (x,y), ...] **or** as a 2D numerix array, **with** X **as** the first column **and** Y **as** the second. Contour **and** quiver output the numerix form. The transforms methods Bbox.update() **and** Transformation.seq_xy_tups() now accept either form.

Collections: LineCollection **is** now a ScalarMappable like PolyCollection, etc.

Specifying a grayscale color **as** a **float is** deprecated; use a string instead, e.g., 0.75 -> '0.75'.

Collections: initializers now accept **any** mpl color arg, **or** sequence of such args; previously only a sequence of rgba tuples was accepted.

Colorbar: completely new version **and** api; see docstring. The original version **is** still accessible **as** colorbar_classic, but **is** deprecated.

Contourf: "extend" kwarg replaces "clip_ends"; see docstring. Masked array support added to pcolormesh.

Modified aspect-ratio handling:

Removed aspect kwarg **from** **imshow**

Axes methods:

set_aspect(**self**, aspect, adjustable=**None**, anchor=**None**)

set_adjustable(**self**, adjustable)

set_anchor(**self**, anchor)

Pylab interface:

axis('image')

Backend developers: ft2font's load_char now takes a flags argument, which you can OR together **from** **the** LOAD_XXX constants.

30.25 Changes for 0.86

Matplotlib data **is** installed into the matplotlib module. This **is** similar to package_data. This should get rid of

(continues on next page)

(continued from previous page)

having to check for many possibilities in `_get_data_path()`.
The `MATPLOTLIBDATA` env key is still checked first to allow for flexibility.

- 1) Separated the color table data from `cm.py` out into a new file, `_cm.py`, to make it easier to find the actual code in `cm.py` and to add new colormaps. Everything from `_cm.py` is imported by `cm.py`, so the split should be transparent.
- 2) Enabled automatic generation of a colormap from a list of colors in contour; see modified `examples/contour_demo.py`.
- 3) Support for `imshow` of a masked array, with the ability to specify colors (or no color at all) for masked regions, and for regions that are above or below the normally mapped region. See `examples/image_masked.py`.
- 4) In support of the above, added two new classes, `ListedColormap`, and `no_norm`, to `colors.py`, and modified the `Colormap` class to include common functionality. Added a `clip` kwarg to the `normalize` class.

30.26 Changes for 0.85

Made `xtick` and `ytick` separate props in `rc`

made `pos=None` the default for tick formatters rather than 0 to indicate "not supplied"

Removed "feature" of minor ticks which prevents them from overlapping major ticks. Often you want major and minor ticks at the same place, and can offset the major ticks with the `pad`. This could be made configurable

Changed the internal structure of `contour.py` to a more OO style. Calls to `contour` or `contourf` in `axes.py` or `pylab.py` now return a `ContourSet` object which contains references to the `LineCollections` or `PolyCollections` created by the call, as well as the configuration variables that were used. The `ContourSet` object is a "mappable" if a colormap was used.

Added a `clip_ends` kwarg to `contourf`. From the docstring:

```
* clip_ends = True
```

(continues on next page)

(continued from previous page)

If `False`, the limits for color scaling are set to the minimum and maximum contour levels.
`True` (default) clips the scaling limits. Example:
 if the contour boundaries are `V = [-100, 2, 1, 0, 1, 2, 100]`,
 then the scaling limits will be `[-100, 100]` if `clip_ends`
 is `False`, and `[-3, 3]` if `clip_ends` is `True`.

Added kwargs `linewidths`, `antialiased`, and `nchunk` to `contourf`. These are experimental; see the docstring.

Changed `Figure.colorbar()`:

`kw` argument order changed;
 if mappable arg is a non-filled `ContourSet`, `colorbar()` shows lines instead of polygons.
 if mappable arg is a filled `ContourSet` with `clip_ends=True`, the endpoints are not labelled, so as to give the correct impression of open-endedness.

Changed `LineCollection.get_linewidths` to `get_linewidth`, for consistency.

30.27 Changes for 0.84

Unified argument handling between `hlines` and `vlines`. Both now take optionally a `fmt` argument (as in `plot`) and a keyword args that can be passed onto `Line2D`.

Removed all references to "data clipping" in `rc` and `lines.py` since these were not used and not optimized. I'm sure they'll be resurrected later with a better implementation when needed.

'`set`' removed - no more deprecation warnings. Use '`setp`' instead.

Backend developers: Added `flipud` method to `image` and removed it from `to_str`. Removed `origin` kwarg from `backend.draw_image`. `origin` is handled entirely by the frontend now.

30.28 Changes for 0.83

- Made `HOME/.matplotlib` the new config `dir` where the `matplotlibrc` file, the `ttf.cache`, and the `tex.cache` live. The new default filenames in `.matplotlib` have no leading dot and are not hidden.

(continues on next page)

(continued from previous page)

e.g., the new names are `matplotlibrc`, `tex.cache`, and `ttfont.cache`. This is how ipython does it so it must be right.

If old files are found, a warning is issued and they are moved to the new location.

- `backends/___init___`.py no longer imports `new_figure_manager`, `draw_if_interactive` and `show` from the default backend, but puts these imports into a call to `pylab_setup`. Also, the `Toolbar` is no longer imported from `WX/WXAgg`. New usage:

```
from backends import pylab_setup
new_figure_manager, draw_if_interactive, show = pylab_setup()
```

- Moved `Figure.get_width_height()` to `FigureCanvasBase`. It now returns `int` instead of `float`.

30.29 Changes for 0.82

- toolbar import change in `GTKAgg`, `GTKCairo` and `WXAgg`
- Added subplot config tool to `GTK*` backends -- note you must now import the `NavigationToolbar2` from your backend of choice rather than from `backend_gtk` because it needs to know about the backend specific canvas -- see `examples/embedding_in_gtk2.py`. Ditto for `wx` backend -- see `examples/embedding_in_wxagg.py`
- hist bin change

Sean Richards notes there was a problem in the way we created the binning for histogram, which made the last bin underrepresented. From his post:

I see that `hist` uses the `linspace` function to create the bins and then uses `searchsorted` to put the values in their correct bin. That's all good but I am confused over the use of `linspace` for the bin creation. I wouldn't have thought that it does what is needed, to quote the docstring it creates a "Linear spaced array from min to max". For it to work correctly shouldn't the values in the bins array be the same bound for each bin? (i.e. each value should be the lower bound of a bin). To provide the correct bins for `hist` would it not be

(continues on next page)

(continued from previous page)

something like

```
def bins(xmin, xmax, N):
    if N==1: return xmax
    dx = (xmax-xmin)/N # instead of N-1
    return xmin + dx*arange(N)
```

This suggestion is implemented in 0.81. My test script with these changes does not reveal any bias in the binning

```
from matplotlib.numerix.mlab import randn, rand, zeros, Float
from matplotlib.mlab import hist, mean
```

```
Nbins = 50
Ntests = 200
results = zeros((Ntests,Nbins), typecode=Float)
for i in range(Ntests):
    print 'computing', i
    x = rand(10000)
    n, bins = hist(x, Nbins)
    results[i] = n
print mean(results)
```

30.30 Changes for 0.81

- pylab **and** artist **"set"** functions renamed to **setp** to avoid clash with python2.4 built-in **set**. Current version will issue a deprecation warning which will be removed in future versions
- **imshow** interpolation arguments changes for advanced interpolation schemes. See help **imshow**, particularly the interpolation, **filtnorm** **and** **filterrad** kwargs
- Support for masked arrays has been added to the **plot** command **and** to the **Line2D** object. Only the valid points are plotted. A **"valid_only"** kwarg was added to the **get_xdata()** **and** **get_ydata()** methods of **Line2D**; by default it is **False**, so that the original data arrays are returned. Setting it to **True** returns the plottable points.
- **contour** changes:

(continues on next page)

(continued from previous page)

Masked arrays: `contour` and `contourf` now accept masked arrays as the variable to be contoured. Masking works correctly for `contour`, but a bug remains to be fixed before it will work for `contourf`. The `"badmask"` kwarg has been removed from both functions.

Level argument changes:

Old version: a list of levels as one of the positional arguments specified the lower bound of each filled region; the upper bound of the last region was taken as a very large number. Hence, it was not possible to specify that z values between 0 and 1, for example, be filled, and that values outside that range remain unfilled.

New version: a list of N levels is taken as specifying the boundaries of N-1 z ranges. Now the user has more control over what is colored and what is not. Repeated calls to `contourf` (with different colormaps or color specifications, for example) can be used to color different ranges of z. Values of z outside an expected range are left uncolored.

Example:

Old: `contourf(z, [0, 1, 2])` would yield 3 regions: 0-1, 1-2, and >2.

New: it would yield 2 regions: 0-1, 1-2. If the same 3 regions were desired, the equivalent list of levels would be `[0, 1, 2, 1e38]`.

30.31 Changes for 0.80

- `xlim/ylim/axis` always return the new limits regardless of arguments. They now take kwargs which allow you to selectively change the upper or lower limits while leaving unnamed limits unchanged. See `help(xlim)` for example

30.32 Changes for 0.73

- Removed deprecated `ColormapJet` and friends
- Removed all error handling from the verbose object
- figure num of zero is now allowed

30.33 Changes for 0.72

- Line2D, Text, and Patch copy `__properties` renamed `update_from` and moved into artist base class
- LineCollecitons.color renamed to LineCollections.set_color for consistency with set/get introspection mechanism,
- pylab figure now defaults to num=None, which creates a new figure with a guaranteed unique number
- contour method syntax changed - now it is MATLAB compatible

unchanged: `contour(Z)`
 old: `contour(Z, x=Y, y=Y)`
 new: `contour(X, Y, Z)`

see <http://matplotlib.sf.net/matplotlib.pylab.html#-contour>
- Increased the default resolution for save command.
- Renamed the base attribute of the ticker classes to `__base` to avoid conflict with the base method. Sitt for subs
- subs=None now does autosubbing in the tick locator.
- New subplots that overlap old will delete the old axes. If you do not want this behavior, use `fig.add_subplot` or the axes command

30.34 Changes for 0.71

Significant numerix namespace changes, introduced to resolve namespace clashes between python built-ins and mlab names. Refactored numerix to maintain separate modules, rather than folding all these names into a single namespace. See the following mailing list threads for more information and background

http://sourceforge.net/mailarchive/forum.php?thread_id=6398890&forum_id=36187
http://sourceforge.net/mailarchive/forum.php?thread_id=6323208&forum_id=36187

OLD usage

(continues on next page)

(continued from previous page)

```
from matplotlib.numerix import array, mean, fft
```

NEW usage

```
from matplotlib.numerix import array
from matplotlib.numerix.mlab import mean
from matplotlib.numerix.fft import fft
```

numerix dir structure mirrors numarray (though it is an incomplete implementation)

```
numerix
numerix/mlab
numerix/linear_algebra
numerix/fft
numerix/random_array
```

but of course you can use 'numerix : Numeric' and still get the symbols.

pylab still imports most of the symbols from Numerix, MLab, fft, etc, but is more cautious. For names that clash with python names (min, max, sum), pylab keeps the builtins and provides the numeric versions with an a* prefix, e.g., (amin, amax, asum)

30.35 Changes for 0.70

MplEvent factored into a base class `Event` and derived classes `MouseEvent` and `KeyEvent`

Removed defunct `set_measurement` in wx toolbar

30.36 Changes for 0.65.1

removed `add_axes` and `add_subplot` from `backend_bases`. Use `figure.add_axes` and `add_subplot` instead. The figure now manages the current axes with `gca` and `sca` for get and set current axes. If you have code you are porting which called, e.g., `figmanager.add_axes`, you can now simply do `figmanager.canvas.figure.add_axes`.

30.37 Changes for 0.65

`mpl_connect` and `mpl_disconnect` in the MATLAB interface renamed to `connect` and `disconnect`

Did away with the text methods `for_angle` since they were ambiguous. `fontangle` could mean `fontstyle` (oblique, etc) or the rotation of the text. Use style `angle` rotation instead.

30.38 Changes for 0.63

Dates are now represented internally as float days since 0001-01-01, UTC.

All date tickers and formatters are now in `matplotlib.dates`, rather than `matplotlib.tickers`

converters have been abolished from all functions and classes. `num2date` and `date2num` are now the converter functions for all date plots

Most of the date tick locators have a different meaning in their constructors. In the prior implementation, the first argument was a base and multiples of the base were ticked. e.g.,

```
HourLocator(5) # old: tick every 5 minutes
```

In the new implementation, the explicit points you want to tick are provided as a number or sequence

```
HourLocator(range(0,5,61)) # new: tick every 5 minutes
```

This gives much greater flexibility. I have tried to make the default constructors (no args) behave similarly, where possible.

Note that `YearLocator` still works under the base/multiple scheme. The difference between the `YearLocator` and the other locators is that years are not recurrent.

Financial functions:

```
matplotlib.finance.quotes_historical_yahoo(ticker, date1, date2)
```

(continues on next page)

(continued from previous page)

date1, date2 are now datetime instances. Return value is a list of quotes where the quote time is a float - days since gregorian start, as returned by date2num

See examples/finance_demo.py for example usage of new API

30.39 Changes for 0.61

canvas.connect is now deprecated for event handling. use mpl_connect and mpl_disconnect instead. The callback signature is func(event) rather than func(widget, event)

30.40 Changes for 0.60

ColormapJet and Grayscale are deprecated. For backwards compatibility, they can be obtained either by doing

```
from matplotlib.cm import ColormapJet
```

or

```
from matplotlib.matlab import *
```

They are replaced by cm.jet and cm.grey

30.41 Changes for 0.54.3

removed the set_default_font / get_default_font scheme from the font_manager to unify customization of font defaults with the rest of the rc scheme. See examples/font_properties_demo.py and help(rc) in matplotlib.matlab.

30.42 Changes for 0.54

30.42.1 MATLAB interface

dpi

Several of the backends used a `PIXELS_PER_INCH` hack that I added to try and make images render consistently across backends. This just complicated matters. So you may find that some font sizes and line widths appear different than before. Apologies for the inconvenience. You should set the dpi to an accurate value for your screen to get true sizes.

pcolor and scatter

There are two changes to the MATLAB interface API, both involving the patch drawing commands. For efficiency, `pcolor` and `scatter` have been rewritten to use polygon collections, which are a new set of objects from `matplotlib.collections` designed to enable efficient handling of large collections of objects. These new collections make it possible to build large scatter plots or `pcolor` plots with no loops at the python level, and are significantly faster than their predecessors. The original `pcolor` and `scatter` functions are retained as `pcolor_classic` and `scatter_classic`.

The return value from `pcolor` is a `PolyCollection`. Most of the properties that are available on rectangles or other patches are also available on `PolyCollections`, e.g., you can say:

```
c = scatter(blah, blah)
c.set_linewidth(1.0)
c.set_facecolor('r')
c.set_alpha(0.5)
```

or:

```
c = scatter(blah, blah)
set(c, 'linewidth', 1.0, 'facecolor', 'r', 'alpha', 0.5)
```

Because the collection is a single object, you no longer need to loop over the return value of `scatter` or `pcolor` to set properties for the entire list.

If you want the different elements of a collection to vary on a property, e.g., to have different line widths, see `matplotlib.collections` for a discussion on how to set the properties as a sequence.

For `scatter`, the size argument is now in points^2 (the area of the symbol in points) as in MATLAB and is not in data coords as before. Using sizes in data coords caused several problems. So you will need to adjust your size arguments accordingly or use `scatter_classic`.

mathtext spacing

For reasons not clear to me (and which I'll eventually fix) spacing no longer works in font groups. However, I added three new spacing commands which compensate for this " (regular space), "/" (small space) and `'hspace{frac}'` where `frac` is a fraction of fontsize in points. You will need to quote spaces in font strings, is:

```
title(r'$\rm{Histogram\ of\ IQ:}\ \mu=100,\ \sigma=15$')
```

30.42.2 Object interface - Application programmers

Autoscaling

The x and y axis instances no longer have autoscale view. These are handled by `axes.autoscale_view`

Axes creation

You should not instantiate your own Axes any more using the OO API. Rather, create a Figure as before and in place of:

```
f = Figure(figsize=(5,4), dpi=100)
a = Subplot(f, 111)
f.add_axis(a)
```

use:

```
f = Figure(figsize=(5,4), dpi=100)
a = f.add_subplot(111)
```

That is, `add_axis` no longer exists and is replaced by:

```
add_axes(rect, axisbg=defaultcolor, frameon=True)
add_subplot(num, axisbg=defaultcolor, frameon=True)
```

Artist methods

If you define your own Artists, you need to rename the `__draw` method to `draw`

Bounding boxes

`matplotlib.transforms.Bounds2D` is replaced by `matplotlib.transforms.Bbox`. If you want to construct a `bbox` from left, bottom, width, height (the signature for `Bounds2D`), use `matplotlib.transforms.lbwh_to_bbox`, as in

```
bbox = clickBBox = lbwh_to_bbox(left, bottom, width, height)
```

The `Bbox` has a different API than the `Bounds2D`. e.g., if you want to get the width and height of the `bbox`

OLD:: `width = fig.bbox.x.interval()` `height = fig.bbox.y.interval()`

New:: `width = fig.bbox.width()` `height = fig.bbox.height()`

Object constructors

You no longer pass the `bbox`, `dpi`, or `transforms` to the various Artist constructors. The old way of creating lines and rectangles was cumbersome because you had to pass so many attributes to the `Line2D` and `Rectangle` classes not related directly to the geometry and properties of the object. Now default values are added to the object when you call `axes.add_line` or `axes.add_patch`, so they are hidden from the user.

If you want to define a custom transformation on these objects, call `o.set_transform(trans)` where `trans` is a `Transformation` instance.

In prior versions of you wanted to add a custom line in data coords, you would have to do

```
l = Line2D(dpi, bbox, x, y, color = color, transx = transx, transy = transy,
           )
```

now all you need is

```
l = Line2D(x, y, color=color)
```

and the axes will set the transformation for you (unless you have set your own already, in which case it will leave it unchanged)

Transformations

The entire transformation architecture has been rewritten. Previously the `x` and `y` transformations were stored in the `xaxis` and `yaxis` instances. The problem with this approach is it only allows for separable transforms (where the `x` and `y` transformations don't depend on one another). But for cases like polar, they do. Now transformations operate on `x,y` together. There is a new base class `matplotlib.transforms.Transformation` and two concrete implementations, `matplotlib.transforms.SeparableTransformation` and `matplotlib.transforms.Affine`. The `SeparableTransformation` is constructed with the bounding box of the input (this determines the rectangular coordinate system of the input, i.e., the `x` and `y` view limits), the bounding box of the display, and possibly non-linear transformations of `x` and `y`. The 2 most frequently used transformations, data coordinates \rightarrow display and axes coordinates \rightarrow display are available as `ax.transData` and `ax.transAxes`. See `alignment_demo.py` which uses axes coords.

Also, the transformations should be much faster now, for two reasons

- they are written entirely in extension code
- because they operate on `x` and `y` together, they can do the entire transformation in one loop. Earlier I did something along the lines of:

```
xt = sx*func(x) + tx
yt = sy*func(y) + ty
```

Although this was done in `numerix`, it still involves 6 `length(x)` for-loops (the multiply, add, and function evaluation each for `x` and `y`). Now all of that is done in a single pass.

If you are using transformations and bounding boxes to get the cursor position in data coordinates, the method calls are a little different now. See the updated examples/coords_demo.py which shows you how to do this.

Likewise, if you are using the artist bounding boxes to pick items on the canvas with the GUI, the bbox methods are somewhat different. You will need to see the updated examples/object_picker.py.

See unit/transforms_unit.py for many examples using the new transformations.

30.43 Changes for 0.50

- * refactored Figure class so it is no longer backend dependent. FigureCanvasBackend takes over the backend specific duties of the Figure. matplotlib.backend_bases.FigureBase moved to matplotlib.figure.Figure.
- * backends must implement FigureCanvasBackend (the thing that controls the figure and handles the events if any) and FigureManagerBackend (wraps the canvas and the window for MATLAB interface). FigureCanvasBase implements a backend switching mechanism
- * Figure is now an Artist (like everything else in the figure) and is totally backend independent
- * GDFONTPATH renamed to TTFFPATH
- * backend faceColor argument changed to rgbFace
- * colormap stuff moved to colors.py
- * arg_to_rgb in backend_bases moved to class ColorConverter in colors.py
- * GD users must upgrade to gd-2.0.22 and gdmodule-0.52 since new gd features (clipping, antialiased lines) are now used.
- * Renderer must implement points_to_pixels

Migrating code:

MATLAB interface:

The only API change for those using the MATLAB interface is in how you call figure redraws for dynamically updating figures. In the

(continues on next page)

(continued from previous page)

old API, you did

```
fig.draw()
```

In the new API, you do

```
manager = get_current_fig_manager()
manager.canvas.draw()
```

See the examples `system_monitor.py`, `dynamic_demo.py`, and `anim.py`

API

There is one important API change for application developers. Figure instances used subclass GUI widgets that enabled them to be placed directly into figures. e.g., `FigureGTK` subclassed `gtk.DrawingArea`. Now the `Figure` class is independent of the backend, and `FigureCanvas` takes over the functionality formerly handled by `Figure`. In order to include figures into your apps, you now need to do, for example

```
# gtk example
fig = Figure(figsize=(5,4), dpi=100)
canvas = FigureCanvasGTK(fig) # a gtk.DrawingArea
canvas.show()
vbox.pack_start(canvas)
```

If you use the `NavigationToolbar`, this is now initialized with a `FigureCanvas`, not a `Figure`. The examples `embedding_in_gtk.py`, `embedding_in_gtk2.py`, and `mpl_with_glade.py` all reflect the new API so use these as a guide.

All prior calls to

```
figure.draw() and
figure.print_figure(args)
```

should now be

```
canvas.draw() and
canvas.print_figure(args)
```

Apologies for the inconvenience. This refactorization brings significant more freedom in developing matplotlib and should bring better plotting capabilities, so I hope the inconvenience is worth it.

30.44 Changes for 0.42

- * Refactoring AxisText to be backend independent. Text drawing and `get_window_extent` functionality will be moved to the `Renderer`.
- * `backend_bases.AxisTextBase` is now `text.Text` module
- * All the erase and reset functionality removed from `AxisText` - not needed with double buffered drawing. Ditto with state change. Text instances have a `get_prop_tup` method that returns a hashable tuple of text properties which you can use to see if text props have changed, e.g., by caching a font or layout instance in a dict with the prop tup as a key -- see `RendererGTK.get_pango_layout` in `backend_gtk` for an example.
- * `Text._get_xy_display` renamed `Text.get_xy_display`
- * Artist `set_renderer` and `wash_brushes` methods removed
- * Moved `Legend` class from `matplotlib.axes` into `matplotlib.legend`
- * Moved `Tick`, `XTick`, `YTick`, `Axis`, `XAxis`, `YAxis` from `matplotlib.axes` to `matplotlib.axis`
- * moved `process_text_args` to `matplotlib.text`
- * After getting Text handled in a backend independent fashion, the import process is much cleaner since there are no longer cyclic dependencies
- * `matplotlib.matlab._get_current_fig_manager` renamed to `matplotlib.matlab.get_current_fig_manager` to allow user access to the GUI window attribute, e.g., `figManager.window` for GTK and `figManager.frame` for wx

30.45 Changes for 0.40

- Artist
 - * `__init__` takes a DPI instance and a `Bound2D` instance which is the bounding box of the artist in display coords
 - * `get_window_extent` returns a `Bound2D` instance
 - * `set_size` is removed; replaced by `bbox` and `dpi`
 - * the `clip_gc` method is removed. Artists now clip themselves with their box

(continues on next page)

(continued from previous page)

- * added `_clipOn` boolean attribute. If True, gc clip to bbox.
- AxisTextBase
 - * Initialized with a `transx`, `transy` which are Transform instances
 - * `set_drawing_area` removed
 - * `get_left_right` and `get_top_bottom` are replaced by `get_window_extent`
- Line2D Patches now take `transx`, `transy`
 - * Initialized with a `transx`, `transy` which are Transform instances
- Patches
 - * Initialized with a `transx`, `transy` which are Transform instances
- FigureBase attributes `dpi` is a DPI instance rather than scalar and new attribute `bbox` is a `Bound2D` in display coords, and I got rid of the `left`, `width`, `height`, etc... attributes. These are now accessible as, for example, `bbox.x.min` is `left`, `bbox.x.interval()` is `width`, `bbox.y.max` is `top`, etc...
- GcfBase attribute `pagesize` renamed to `figsize`
- Axes
 - * removed `figbg` attribute
 - * added `fig` instance to `__init__`
 - * resizing is handled by figure call to `resize`.
- Subplot
 - * added `fig` instance to `__init__`
- Renderer methods for patches now take `gcEdge` and `gcFace` instances. `gcFace=None` takes the place of `filled=False`
- True and False symbols provided by `cbook` in a python2.3 compatible way
- new module `transforms` supplies `Bound1D`, `Bound2D` and `Transform` instances and more
- Changes to the MATLAB helpers API
 - * `_matplotlib_helpers.GcfBase` is renamed by `Gcf`. Backends no longer need to derive from this class. Instead, they provide a factory function `new_figure_manager(num, figsize, dpi)`. The `destroy` method of the `GcfDerived` from the backends is moved to the derived `FigureManager`.

(continues on next page)

(continued from previous page)

- * FigureManagerBase moved to backend_bases
- * Gcf.get_all_figwins renamed to Gcf.get_all_fig_managers

Jeremy:

Make sure to self._reset = False in AxisTextWX._set_font. This was something missing in my backend code.

THE TOP LEVEL MATPLOTLIB MODULE

`matplotlib.use(arg, warn=True, force=False)`

Set the matplotlib backend to one of the known backends.

The argument is case-insensitive. `warn` specifies whether a warning should be issued if a backend has already been set up. `force` is an experimental flag that tells matplotlib to attempt to initialize a new backend by reloading the backend module.

Note: This function must be called before importing pyplot for the first time; or, if you are not using pyplot, it must be called before importing `matplotlib.backends`. If `warn` is `True`, a warning is issued if you try and call this after `pylab` or `pyplot` have been loaded. In certain black magic use cases, e.g. `pyplot.switch_backend()`, we are doing the reloading necessary to make the backend switch work (in some cases, e.g., pure image backends) so one can set `warn=False` to suppress the warnings.

To find out which backend is currently set, see `matplotlib.get_backend()`.

`matplotlib.get_backend()`

Return the name of the current backend.

`matplotlib.rcParams`

An instance of `RcParams` for handling default matplotlib values.

`matplotlib.rc_context(rc=None, fname=None)`

Return a context manager for managing rc settings.

This allows one to do:

```
with mpl.rc_context(fname='screen.rc'):
    plt.plot(x, a)
    with mpl.rc_context(fname='print.rc'):
        plt.plot(x, b)
    plt.plot(x, c)
```

The 'a' vs 'x' and 'c' vs 'x' plots would have settings from 'screen.rc', while the 'b' vs 'x' plot would have settings from 'print.rc'.

A dictionary can also be passed to the context manager:

```
with mpl.rc_context(rc={'text.usetex': True}, fname='screen.rc'):
    plt.plot(x, a)
```

The 'rc' dictionary takes precedence over the settings loaded from 'fname'. Passing a dictionary only is also valid. For example a common usage is:

```
with mpl.rc_context(rc={'interactive': False}):
    fig, ax = plt.subplots()
    ax.plot(range(3), range(3))
    fig.savefig('A.png', format='png')
    plt.close(fig)
```

`matplotlib.rc(group, **kwargs)`

Set the current rc params. Group is the grouping for the rc, e.g., for `lines.linewidth` the group is `lines`, for `axes.facecolor`, the group is `axes`, and so on. Group may also be a list or tuple of group names, e.g., (`xtick`, `ytick`). `kwargs` is a dictionary attribute name/value pairs, e.g.,:

```
rc('lines', linewidth=2, color='r')
```

sets the current rc params and is equivalent to:

```
rcParams['lines.linewidth'] = 2
rcParams['lines.color'] = 'r'
```

The following aliases are available to save typing for interactive users:

Alias	Property
'lw'	'linewidth'
'ls'	'linestyle'
'c'	'color'
'fc'	'facecolor'
'ec'	'edgecolor'
'mew'	'markeredgewidth'
'aa'	'antialiased'

Thus you could abbreviate the above rc command as:

```
rc('lines', lw=2, c='r')
```

Note you can use python's `kwargs` dictionary facility to store dictionaries of default parameters. e.g., you can customize the font rc as follows:

```
font = {'family' : 'monospace',
        'weight' : 'bold',
        'size'   : 'larger'}
```

(continues on next page)

(continued from previous page)

```
rc('font', **font) # pass in the font dict as kwargs
```

This enables you to easily switch between several configurations. Use `matplotlib.style.use('default')` or `rcdefaults()` to restore the default rc params after changes.

`matplotlib.rc_file(fname)`

Update rc params from file.

`matplotlib.rcdefaults()`

Restore the rc params from Matplotlib's internal defaults.

See also:

`rc_file_defaults` Restore the rc params from the rc file originally loaded by Matplotlib.

`matplotlib.style.use` Use a specific style file. Call `style.use('default')` to restore the default style.

`matplotlib.rc_file_defaults()`

Restore the rc params from the original rc file loaded by Matplotlib.

`class matplotlib.RcParams(*args, **kwargs)`

A dictionary object including validation

validating functions are defined and associated with rc parameters in `matplotlib.rcsetup`

`find_all(pattern)`

Return the subset of this RcParams dictionary whose keys match, using `re.search()`, the given pattern.

Note: Changes to the returned dictionary are not propagated to the parent RcParams dictionary.

`msg_backend_obsolete = 'The {} rcParam was deprecated in version 2.2. In order to force the use of a spe`

`msg_depr = '%s is deprecated and replaced with %s; please use the latter.'`

`msg_depr_ignore = '%s is deprecated and ignored. Use %s instead.'`

`msg_depr_set = '%s is deprecated. Please remove it from your matplotlibrc and/or style files.'`

`msg_obsolete = '%s is obsolete. Please remove it from your matplotlibrc and/or style files.'`

`validate = {'_internal.classic_mode': <function validate_bool>, 'agg.path.chunksize': <function validate_`

`matplotlib.rc_params(fail_on_error=False)`

Return a `matplotlib.RcParams` instance from the default matplotlib rc file.

`matplotlib.rc_params_from_file(fname, fail_on_error=False, use_default_template=True)`

Return `matplotlib.RcParams` from the contents of the given file.

Parameters

`fname` [str] Name of file parsed for matplotlib settings.

`fail_on_error` [bool] If True, raise an error when the parser fails to convert a parameter.

`use_default_template` [bool] If True, initialize with default parameters before updating with those in the given file. If False, the configuration class only contains the parameters specified in the file. (Useful for updating dicts.)

`matplotlib.matplotlib_fname()`

Get the location of the config file.

The file location is determined in the following order

- `$PWD/matplotlibrc`
- `$MATPLOTLIBRC` if it is a file (or a named pipe, which can be created e.g. by process substitution)
- `$MATPLOTLIBRC/matplotlibrc`
- `$MPLCONFIGDIR/matplotlibrc`
- On Linux,
 - `$XDG_CONFIG_HOME/matplotlib/matplotlibrc` (if `$XDG_CONFIG_HOME` is defined)
 - or `$HOME/.config/matplotlib/matplotlibrc` (if `$XDG_CONFIG_HOME` is not defined)
- On other platforms,
 - `$HOME/.matplotlib/matplotlibrc` if `$HOME` is defined.
- Lastly, it looks in `$MATPLOTLIBDATA/matplotlibrc` for a system-defined copy.

`matplotlib.interactive(b)`

Set interactive mode to boolean `b`.

If `b` is True, then draw after every plotting command, e.g., after `xlabel`

`matplotlib.is_interactive()`

Return true if plot mode is interactive

AFM (ADOBE FONT METRICS INTERFACE)

32.1 matplotlib.afm

This is a python interface to Adobe Font Metrics Files. Although a number of other python implementations exist, and may be more complete than this, it was decided not to go with them because they were either:

- 1) copyrighted or used a non-BSD compatible license
- 2) had too many dependencies and a free standing lib was needed
- 3) Did more than needed and it was easier to write afresh rather than figure out how to get just what was needed.

It is pretty easy to use, and requires only built-in python libs:

```
>>> from matplotlib import rcParams
>>> import os.path
>>> afm_fname = os.path.join(rcParams['datapath'],
...                           'fonts', 'afm', 'ptmr8a.afm')
>>>
>>> from matplotlib.afm import AFM
>>> with open(afm_fname, 'rb') as fh:
...     afm = AFM(fh)
>>> afm.string_width_height('What the heck?')
(6220.0, 694)
>>> afm.get_fontname()
'Times-Roman'
>>> afm.get_kern_dist('A', 'f')
0
>>> afm.get_kern_dist('A', 'y')
-92.0
>>> afm.get_bbox_char('!')
[130, -9, 238, 676]
```

```
class matplotlib.afm.AFM(fh)
    Bases: object
    Parse the AFM file in file object fh
```


family_name

get_angle()

Return the fontangle as float

get_bbox_char(c, isord=False)

get_capheight()

Return the cap height as float

get_familyname()

Return the font family name, e.g., 'Times'

get_fontname()

Return the font name, e.g., 'Times-Roman'

get_fullname()

Return the font full name, e.g., 'Times-Roman'

get_height_char(c, isord=False)

Get the height of character c from the bounding box. This is the ink height (space is 0)

get_horizontal_stem_width()

Return the standard horizontal stem width as float, or None if not specified in AFM file.

get_kern_dist(c1, c2)

Return the kerning pair distance (possibly 0) for chars c1 and c2

get_kern_dist_from_name(name1, name2)

Return the kerning pair distance (possibly 0) for chars name1 and name2

get_name_char(c, isord=False)

Get the name of the character, i.e., ';' is 'semicolon'

get_str_bbox(s)

Return the string bounding box

get_str_bbox_and_descent(s)

Return the string bounding box

get_underline_thickness()

Return the underline thickness as float

get_vertical_stem_width()

Return the standard vertical stem width as float, or None if not specified in AFM file.

get_weight()

Return the font weight, e.g., 'Bold' or 'Roman'

get_width_char(c, isord=False)

Get the width of the character from the character metric WX field

get_width_from_char_name(name)

Get the width of the character from a type1 character name

`get_xheight()`

Return the xheight as float

`string_width_height(s)`

Return the string width (including kerning) and string height as a (w, h) tuple.

`matplotlib.afm.parse_afm(fh)`

Parse the Adobe Font Metrics file in file handle fh. Return value is a (dhead, dcmetrics_ascii, dmetrics_name, dkernpairs, dcomposite) tuple where dhead is a `__parse_header()` dict, dcmetrics_ascii and dmetrics_name are the two resulting dicts from `__parse_char_metrics()`, dkernpairs is a `__parse_kern_pairs()` dict (possibly {}) and dcomposite is a `__parse_composites()` dict (possibly {})

Table of Contents

- Animation
- Writer Classes
- Helper Classes
- Inheritance Diagrams

33.1 Animation

The easiest way to make a live animation in matplotlib is to use one of the Animation classes.

FuncAnimation	Makes an animation by repeatedly calling a function func.
ArtistAnimation	Animation using a fixed set of Artist objects.

33.1.1 matplotlib.animation.FuncAnimation

```
class matplotlib.animation.FuncAnimation(fig, func, frames=None, init_func=None,
                                         fargs=None, save_count=None, **kwargs)
```

Bases: matplotlib.animation.TimedAnimation

Makes an animation by repeatedly calling a function func.

Parameters

fig [matplotlib.figure.Figure] The figure object that is used to get draw, resize, and any other needed events.

func [callable] The function to call at each frame. The first argument will be the next value in frames. Any additional positional arguments can be supplied via the fargs parameter.

The required signature is:

```
def func(frame, *fargs) -> iterable_of_artists:
```

frames [iterable, int, generator function, or None, optional] Source of data to pass func and each frame of the animation

If an iterable, then simply use the values provided. If the iterable has a length, it will override the `save_count` kwarg.

If an integer, then equivalent to passing `range(frames)`

If a generator function, then must have the signature:

```
def gen_function() -> obj:
```

If None, then equivalent to passing `itertools.count`.

In all of these cases, the values in frames is simply passed through to the user-supplied func and thus can be of any type.

init_func [callable, optional] A function used to draw a clear frame. If not given, the results of drawing from the first item in the frames sequence will be used. This function will be called once before the first frame.

If `blit == True`, `init_func` must return an iterable of artists to be re-drawn.

The required signature is:

```
def init_func() -> iterable_of_artists:
```

fargs [tuple or None, optional] Additional arguments to pass to each call to func.

save_count [int, optional] The number of values from frames to cache.

interval [number, optional] Delay between frames in milliseconds. Defaults to 200.

repeat_delay [number, optional] If the animation is repeated, adds a delay in milliseconds before repeating the animation. Defaults to None.

repeat [bool, optional] Controls whether the animation should repeat when the sequence of frames is completed. Defaults to True.

blit [bool, optional] Controls whether blitting is used to optimize drawing. Defaults to False.

`new_frame_seq()`

Creates a new sequence of frame information.

`new_saved_frame_seq()`

Creates a new sequence of saved/cached frame information.

`save(filename, writer=None, fps=None, dpi=None, codec=None, bitrate=None, extra_args=None, metadata=None, extra_anim=None, savefig_kwargs=None)`
 Saves a movie file by drawing every frame.

Parameters

- `filename` [str] The output filename, e.g., `mymovie.mp4`.
- `writer` [MovieWriter or str, optional] A MovieWriter instance to use or a key that identifies a class to use, such as `'ffmpeg'`. If None, defaults to `rcParams["animation.writer"]`.
- `fps` [number, optional] Frames per second in the movie. Defaults to None, which will use the animation's specified interval to set the frames per second.
- `dpi` [number, optional] Controls the dots per inch for the movie frames. This combined with the figure's size in inches controls the size of the movie. If None, defaults to `rcParams["savefig.dpi"]`.
- `codec` [str, optional] The video codec to be used. Not all codecs are supported by a given MovieWriter. If None, default to `rcParams["animation.codec"]`.
- `bitrate` [number, optional] Specifies the number of bits used per second in the compressed movie, in kilobits per second. A higher number means a higher quality movie, but at the cost of increased file size. If None, defaults to `rcParams["animation.bitrate"]`.
- `extra_args` [list, optional] List of extra string arguments to be passed to the underlying movie utility. If None, defaults to `rcParams["animation.extra_args"]`.
- `metadata` [Dict[str, str], optional] Dictionary of keys and values for metadata to include in the output file. Some keys that may be of use include: `title`, `artist`, `genre`, `subject`, `copyright`, `srcform`, `comment`.
- `extra_anim` [list, optional] Additional Animation objects that should be included in the saved movie file. These need to be from the same `matplotlib.figure.Figure` instance. Also, animation frames will just be simply combined, so there should be a 1:1 correspondence between the frames from the different animations.
- `savefig_kwargs` [dict, optional] Is a dictionary containing keyword arguments to be passed on to the `savefig` command which is called repeatedly to save the individual frames.

Notes

`fps`, `codec`, `bitrate`, `extra_args`, `metadata` are used to construct a MovieWriter instance and can only be passed if `writer` is a string. If they are passed as non-None and `writer` is a MovieWriter, a `RuntimeError` will be raised.

`to_html5_video(embed_limit=None)`

Returns animation as an HTML5 video tag.

This saves the animation as an h264 video, encoded in base64 directly into the HTML5 video tag. This respects the rc parameters for the writer as well as the bitrate. This also makes use of the interval to control the speed, and uses the repeat parameter to decide whether to loop.

`to_jshtml(fps=None, embed_frames=True, default_mode=None)`

Generate HTML representation of the animation

Examples using `matplotlib.animation.FuncAnimation`

- Animated line plot
- Decay
- Animated histogram
- Oscilloscope
- The Bayes update
- MATPLOTLIB UNCHAINED
- Animated 3D random walk
- The double pendulum problem
- Rain simulation

33.1.2 `matplotlib.animation.ArtistAnimation`

`class matplotlib.animation.ArtistAnimation(fig, artists, *args, **kwargs)`

Bases: `matplotlib.animation.TimedAnimation`

Animation using a fixed set of Artist objects.

Before creating an instance, all plotting should have taken place and the relevant artists saved.

Parameters

`fig` [`matplotlib.figure.Figure`] The figure object that is used to get draw, resize, and any other needed events.

`artists` [list] Each list entry a collection of artists that represent what needs to be enabled on each frame. These will be disabled for other frames.

`interval` [number, optional] Delay between frames in milliseconds. Defaults to 200.

`repeat_delay` [number, optional] If the animation is repeated, adds a delay in milliseconds before repeating the animation. Defaults to None.

`repeat` [bool, optional] Controls whether the animation should repeat when the sequence of frames is completed. Defaults to True.

`blit` [bool, optional] Controls whether blitting is used to optimize drawing. Defaults to False.

`new_frame_seq()`

Creates a new sequence of frame information.

`new_saved_frame_seq()`

Creates a new sequence of saved/cached frame information.

`save(filename, writer=None, fps=None, dpi=None, codec=None, bitrate=None, extra_args=None, metadata=None, extra_anim=None, savefig_kwargs=None)`
Saves a movie file by drawing every frame.

Parameters

`filename` [str] The output filename, e.g., `mymovie.mp4`.

`writer` [MovieWriter or str, optional] A MovieWriter instance to use or a key that identifies a class to use, such as `'ffmpeg'`. If None, defaults to `rcParams["animation.writer"]`.

`fps` [number, optional] Frames per second in the movie. Defaults to None, which will use the animation's specified interval to set the frames per second.

`dpi` [number, optional] Controls the dots per inch for the movie frames. This combined with the figure's size in inches controls the size of the movie. If None, defaults to `rcParams["savefig.dpi"]`.

`codec` [str, optional] The video codec to be used. Not all codecs are supported by a given MovieWriter. If None, default to `rcParams["animation.codec"]`.

`bitrate` [number, optional] Specifies the number of bits used per second in the compressed movie, in kilobits per second. A higher number means a higher quality movie, but at the cost of increased file size. If None, defaults to `rcParams["animation.bitrate"]`.

`extra_args` [list, optional] List of extra string arguments to be passed to the underlying movie utility. If None, defaults to `rcParams["animation.extra_args"]`.

`metadata` [Dict[str, str], optional] Dictionary of keys and values for metadata to include in the output file. Some keys that may be of use include: `title`, `artist`, `genre`, `subject`, `copyright`, `srcform`, `comment`.

`extra_anim` [list, optional] Additional Animation objects that should be included in the saved movie file. These need to be from the same matplotlib.figure.Figure instance. Also, animation frames will just be simply combined, so there should be a 1:1 correspondence between the frames from the different animations.

`savefig_kwargs` [dict, optional] Is a dictionary containing keyword arguments to be passed on to the `savefig` command which is called repeatedly to save the individual frames.

Notes

`fps`, `codec`, `bitrate`, `extra_args`, `metadata` are used to construct a `MovieWriter` instance and can only be passed if `writer` is a string. If they are passed as non-`None` and `writer` is a `MovieWriter`, a `RuntimeError` will be raised.

`to_html5_video(embed_limit=None)`

Returns animation as an HTML5 video tag.

This saves the animation as an h264 video, encoded in base64 directly into the HTML5 video tag. This respects the rc parameters for the writer as well as the bitrate. This also makes use of the interval to control the speed, and uses the repeat parameter to decide whether to loop.

`to_jshtml(fps=None, embed_frames=True, default_mode=None)`

Generate HTML representation of the animation

Examples using `matplotlib.animation.ArtistAnimation`

- Animated image using a precomputed list of images

In both cases it is critical to keep a reference to the instance object. The animation is advanced by a timer (typically from the host GUI framework) which the Animation object holds the only reference to. If you do not hold a reference to the Animation object, it (and hence the timers), will be garbage collected which will stop the animation.

To save an animation to disk use `Animation.save` or `Animation.to_html5_video`

See Helper Classes below for details about what movie formats are supported.

33.1.3 FuncAnimation

The inner workings of `FuncAnimation` is more-or-less:

```
for d in frames:
    artists = func(d, *fargs)
    fig.canvas.draw_idle()
    fig.canvas.start_event_loop(interval)
```

with details to handle 'blitting' (to dramatically improve the live performance), to be non-blocking, not repeatedly start/stop the GUI event loop, handle repeats, multiple animated axes, and easily save the animation to a movie file.

'Blitting' is a [old technique](#) in computer graphics. The general gist is to take an existing bit map (in our case a mostly rasterized figure) and then 'blit' one more artist on top. Thus, by managing

a saved 'clean' bitmap, we can only re-draw the few artists that are changing at each frame and possibly save significant amounts of time. When using blitting (by passing `blit=True`) the core loop of `FuncAnimation` gets a bit more complicated

```
ax = fig.gca()

def update_blit(artists):
    fig.canvas.restore_region(bg_cache)
    for a in artists:
        a.axes.draw_artist(a)

    ax.figure.canvas.blit(ax.bbox)

artists = init_func()

for a in artists:
    a.set_animated(True)

fig.canvas.draw()
bg_cache = fig.canvas.copy_from_bbox(ax.bbox)

for f in frames:
    artists = func(f, *fargs)
    update_blit(artists)
    fig.canvas.start_event_loop(interval)
```

This is of course leaving out many details (such as updating the background when the figure is resized or fully re-drawn). However, this hopefully minimalist example gives a sense of how `init_func` and `func` are used inside of `FuncAnimation` and the theory of how 'blitting' works.

The expected signature on `func` and `init_func` is very simple to keep `FuncAnimation` out of your book keeping and plotting logic, but this means that the callable objects you pass in must know what artists they should be working on. There are several approaches to handling this, of varying complexity and encapsulation. The simplest approach, which works quite well in the case of a script, is to define the artist at a global scope and let Python sort things out. For example

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

fig, ax = plt.subplots()
xdata, ydata = [], []
ln, = plt.plot([], [], 'ro', animated=True)

def init():
    ax.set_xlim(0, 2*np.pi)
    ax.set_ylim(-1, 1)
    return ln,
```

(continues on next page)

(continued from previous page)

```
def update(frame):
    xdata.append(frame)
    ydata.append(np.sin(frame))
    ln.set_data(xdata, ydata)
    return ln,

ani = FuncAnimation(fig, update, frames=np.linspace(0, 2*np.pi, 128),
                    init_func=init, blit=True)
plt.show()
```

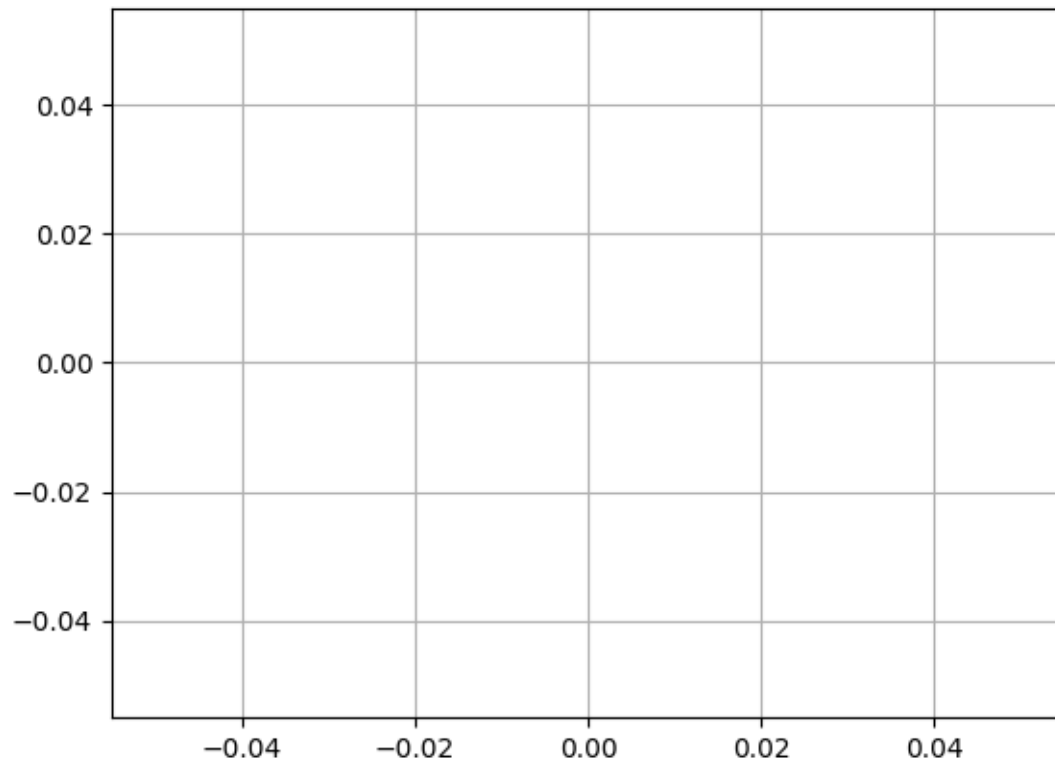
The second method is to use `functools.partial` to 'bind' artists to function. A third method is to use closures to build up the required artists and functions. A fourth method is to create a class.

Examples

Note: [Click here to download the full example code](#)

Decay

This example showcases: - using a generator to drive an animation, - changing axes limits during an animation.



```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

def data_gen(t=0):
    cnt = 0
    while cnt < 1000:
        cnt += 1
        t += 0.1
        yield t, np.sin(2*np.pi*t) * np.exp(-t/10.)

def init():
    ax.set_ylim(-1.1, 1.1)
    ax.set_xlim(0, 10)
    del xdata[:]
    del ydata[:]
    line.set_data(xdata, ydata)
    return line,
```

(continues on next page)

(continued from previous page)

```
fig, ax = plt.subplots()
line, = ax.plot([], [], lw=2)
ax.grid()
xdata, ydata = [], []

def run(data):
    # update the data
    t, y = data
    xdata.append(t)
    ydata.append(y)
    xmin, xmax = ax.get_xlim()

    if t >= xmax:
        ax.set_xlim(xmin, 2*xmax)
        ax.figure.canvas.draw()
    line.set_data(xdata, ydata)

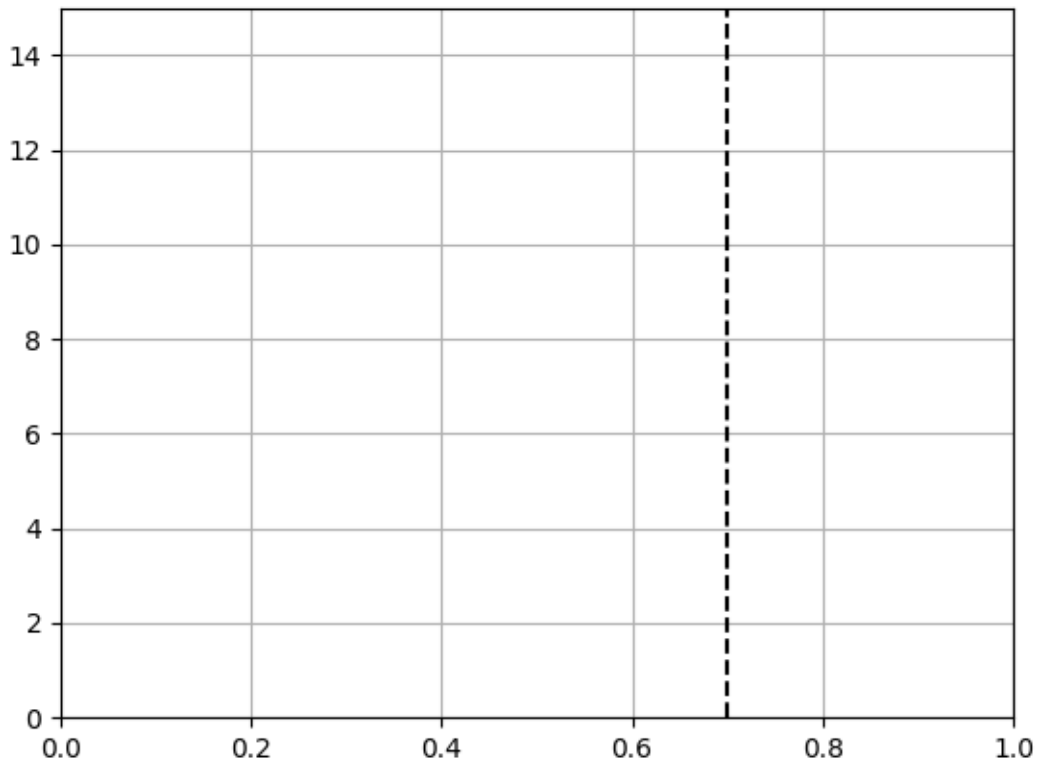
    return line,

ani = animation.FuncAnimation(fig, run, data_gen, blit=False, interval=10,
                             repeat=False, init_func=init)
plt.show()
```

Note: [Click here to download the full example code](#)

The Bayes update

This animation displays the posterior estimate updates as it is refitted when new data arrives. The vertical line represents the theoretical value to which the plotted distribution should converge.



```
import math

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

def beta_pdf(x, a, b):
    return (x**(a-1) * (1-x)**(b-1) * math.gamma(a + b)
           / (math.gamma(a) * math.gamma(b)))

class UpdateDist(object):
    def __init__(self, ax, prob=0.5):
        self.success = 0
        self.prob = prob
        self.line, = ax.plot([], [], 'k-')
        self.x = np.linspace(0, 1, 200)
        self.ax = ax
```

(continues on next page)

(continued from previous page)

```
# Set up plot parameters
self.ax.set_xlim(0, 1)
self.ax.set_ylim(0, 15)
self.ax.grid(True)

# This vertical line represents the theoretical value, to
# which the plotted distribution should converge.
self.ax.axvline(prob, linestyle='--', color='black')

def init(self):
    self.success = 0
    self.line.set_data([], [])
    return self.line,

def __call__(self, i):
    # This way the plot can continuously run and we just keep
    # watching new realizations of the process
    if i == 0:
        return self.init()

    # Choose success based on exceed a threshold with a uniform pick
    if np.random.rand(1,) < self.prob:
        self.success += 1
    y = beta_pdf(self.x, self.success + 1, (i - self.success) + 1)
    self.line.set_data(self.x, y)
    return self.line,

# Fixing random state for reproducibility
np.random.seed(19680801)

fig, ax = plt.subplots()
ud = UpdateDist(ax, prob=0.7)
anim = FuncAnimation(fig, ud, frames=np.arange(100), init_func=ud.init,
                    interval=100, blit=True)
plt.show()
```

Note: Click [here](#) to download the full example code

The double pendulum problem

This animation illustrates the double pendulum problem.

Double pendulum formula translated from the C code at <http://www.physics.usyd.edu.au/~wheat/>

`depend_html/solve_depend.c`

```

from numpy import sin, cos
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as integrate
import matplotlib.animation as animation

G = 9.8 # acceleration due to gravity, in m/s^2
L1 = 1.0 # length of pendulum 1 in m
L2 = 1.0 # length of pendulum 2 in m
M1 = 1.0 # mass of pendulum 1 in kg
M2 = 1.0 # mass of pendulum 2 in kg

def derivs(state, t):

    dydx = np.zeros_like(state)
    dydx[0] = state[1]

    del_ = state[2] - state[0]
    den1 = (M1 + M2)*L1 - M2*L1*cos(del_)*cos(del_)
    dydx[1] = (M2*L1*state[1]*state[1]*sin(del_)*cos(del_) +
               M2*G*sin(state[2])*cos(del_) +
               M2*L2*state[3]*state[3]*sin(del_) -
               (M1 + M2)*G*sin(state[0]))/den1

    dydx[2] = state[3]

    den2 = (L2/L1)*den1
    dydx[3] = (-M2*L2*state[3]*state[3]*sin(del_)*cos(del_) +
               (M1 + M2)*G*sin(state[0])*cos(del_) -
               (M1 + M2)*L1*state[1]*state[1]*sin(del_) -
               (M1 + M2)*G*sin(state[2]))/den2

    return dydx

# create a time array from 0..100 sampled at 0.05 second steps
dt = 0.05
t = np.arange(0.0, 20, dt)

# th1 and th2 are the initial angles (degrees)
# w10 and w20 are the initial angular velocities (degrees per second)
th1 = 120.0
w1 = 0.0
th2 = -10.0
w2 = 0.0

```

(continues on next page)

(continued from previous page)

```
# initial state
state = np.radians([th1, w1, th2, w2])

# integrate your ODE using scipy.integrate.
y = integrate.odeint(derivs, state, t)

x1 = L1*sin(y[:, 0])
y1 = -L1*cos(y[:, 0])

x2 = L2*sin(y[:, 2]) + x1
y2 = -L2*cos(y[:, 2]) + y1

fig = plt.figure()
ax = fig.add_subplot(111, autoscale_on=False, xlim=(-2, 2), ylim=(-2, 2))
ax.set_aspect('equal')
ax.grid()

line, = ax.plot([], [], 'o-', lw=2)
time_template = 'time = %.1fs'
time_text = ax.text(0.05, 0.9, '', transform=ax.transAxes)

def init():
    line.set_data([], [])
    time_text.set_text('')
    return line, time_text

def animate(i):
    thisx = [0, x1[i], x2[i]]
    thisy = [0, y1[i], y2[i]]

    line.set_data(thisx, thisy)
    time_text.set_text(time_template % (i*dt))
    return line, time_text

ani = animation.FuncAnimation(fig, animate, np.arange(1, len(y)),
                              interval=25, blit=True, init_func=init)

plt.show()
```

Note: [Click here to download the full example code](#)

Animated histogram

Use a path patch to draw a bunch of rectangles for an animated histogram.

```
import numpy as np

import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.path as path
import matplotlib.animation as animation

# Fixing random state for reproducibility
np.random.seed(19680801)

# histogram our data with numpy
data = np.random.randn(1000)
n, bins = np.histogram(data, 100)

# get the corners of the rectangles for the histogram
left = np.array(bins[:-1])
right = np.array(bins[1:])
bottom = np.zeros(len(left))
top = bottom + n
nrects = len(left)
```

Here comes the tricky part – we have to set up the vertex and path codes arrays using `plt.Path.MOVETO`, `plt.Path.LINETO` and `plt.Path.CLOSEPOLY` for each rect.

- We need 1 `MOVETO` per rectangle, which sets the initial point.
- We need 3 `LINETO`'s, which tell Matplotlib to draw lines from vertex 1 to vertex 2, v2 to v3, and v3 to v4.
- We then need one `CLOSEPOLY` which tells Matplotlib to draw a line from the v4 to our initial vertex (the `MOVETO` vertex), in order to close the polygon.

Note: The vertex for `CLOSEPOLY` is ignored, but we still need a placeholder in the `verts` array to keep the codes aligned with the vertices.

```
nverts = nrects * (1 + 3 + 1)
verts = np.zeros((nverts, 2))
codes = np.ones(nverts, int) * path.Path.LINETO
codes[0::5] = path.Path.MOVETO
codes[4::5] = path.Path.CLOSEPOLY
verts[0::5, 0] = left
verts[0::5, 1] = bottom
verts[1::5, 0] = left
```

(continues on next page)

(continued from previous page)

```
verts[1::5, 1] = top
verts[2::5, 0] = right
verts[2::5, 1] = top
verts[3::5, 0] = right
verts[3::5, 1] = bottom
```

To animate the histogram, we need an animate function, which generates a random set of numbers and updates the locations of the vertices for the histogram (in this case, only the heights of each rectangle). patch will eventually be a Patch object.

```
patch = None

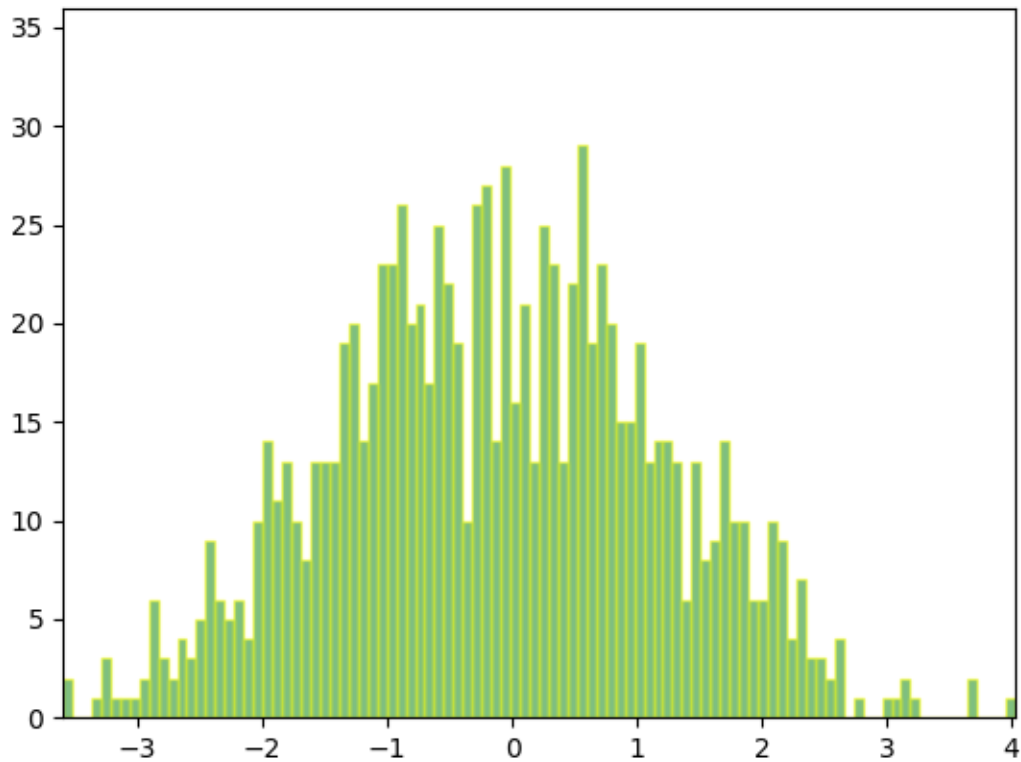
def animate(i):
    # simulate new data coming in
    data = np.random.randn(1000)
    n, bins = np.histogram(data, 100)
    top = bottom + n
    verts[1::5, 1] = top
    verts[2::5, 1] = top
    return [patch, ]
```

And now we build the Path and Patch instances for the histogram using our vertices and codes. We add the patch to the Axes instance, and setup the FuncAnimation with our animate function.

```
fig, ax = plt.subplots()
barpath = path.Path(verts, codes)
patch = patches.PathPatch(
    barpath, facecolor='green', edgecolor='yellow', alpha=0.5)
ax.add_patch(patch)

ax.set_xlim(left[0], right[-1])
ax.set_ylim(bottom.min(), top.max())

ani = animation.FuncAnimation(fig, animate, 100, repeat=False, blit=True)
plt.show()
```



Note: [Click here to download the full example code](#)

Rain simulation

Simulates rain drops on a surface by animating the scale and opacity of 50 scatter points.

Author: Nicolas P. Rougier

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# Fixing random state for reproducibility
np.random.seed(19680801)

# Create new Figure and an Axes which fills it.
fig = plt.figure(figsize=(7, 7))
ax = fig.add_axes([0, 0, 1, 1], frameon=False)
```

(continues on next page)

(continued from previous page)

```

ax.set_xlim(0, 1), ax.set_xticks([])
ax.set_ylim(0, 1), ax.set_yticks([])

# Create rain data
n_drops = 50
rain_drops = np.zeros(n_drops, dtype=[('position', float, 2),
                                       ('size', float, 1),
                                       ('growth', float, 1),
                                       ('color', float, 4)])

# Initialize the raindrops in random positions and with
# random growth rates.
rain_drops['position'] = np.random.uniform(0, 1, (n_drops, 2))
rain_drops['growth'] = np.random.uniform(50, 200, n_drops)

# Construct the scatter which we will update during animation
# as the raindrops develop.
scat = ax.scatter(rain_drops['position'][:, 0], rain_drops['position'][:, 1],
                  s=rain_drops['size'], lw=0.5, edgecolors=rain_drops['color'],
                  facecolors='none')

def update(frame_number):
    # Get an index which we can use to re-spawn the oldest raindrop.
    current_index = frame_number % n_drops

    # Make all colors more transparent as time progresses.
    rain_drops['color'][:, 3] -= 1.0/len(rain_drops)
    rain_drops['color'][:, 3] = np.clip(rain_drops['color'][:, 3], 0, 1)

    # Make all circles bigger.
    rain_drops['size'] += rain_drops['growth']

    # Pick a new position for oldest rain drop, resetting its size,
    # color and growth factor.
    rain_drops['position'][current_index] = np.random.uniform(0, 1, 2)
    rain_drops['size'][current_index] = 5
    rain_drops['color'][current_index] = (0, 0, 0, 1)
    rain_drops['growth'][current_index] = np.random.uniform(50, 200)

    # Update the scatter collection, with the new colors, sizes and positions.
    scat.set_edgecolors(rain_drops['color'])
    scat.set_sizes(rain_drops['size'])
    scat.set_offsets(rain_drops['position'])

```

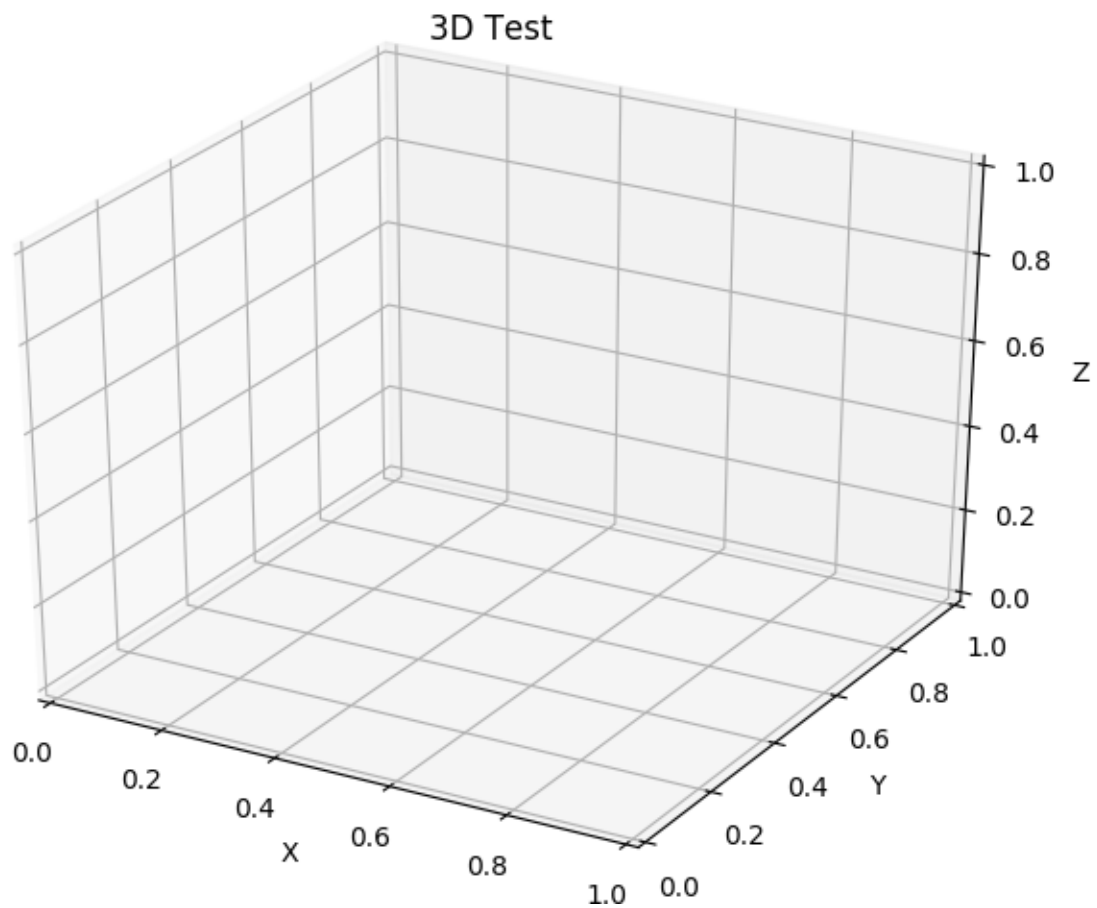
(continues on next page)

(continued from previous page)

```
# Construct the animation, using the update function as the animation director.  
animation = FuncAnimation(fig, update, interval=10)  
plt.show()
```

Note: [Click here to download the full example code](#)

Animated 3D random walk



```
import numpy as np  
import matplotlib.pyplot as plt  
import mpl_toolkits.mplot3d.axes3d as p3  
import matplotlib.animation as animation  
  
# Fixing random state for reproducibility  
np.random.seed(19680801)
```

(continues on next page)

(continued from previous page)

```

def Gen_RandLine(length, dims=2):
    """
    Create a line using a random walk algorithm

    length is the number of points for the line.
    dims is the number of dimensions the line has.
    """
    lineData = np.empty((dims, length))
    lineData[:, 0] = np.random.rand(dims)
    for index in range(1, length):
        # scaling the random numbers by 0.1 so
        # movement is small compared to position.
        # subtraction by 0.5 is to change the range to [-0.5, 0.5]
        # to allow a line to move backwards.
        step = ((np.random.rand(dims) - 0.5) * 0.1)
        lineData[:, index] = lineData[:, index - 1] + step

    return lineData

def update_lines(num, dataLines, lines):
    for line, data in zip(lines, dataLines):
        # NOTE: there is no .set_data() for 3 dim data...
        line.set_data(data[0:2, :num])
        line.set_3d_properties(data[2, :num])
    return lines

# Attaching 3D axis to the figure
fig = plt.figure()
ax = p3.Axes3D(fig)

# Fifty lines of random 3-D lines
data = [Gen_RandLine(25, 3) for index in range(50)]

# Creating fifty line objects.
# NOTE: Can't pass empty arrays into 3d version of plot()
lines = [ax.plot(dat[0, 0:1], dat[1, 0:1], dat[2, 0:1])[0] for dat in data]

# Setting the axes properties
ax.set_xlim3d([0.0, 1.0])
ax.set_xlabel('X')

ax.set_ylim3d([0.0, 1.0])

```

(continues on next page)

(continued from previous page)

```
ax.set_ylabel('Y')

ax.set_zlim3d([0.0, 1.0])
ax.set_zlabel('Z')

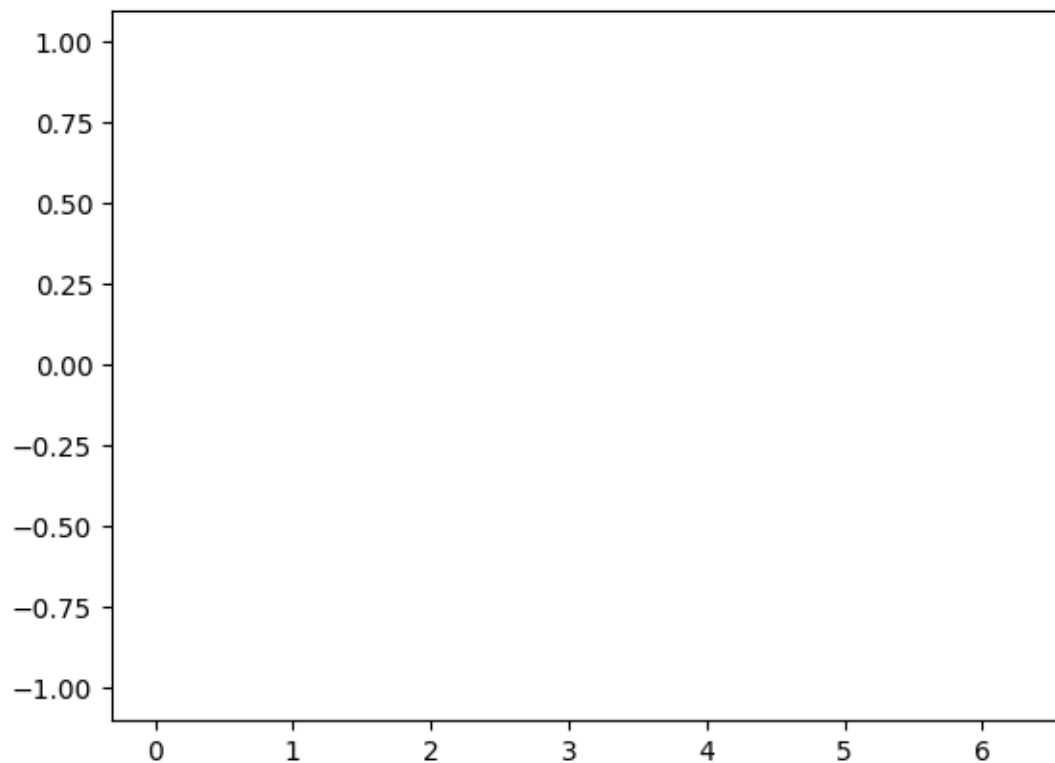
ax.set_title('3D Test')

# Creating the Animation object
line_ani = animation.FuncAnimation(fig, update_lines, 25, fargs=(data, lines),
                                   interval=50, blit=False)

plt.show()
```

Note: [Click here to download the full example code](#)

Animated line plot



```

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

fig, ax = plt.subplots()

x = np.arange(0, 2*np.pi, 0.01)
line, = ax.plot(x, np.sin(x))

def init(): # only required for blitting to give a clean slate.
    line.set_ydata([np.nan] * len(x))
    return line,

def animate(i):
    line.set_ydata(np.sin(x + i / 100)) # update the data.
    return line,

ani = animation.FuncAnimation(
    fig, animate, init_func=init, interval=2, blit=True, save_count=50)

# To save the animation, use e.g.
#
# ani.save("movie.mp4")
#
# or
#
# from matplotlib.animation import FFMpegWriter
# writer = FFMpegWriter(fps=15, metadata=dict(artist='Me'), bitrate=1800)
# ani.save("movie.mp4", writer=writer)

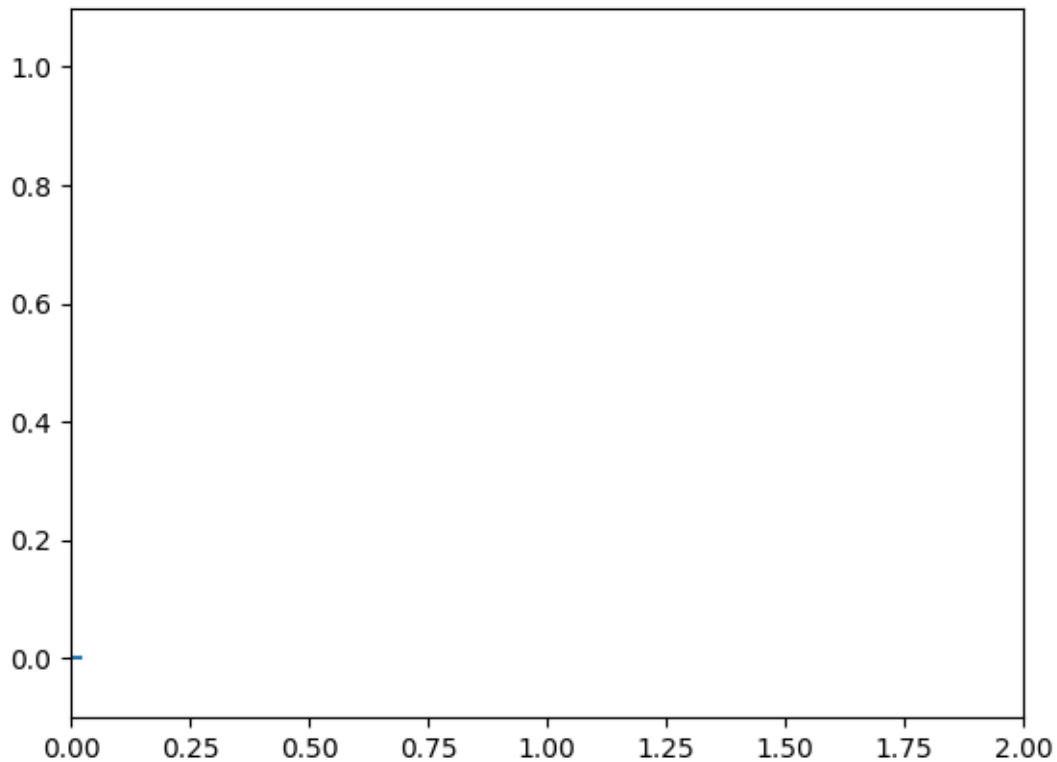
plt.show()

```

Note: [Click here to download the full example code](#)

Oscilloscope

Emulates an oscilloscope.



```
import numpy as np
from matplotlib.lines import Line2D
import matplotlib.pyplot as plt
import matplotlib.animation as animation

class Scope(object):
    def __init__(self, ax, maxt=2, dt=0.02):
        self.ax = ax
        self.dt = dt
        self.maxt = maxt
        self.tdata = [0]
        self.ydata = [0]
        self.line = Line2D(self.tdata, self.ydata)
        self.ax.add_line(self.line)
        self.ax.set_ylim(-.1, 1.1)
        self.ax.set_xlim(0, self.maxt)

    def update(self, y):
        lastt = self.tdata[-1]
```

(continues on next page)

(continued from previous page)

```

    if lastt > self.tdata[0] + self.maxt: # reset the arrays
        self.tdata = [self.tdata[-1]]
        self.ydata = [self.ydata[-1]]
        self.ax.set_xlim(self.tdata[0], self.tdata[0] + self.maxt)
        self.ax.figure.canvas.draw()

    t = self.tdata[-1] + self.dt
    self.tdata.append(t)
    self.ydata.append(y)
    self.line.set_data(self.tdata, self.ydata)
    return self.line,

def emitter(p=0.03):
    'return a random value with probability p, else 0'
    while True:
        v = np.random.rand(1)
        if v > p:
            yield 0.
        else:
            yield np.random.rand(1)

# Fixing random state for reproducibility
np.random.seed(19680801)

fig, ax = plt.subplots()
scope = Scope(ax)

# pass a generator in "emitter" to produce data for the update func
ani = animation.FuncAnimation(fig, scope.update, emitter, interval=10,
                              blit=True)

plt.show()

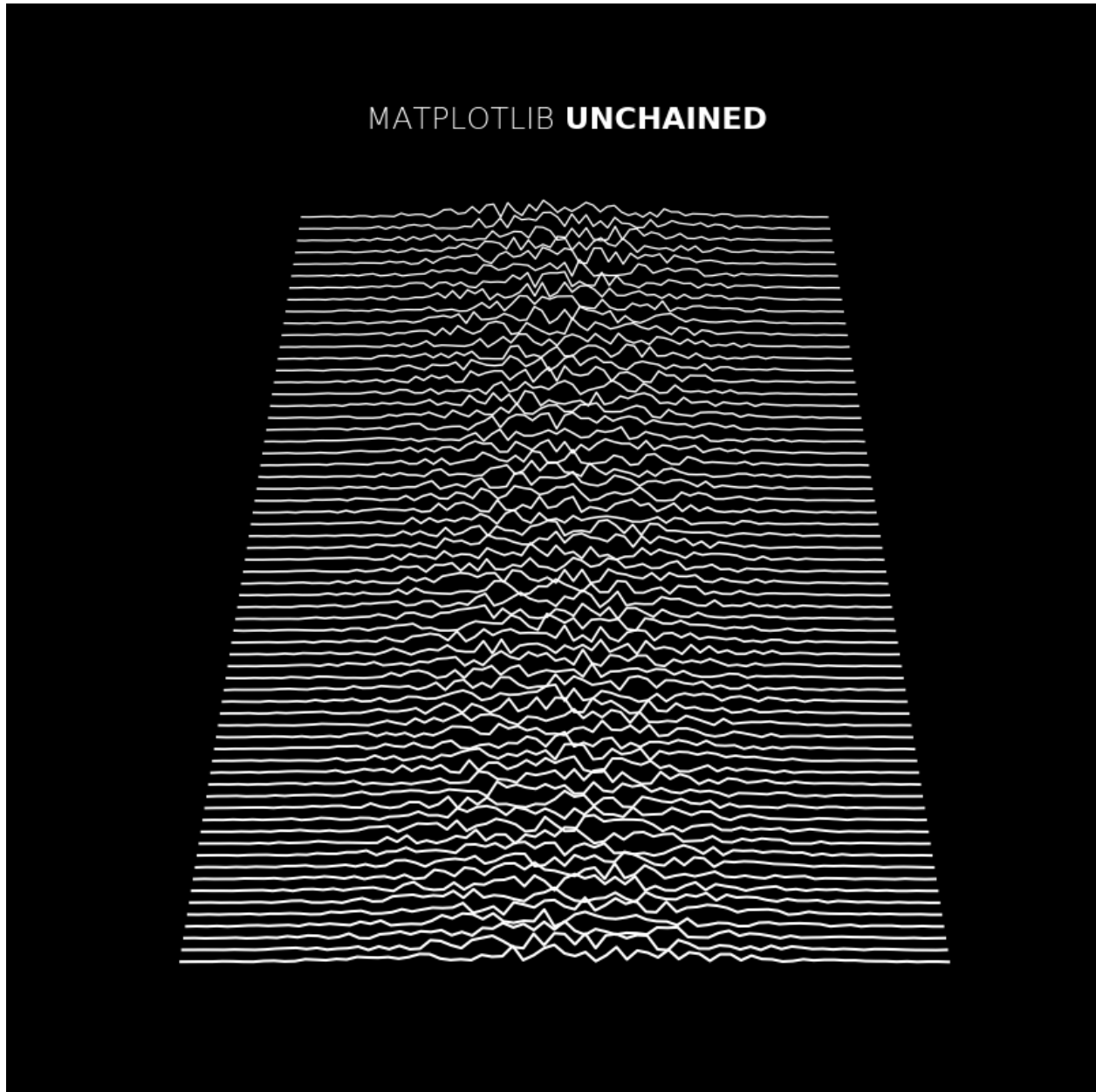
```

Note: [Click here to download the full example code](#)

MATPLOTLIB UNCHAINED

Comparative path demonstration of frequency from a fake signal of a pulsar (mostly known because of the cover for Joy Division's Unknown Pleasures).

Author: Nicolas P. Rougier



```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# Fixing random state for reproducibility
np.random.seed(19680801)

# Create new Figure with black background
fig = plt.figure(figsize=(8, 8), facecolor='black')
```

(continues on next page)

(continued from previous page)

```

# Add a subplot with no frame
ax = plt.subplot(111, frameon=False)

# Generate random data
data = np.random.uniform(0, 1, (64, 75))
X = np.linspace(-1, 1, data.shape[-1])
G = 1.5 * np.exp(-4 * X ** 2)

# Generate line plots
lines = []
for i in range(len(data)):
    # Small reduction of the X extents to get a cheap perspective effect
    xscale = 1 - i / 200.
    # Same for linewidth (thicker strokes on bottom)
    lw = 1.5 - i / 100.0
    line, = ax.plot(xscale * X, i + G * data[i], color="w", lw=lw)
    lines.append(line)

# Set y limit (or first line is cropped because of thickness)
ax.set_ylim(-1, 70)

# No ticks
ax.set_xticks([])
ax.set_yticks([])

# 2 part titles to get different font weights
ax.text(0.5, 1.0, "MATPLOTLIB ", transform=ax.transAxes,
        ha="right", va="bottom", color="w",
        family="sans-serif", fontweight="light", fontsize=16)
ax.text(0.5, 1.0, "UNCHAINED", transform=ax.transAxes,
        ha="left", va="bottom", color="w",
        family="sans-serif", fontweight="bold", fontsize=16)

def update(*args):
    # Shift all data to the right
    data[:, 1:] = data[:, :-1]

    # Fill-in new values
    data[:, 0] = np.random.uniform(0, 1, len(data))

    # Update data
    for i in range(len(data)):
        lines[i].set_ydata(i + G * data[i])

```

(continues on next page)

(continued from previous page)

```
# Return modified artists
return lines

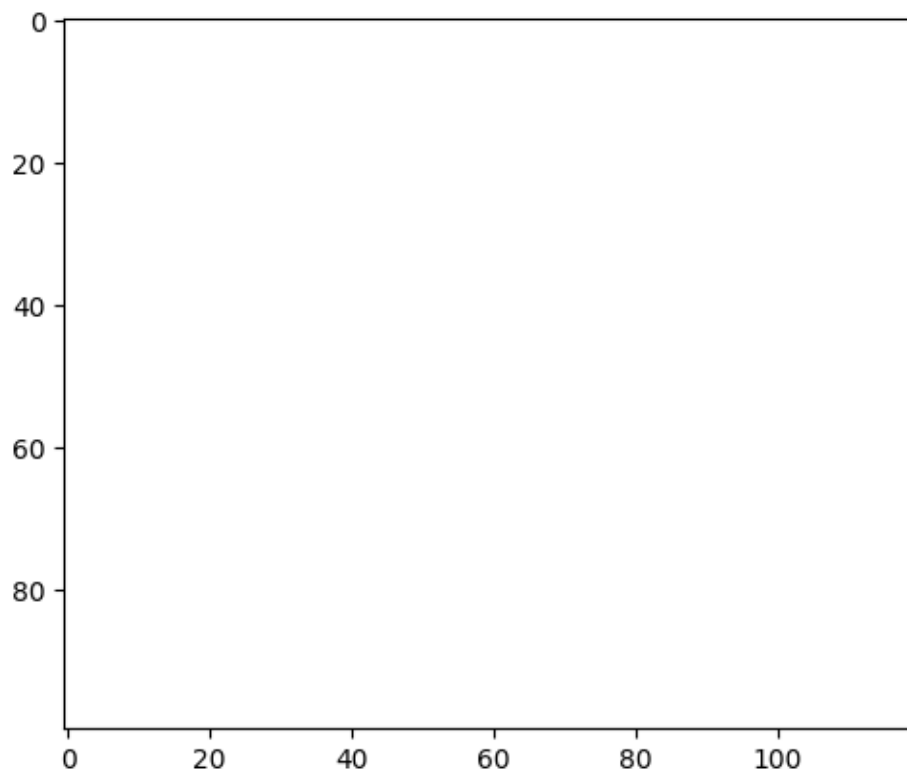
# Construct the animation, using the update function as the animation director.
anim = animation.FuncAnimation(fig, update, interval=10)
plt.show()
```

33.1.4 ArtistAnimation

Examples

Note: [Click here to download the full example code](#)

Animated image using a precomputed list of images



```

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

fig = plt.figure()

def f(x, y):
    return np.sin(x) + np.cos(y)

x = np.linspace(0, 2 * np.pi, 120)
y = np.linspace(0, 2 * np.pi, 100).reshape(-1, 1)
# ims is a list of lists, each row is a list of artists to draw in the
# current frame; here we are just animating one artist, the image, in
# each frame
ims = []
for i in range(60):
    x += np.pi / 15.
    y += np.pi / 20.
    im = plt.imshow(f(x, y), animated=True)
    ims.append([im])

ani = animation.ArtistAnimation(fig, ims, interval=50, blit=True,
                                repeat_delay=1000)

# To save the animation, use e.g.
#
# ani.save("movie.mp4")
#
# or
#
# from matplotlib.animation import FFMpegWriter
# writer = FFMpegWriter(fps=15, metadata=dict(artist='Me'), bitrate=1800)
# ani.save("movie.mp4", writer=writer)

plt.show()

```

33.2 Writer Classes

The provided writers fall into a few broad categories.

The Pillow writer relies on the Pillow library to write the animation, keeping all data in memory.

PillowWriter

33.2.1 matplotlib.animation.PillowWriter

class matplotlib.animation.PillowWriter(*args, **kwargs)

Bases: matplotlib.animation.MovieWriter

classmethod bin_path()

Returns the binary path to the commandline tool used by a specific subclass. This is a class method so that the tool can be looked for before making a particular MovieWriter subclass available.

cleanup()

Clean-up and collect the process used to write the movie file.

finish()

Finish any processing for writing the movie.

frame_size

A tuple (width, height) in pixels of a movie frame.

grab_frame(**savefig_kwargs)

Grab the image information from the figure and save as a movie frame.

All keyword arguments in savefig_kwargs are passed on to the savefig command that saves the figure.

classmethod isAvailable()

Check to see if a MovieWriter subclass is actually available by running the commandline tool.

saving(fig, outfile, dpi, *args, **kwargs)

Context manager to facilitate writing the movie file.

*args, **kw are any parameters that should be passed to setup.

setup(fig, outfile, dpi=None)

Perform setup for writing the movie file.

Parameters

fig [matplotlib.figure.Figure] The figure object that contains the information for frames

outfile [string] The filename of the resulting movie file

dpi [int, optional] The DPI (or resolution) for the file. This controls the size in pixels of the resulting movie file. Default is fig.dpi.

The pipe-based writers stream the captured frames over a pipe to an external process. The pipe-based variants tend to be more performant, but may not work on all systems.

FFMpegWriter	Pipe-based ffmpeg writer.
ImageMagickWriter	Pipe-based animated gif.
AVConvWriter	Pipe-based avconv writer.

33.2.2 matplotlib.animation.FFMpegWriter

```
class matplotlib.animation.FFMpegWriter(fps=5, codec=None, bitrate=None, extra_args=None, metadata=None)
```

Bases: matplotlib.animation.FFMpegBase, matplotlib.animation.MovieWriter

Pipe-based ffmpeg writer.

Frames are streamed directly to ffmpeg via a pipe and written in a single pass.

Parameters

`fps`: int Framerate for movie.

`codec`: string or None, optional The codec to use. If None (the default) the `animation.codec rcParam` is used.

`bitrate`: int or None, optional The bitrate for the saved movie file, which is one way to control the output file size and quality. The default value is None, which uses the `animation.bitrate rcParam`. A value of -1 implies that the bitrate should be determined automatically by the underlying utility.

`extra_args`: list of strings or None, optional A list of extra string arguments to be passed to the underlying movie utility. The default is None, which passes the additional arguments in the `animation.extra_args rcParam`.

`metadata`: Dict[str, str] or None A dictionary of keys and values for metadata to include in the output file. Some keys that may be of use include: title, artist, genre, subject, copyright, srcform, comment.

`args_key` = 'animation.ffmpeg_args'

`classmethod bin_path()`

Returns the binary path to the commandline tool used by a specific subclass. This is a class method so that the tool can be looked for before making a particular MovieWriter subclass available.

`cleanup()`

Clean-up and collect the process used to write the movie file.

`exec_key` = 'animation.ffmpeg_path'

`finish()`

Finish any processing for writing the movie.

`frame_size`

A tuple (width, height) in pixels of a movie frame.

`grab_frame(**savefig_kwargs)`

Grab the image information from the figure and save as a movie frame.

All keyword arguments in `savefig_kwargs` are passed on to the `savefig` command that saves the figure.

classmethod `isAvailable()`

Check to see if a `MovieWriter` subclass is actually available by running the commandline tool.

`output_args`

`saving(fig, outfile, dpi, *args, **kwargs)`

Context manager to facilitate writing the movie file.

`*args, **kw` are any parameters that should be passed to `setup`.

`setup(fig, outfile, dpi=None)`

Perform setup for writing the movie file.

Parameters

`fig` [`matplotlib.figure.Figure`] The figure object that contains the information for frames

`outfile` [`string`] The filename of the resulting movie file

`dpi` [`int`, optional] The DPI (or resolution) for the file. This controls the size in pixels of the resulting movie file. Default is `fig.dpi`.

Examples using `matplotlib.animation.FFMpegWriter`

- Frame grabbing

33.2.3 `matplotlib.animation.ImageMagickWriter`

`class matplotlib.animation.ImageMagickWriter(fps=5, codec=None, bitrate=None, extra_args=None, metadata=None)`

Bases: `matplotlib.animation.ImageMagickBase`, `matplotlib.animation.MovieWriter`

Pipe-based animated gif.

Frames are streamed directly to ImageMagick via a pipe and written in a single pass.

Parameters

`fps`: `int` Framerate for movie.

`codec`: `string` or `None`, optional The codec to use. If `None` (the default) the `animation.codec rcParam` is used.

`bitrate`: `int` or `None`, optional The bitrate for the saved movie file, which is one way to control the output file size and quality. The default value is `None`, which uses the `animation.bitrate rcParam`. A value of `-1` implies that the bitrate should be determined automatically by the underlying utility.

`extra_args`: `list of strings` or `None`, optional A list of extra string arguments to be passed to the underlying movie utility. The default is `None`, which passes the additional arguments in the `animation.extra_args rcParam`.

metadata: Dict[str, str] or None A dictionary of keys and values for metadata to include in the output file. Some keys that may be of use include: title, artist, genre, subject, copyright, srcform, comment.

args_key = 'animation.convert_args'

classmethod bin_path()

Returns the binary path to the commandline tool used by a specific subclass. This is a class method so that the tool can be looked for before making a particular MovieWriter subclass available.

cleanup()

Clean-up and collect the process used to write the movie file.

delay

exec_key = 'animation.convert_path'

finish()

Finish any processing for writing the movie.

frame_size

A tuple (width, height) in pixels of a movie frame.

grab_frame(**savefig_kwargs)

Grab the image information from the figure and save as a movie frame.

All keyword arguments in savefig_kwargs are passed on to the savefig command that saves the figure.

classmethod isAvailable()

Check to see if a ImageMagickWriter is actually available.

Done by first checking the windows registry (if applicable) and then running the commandline tool.

output_args

saving(fig, outfile, dpi, *args, **kwargs)

Context manager to facilitate writing the movie file.

*args, **kw are any parameters that should be passed to setup.

setup(fig, outfile, dpi=None)

Perform setup for writing the movie file.

Parameters

fig [matplotlib.figure.Figure] The figure object that contains the information for frames

outfile [string] The filename of the resulting movie file

`dpi` [int, optional] The DPI (or resolution) for the file. This controls the size in pixels of the resulting movie file. Default is `fig.dpi`.

33.2.4 `matplotlib.animation.AVConvWriter`

`class matplotlib.animation.AVConvWriter(fps=5, codec=None, bitrate=None, extra_args=None, metadata=None)`

Bases: `matplotlib.animation.AVConvBase`, `matplotlib.animation.FFMpegWriter`

Pipe-based avconv writer.

Frames are streamed directly to avconv via a pipe and written in a single pass.

Parameters

`fps`: int Framerate for movie.

`codec`: string or None, optional The codec to use. If None (the default) the `animation.codec rcParam` is used.

`bitrate`: int or None, optional The bitrate for the saved movie file, which is one way to control the output file size and quality. The default value is None, which uses the `animation.bitrate rcParam`. A value of -1 implies that the bitrate should be determined automatically by the underlying utility.

`extra_args`: list of strings or None, optional A list of extra string arguments to be passed to the underlying movie utility. The default is None, which passes the additional arguments in the `animation.extra_args rcParam`.

`metadata`: Dict[str, str] or None A dictionary of keys and values for metadata to include in the output file. Some keys that may be of use include: title, artist, genre, subject, copyright, srcform, comment.

`args_key` = 'animation.avconv_args'

`classmethod bin_path()`

Returns the binary path to the commandline tool used by a specific subclass. This is a class method so that the tool can be looked for before making a particular `MovieWriter` subclass available.

`cleanup()`

Clean-up and collect the process used to write the movie file.

`exec_key` = 'animation.avconv_path'

`finish()`

Finish any processing for writing the movie.

`frame_size`

A tuple (width, height) in pixels of a movie frame.

`grab_frame(**savefig_kwargs)`

Grab the image information from the figure and save as a movie frame.

All keyword arguments in `savefig_kwargs` are passed on to the `savefig` command that saves the figure.

`classmethod isAvailable()`

Check to see if a `MovieWriter` subclass is actually available by running the commandline tool.

`output_args`

`saving(fig, outfile, dpi, *args, **kwargs)`

Context manager to facilitate writing the movie file.

`*args, **kw` are any parameters that should be passed to `setup`.

`setup(fig, outfile, dpi=None)`

Perform setup for writing the movie file.

Parameters

`fig` [`matplotlib.figure.Figure`] The figure object that contains the information for frames

`outfile` [`string`] The filename of the resulting movie file

`dpi` [`int`, optional] The DPI (or resolution) for the file. This controls the size in pixels of the resulting movie file. Default is `fig.dpi`.

The file-based writers save temporary files for each frame which are stitched into a single file at the end. Although slower, these writers can be easier to debug.

<code>FFMpegFileWriter</code>	File-based ffmpeg writer.
<code>ImageMagickFileWriter</code>	File-based animated gif writer.
<code>AVConvFileWriter</code>	File-based avconv writer.

33.2.5 matplotlib.animation.FFMpegFileWriter

`class matplotlib.animation.FFMpegFileWriter(*args, **kwargs)`

Bases: `matplotlib.animation.FFMpegBase`, `matplotlib.animation.FileMovieWriter`

File-based ffmpeg writer.

Frames are written to temporary files on disk and then stitched together at the end.

`args_key = 'animation.ffmpeg_args'`

`classmethod bin_path()`

Returns the binary path to the commandline tool used by a specific subclass. This is a class method so that the tool can be looked for before making a particular `MovieWriter` subclass available.

`cleanup()`

Clean-up and collect the process used to write the movie file.

`exec_key = 'animation.ffmpeg_path'`

`finish()`

Finish any processing for writing the movie.

`frame_format`

Format (png, jpeg, etc.) to use for saving the frames, which can be decided by the individual subclasses.

`frame_size`

A tuple (width, height) in pixels of a movie frame.

`grab_frame(**savefig_kwargs)`

Grab the image information from the figure and save as a movie frame. All keyword arguments in `savefig_kwargs` are passed on to the `savefig` command that saves the figure.

`classmethod isAvailable()`

Check to see if a `MovieWriter` subclass is actually available by running the commandline tool.

`output_args`

`saving(fig, outfile, dpi, *args, **kwargs)`

Context manager to facilitate writing the movie file.

`*args, **kw` are any parameters that should be passed to `setup`.

`setup(fig, outfile, dpi=None, frame_prefix='_tmp', clear_temp=True)`

Perform setup for writing the movie file.

Parameters

`fig` [`matplotlib.figure.Figure`] The figure to grab the rendered frames from.

`outfile` [`str`] The filename of the resulting movie file.

`dpi` [`number`, optional] The dpi of the output file. This, with the figure size, controls the size in pixels of the resulting movie file. Default is `fig.dpi`.

`frame_prefix` [`str`, optional] The filename prefix to use for temporary files. Defaults to `'_tmp'`.

`clear_temp` [`bool`, optional] If the temporary files should be deleted after stitching the final result. Setting this to `False` can be useful for debugging. Defaults to `True`.

`supported_formats = ['png', 'jpeg', 'ppm', 'tiff', 'sgi', 'bmp', 'pbm', 'raw', 'rgba']`

33.2.6 matplotlib.animation.ImageMagickFileWriter

class matplotlib.animation.ImageMagickFileWriter(*args, **kwargs)

Bases: matplotlib.animation.ImageMagickBase, matplotlib.animation.FileMovieWriter

File-based animated gif writer.

Frames are written to temporary files on disk and then stitched together at the end.

args_key = 'animation.convert_args'

classmethod bin_path()

Returns the binary path to the commandline tool used by a specific subclass. This is a class method so that the tool can be looked for before making a particular MovieWriter subclass available.

cleanup()

Clean-up and collect the process used to write the movie file.

delay

exec_key = 'animation.convert_path'

finish()

Finish any processing for writing the movie.

frame_format

Format (png, jpeg, etc.) to use for saving the frames, which can be decided by the individual subclasses.

frame_size

A tuple (width, height) in pixels of a movie frame.

grab_frame(**savefig_kwargs)

Grab the image information from the figure and save as a movie frame. All keyword arguments in savefig_kwargs are passed on to the savefig command that saves the figure.

classmethod isAvailable()

Check to see if a ImageMagickWriter is actually available.

Done by first checking the windows registry (if applicable) and then running the commandline tool.

output_args

saving(fig, outfile, dpi, *args, **kwargs)

Context manager to facilitate writing the movie file.

*args, **kw are any parameters that should be passed to setup.


```
setup(fig, outfile, dpi=None, frame_prefix='_tmp', clear_temp=True)
```

Perform setup for writing the movie file.

Parameters

`fig` [matplotlib.figure.Figure] The figure to grab the rendered frames from.

`outfile` [str] The filename of the resulting movie file.

`dpi` [number, optional] The dpi of the output file. This, with the figure size, controls the size in pixels of the resulting movie file. Default is `fig.dpi`.

`frame_prefix` [str, optional] The filename prefix to use for temporary files. Defaults to `'_tmp'`.

`clear_temp` [bool, optional] If the temporary files should be deleted after stitching the final result. Setting this to `False` can be useful for debugging. Defaults to `True`.

```
supported_formats = ['png', 'jpeg', 'ppm', 'tiff', 'sgi', 'bmp', 'pbm', 'raw', 'rgba']
```

33.2.7 matplotlib.animation.AVConvFileWriter

```
class matplotlib.animation.AVConvFileWriter(*args, **kwargs)
```

Bases: `matplotlib.animation.AVConvBase`, `matplotlib.animation.FFMpegFileWriter`

File-based avconv writer.

Frames are written to temporary files on disk and then stitched together at the end.

```
args_key = 'animation.avconv_args'
```

```
classmethod bin_path()
```

Returns the binary path to the commandline tool used by a specific subclass. This is a class method so that the tool can be looked for before making a particular `MovieWriter` subclass available.

```
cleanup()
```

Clean-up and collect the process used to write the movie file.

```
exec_key = 'animation.avconv_path'
```

```
finish()
```

Finish any processing for writing the movie.

```
frame_format
```

Format (png, jpeg, etc.) to use for saving the frames, which can be decided by the individual subclasses.

```
frame_size
```

A tuple (width, height) in pixels of a movie frame.

`grab_frame(**savefig_kwargs)`

Grab the image information from the figure and save as a movie frame. All keyword arguments in `savefig_kwargs` are passed on to the `savefig` command that saves the figure.

classmethod `isAvailable()`

Check to see if a `MovieWriter` subclass is actually available by running the commandline tool.

`output_args`

`saving(fig, outfile, dpi, *args, **kwargs)`

Context manager to facilitate writing the movie file.

`*args, **kw` are any parameters that should be passed to `setup`.

`setup(fig, outfile, dpi=None, frame_prefix='_tmp', clear_temp=True)`

Perform setup for writing the movie file.

Parameters

`fig` [`matplotlib.figure.Figure`] The figure to grab the rendered frames from.

`outfile` [`str`] The filename of the resulting movie file.

`dpi` [`number`, optional] The dpi of the output file. This, with the figure size, controls the size in pixels of the resulting movie file. Default is `fig.dpi`.

`frame_prefix` [`str`, optional] The filename prefix to use for temporary files. Defaults to `'_tmp'`.

`clear_temp` [`bool`, optional] If the temporary files should be deleted after stitching the final result. Setting this to `False` can be useful for debugging. Defaults to `True`.

`supported_formats` = [`'png'`, `'jpeg'`, `'ppm'`, `'tiff'`, `'sgi'`, `'bmp'`, `'pbm'`, `'raw'`, `'rgba'`]

Fundamentally, a `MovieWriter` provides a way to grab sequential frames from the same underlying `Figure` object. The base class `MovieWriter` implements 3 methods and a context manager. The only difference between the pipe-based and file-based writers is in the arguments to their respective `setup` methods.

The `setup()` method is used to prepare the writer (possibly opening a pipe), successive calls to `grab_frame()` capture a single frame at a time and `finish()` finalizes the movie and writes the output file to disk. For example

```
moviewriter = MovieWriter(...)
moviewriter.setup(fig=fig, 'my_movie.ext', dpi=100)
for j in range(n):
    update_figure(n)
    moviewriter.grab_frame()
moviewriter.finish()
```

If using the writer classes directly (not through `Animation.save`), it is strongly encouraged to use the saving context manager

```
with moviewriter.saving(fig, 'myfile.mp4', dpi=100):
    for j in range(n):
        update_figure(n)
        moviewriter.grab_frame()
```

to ensure that setup and cleanup are performed as necessary.

33.2.8 Examples

Note: [Click here to download the full example code](#)

Frame grabbing

Use a `MovieWriter` directly to grab individual frames and write them to a file. This avoids any event loop integration, and thus works even with the Agg backend. This is not recommended for use in an interactive setting.

```
import numpy as np
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from matplotlib.animation import FFMpegWriter

# Fixing random state for reproducibility
np.random.seed(19680801)

metadata = dict(title='Movie Test', artist='Matplotlib',
                comment='Movie support!')
writer = FFMpegWriter(fps=15, metadata=metadata)

fig = plt.figure()
l, = plt.plot([], [], 'k-o')

plt.xlim(-5, 5)
plt.ylim(-5, 5)

x0, y0 = 0, 0

with writer.saving(fig, "writer_test.mp4", 100):
    for i in range(100):
```

(continues on next page)

(continued from previous page)

```
x0 += 0.1 * np.random.randn()
y0 += 0.1 * np.random.randn()
l.set_data(x0, y0)
writer.grab_frame()
```

33.3 Helper Classes

33.3.1 Animation Base Classes

Animation	This class wraps the creation of an animation using matplotlib.
TimedAnimation	Animation subclass for time-based animation.

`matplotlib.animation.Animation`

`class matplotlib.animation.Animation(fig, event_source=None, blit=False)`

Bases: `object`

This class wraps the creation of an animation using matplotlib.

It is only a base class which should be subclassed to provide needed behavior.

This class is not typically used directly.

Parameters

`fig` [`matplotlib.figure.Figure`] The figure object that is used to get draw, resize, and any other needed events.

`event_source` [`object`, optional] A class that can run a callback when desired events are generated, as well as be stopped and started.

Examples include timers (see `TimedAnimation`) and file system notifications.

`blit` [`bool`, optional] controls whether blitting is used to optimize drawing. Defaults to `False`.

See also:

`FuncAnimation`, `ArtistAnimation`

`new_frame_seq()`

Creates a new sequence of frame information.

`new_saved_frame_seq()`

Creates a new sequence of saved/cached frame information.

`save(filename, writer=None, fps=None, dpi=None, codec=None, bitrate=None, extra_args=None, metadata=None, extra_anim=None, savefig_kwargs=None)`
Saves a movie file by drawing every frame.

Parameters

- `filename` [str] The output filename, e.g., `mymovie.mp4`.
- `writer` [MovieWriter or str, optional] A MovieWriter instance to use or a key that identifies a class to use, such as `'ffmpeg'`. If None, defaults to `rcParams["animation.writer"]`.
- `fps` [number, optional] Frames per second in the movie. Defaults to None, which will use the animation's specified interval to set the frames per second.
- `dpi` [number, optional] Controls the dots per inch for the movie frames. This combined with the figure's size in inches controls the size of the movie. If None, defaults to `rcParams["savefig.dpi"]`.
- `codec` [str, optional] The video codec to be used. Not all codecs are supported by a given MovieWriter. If None, default to `rcParams["animation.codec"]`.
- `bitrate` [number, optional] Specifies the number of bits used per second in the compressed movie, in kilobits per second. A higher number means a higher quality movie, but at the cost of increased file size. If None, defaults to `rcParams["animation.bitrate"]`.
- `extra_args` [list, optional] List of extra string arguments to be passed to the underlying movie utility. If None, defaults to `rcParams["animation.extra_args"]`.
- `metadata` [Dict[str, str], optional] Dictionary of keys and values for metadata to include in the output file. Some keys that may be of use include: `title`, `artist`, `genre`, `subject`, `copyright`, `srcform`, `comment`.
- `extra_anim` [list, optional] Additional Animation objects that should be included in the saved movie file. These need to be from the same `matplotlib.figure.Figure` instance. Also, animation frames will just be simply combined, so there should be a 1:1 correspondence between the frames from the different animations.
- `savefig_kwargs` [dict, optional] Is a dictionary containing keyword arguments to be passed on to the `savefig` command which is called repeatedly to save the individual frames.

Notes

`fps`, `codec`, `bitrate`, `extra_args`, `metadata` are used to construct a MovieWriter instance and can only be passed if `writer` is a string. If they are passed as non-None and `writer` is a MovieWriter, a `RuntimeError` will be raised.

`to_html5_video(embed_limit=None)`

Returns animation as an HTML5 video tag.

This saves the animation as an h264 video, encoded in base64 directly into the HTML5 video tag. This respects the rc parameters for the writer as well as the bitrate. This also makes use of the interval to control the speed, and uses the repeat parameter to decide whether to loop.

`to_jshtml(fps=None, embed_frames=True, default_mode=None)`

Generate HTML representation of the animation

`matplotlib.animation.TimedAnimation`

```
class matplotlib.animation.TimedAnimation(fig, interval=200, repeat_delay=None, repeat=True, event_source=None, *args, **kwargs)
```

Bases: `matplotlib.animation.Animation`

Animation subclass for time-based animation.

A new frame is drawn every interval milliseconds.

Parameters

`fig` [`matplotlib.figure.Figure`] The figure object that is used to get draw, resize, and any other needed events.

`interval` [number, optional] Delay between frames in milliseconds. Defaults to 200.

`repeat_delay` [number, optional] If the animation is repeated, adds a delay in milliseconds before repeating the animation. Defaults to None.

`repeat` [bool, optional] Controls whether the animation should repeat when the sequence of frames is completed. Defaults to True.

`blit` [bool, optional] Controls whether blitting is used to optimize drawing. Defaults to False.

`new_frame_seq()`

Creates a new sequence of frame information.

`new_saved_frame_seq()`

Creates a new sequence of saved/cached frame information.

`save(filename, writer=None, fps=None, dpi=None, codec=None, bitrate=None, extra_args=None, metadata=None, extra_anim=None, savefig_kwargs=None)`
Saves a movie file by drawing every frame.

Parameters

`filename` [str] The output filename, e.g., `mymovie.mp4`.

- `writer` [MovieWriter or str, optional] A MovieWriter instance to use or a key that identifies a class to use, such as 'ffmpeg'. If None, defaults to `rcParams["animation.writer"]`.
- `fps` [number, optional] Frames per second in the movie. Defaults to None, which will use the animation's specified interval to set the frames per second.
- `dpi` [number, optional] Controls the dots per inch for the movie frames. This combined with the figure's size in inches controls the size of the movie. If None, defaults to `rcParams["savefig.dpi"]`.
- `codec` [str, optional] The video codec to be used. Not all codecs are supported by a given MovieWriter. If None, default to `rcParams["animation.codec"]`.
- `bitrate` [number, optional] Specifies the number of bits used per second in the compressed movie, in kilobits per second. A higher number means a higher quality movie, but at the cost of increased file size. If None, defaults to `rcParams["animation.bitrate"]`.
- `extra_args` [list, optional] List of extra string arguments to be passed to the underlying movie utility. If None, defaults to `rcParams["animation.extra_args"]`.
- `metadata` [Dict[str, str], optional] Dictionary of keys and values for metadata to include in the output file. Some keys that may be of use include: title, artist, genre, subject, copyright, srcform, comment.
- `extra_anim` [list, optional] Additional Animation objects that should be included in the saved movie file. These need to be from the same matplotlib.figure.Figure instance. Also, animation frames will just be simply combined, so there should be a 1:1 correspondence between the frames from the different animations.
- `savefig_kwargs` [dict, optional] Is a dictionary containing keyword arguments to be passed on to the savefig command which is called repeatedly to save the individual frames.

Notes

`fps`, `codec`, `bitrate`, `extra_args`, `metadata` are used to construct a MovieWriter instance and can only be passed if `writer` is a string. If they are passed as non-None and `writer` is a MovieWriter, a `RuntimeError` will be raised.

`to_html5_video(embed_limit=None)`

Returns animation as an HTML5 video tag.

This saves the animation as an h264 video, encoded in base64 directly into the HTML5 video tag. This respects the rc parameters for the writer as well as the bitrate. This also makes use of the interval to control the speed, and uses the repeat parameter to decide whether to loop.

`to_jshtml(fps=None, embed_frames=True, default_mode=None)`
 Generate HTML representation of the animation

33.3.2 Writer Registry

A module-level registry is provided to map between the name of the writer and the class to allow a string to be passed to `Animation.save` instead of a writer instance.

<code>MovieWriterRegistry</code>	Registry of available writer classes by human readable name.
----------------------------------	--

`matplotlib.animation.MovieWriterRegistry`

class `matplotlib.animation.MovieWriterRegistry`

Bases: `object`

Registry of available writer classes by human readable name.

`ensure_not_dirty()`

If dirty, reasks the writers if they are available

`is_available(name)`

Check if given writer is available by name.

Parameters

name [str]

Returns

available [bool]

`list()`

Get a list of available `MovieWriters`.

`register(name)`

Decorator for registering a class under a name.

Example use:

```
@registry.register(name)
class Foo:
    pass
```

`reset_available_writers()`

Reset the available state of all registered writers

`set_dirty()`

Sets a flag to re-setup the writers.

33.3.3 Writer Base Classes

To reduce code duplication base classes

AbstractMovieWriter	Abstract base class for writing movies.
MovieWriter	Base class for writing movies.
FileMovieWriter	MovieWriter for writing to individual files and stitching at the end.

`matplotlib.animation.AbstractMovieWriter`

`class matplotlib.animation.AbstractMovieWriter`

Bases: `object`

Abstract base class for writing movies. Fundamentally, what a `MovieWriter` does is provide is a way to grab frames by calling `grab_frame()`.

`setup()` is called to start the process and `finish()` is called afterwards.

This class is set up to provide for writing movie frame data to a pipe. `saving()` is provided as a context manager to facilitate this process as:

```
with moviewriter.saving(fig, outfile='myfile.mp4', dpi=100):
    # Iterate over frames
    moviewriter.grab_frame(**savefig_kwargs)
```

The use of the context manager ensures that `setup()` and `finish()` are performed as necessary.

An instance of a concrete subclass of this class can be given as the `writer` argument of `Animation.save()`.

`finish()`

Finish any processing for writing the movie.

`grab_frame(**savefig_kwargs)`

Grab the image information from the figure and save as a movie frame.

All keyword arguments in `savefig_kwargs` are passed on to the `savefig` command that saves the figure.

`saving(fig, outfile, dpi, *args, **kwargs)`

Context manager to facilitate writing the movie file.

`*args, **kw` are any parameters that should be passed to `setup`.

`setup(fig, outfile, dpi=None)`

Perform setup for writing the movie file.

Parameters

`fig`: `'matplotlib.figure.Figure'` instance The figure object that contains the information for frames

outfile: string The filename of the resulting movie file

dpi: int, optional The DPI (or resolution) for the file. This controls the size in pixels of the resulting movie file. Default is fig.dpi.

matplotlib.animation.MovieWriter

```
class matplotlib.animation.MovieWriter(fps=5, codec=None, bitrate=None, extra_args=None, metadata=None)
```

Bases: matplotlib.animation.AbstractMovieWriter

Base class for writing movies.

This class is set up to provide for writing movie frame data to a pipe. See examples for how to use these classes.

Attributes

frame_format [str] The format used in writing frame data, defaults to 'rgba'

fig [Figure] The figure to capture data from. This must be provided by the sub-classes.

Parameters

fps: int Framerate for movie.

codec: string or None, optional The codec to use. If None (the default) the animation.codec rcParam is used.

bitrate: int or None, optional The bitrate for the saved movie file, which is one way to control the output file size and quality. The default value is None, which uses the animation.bitrate rcParam. A value of -1 implies that the bitrate should be determined automatically by the underlying utility.

extra_args: list of strings or None, optional A list of extra string arguments to be passed to the underlying movie utility. The default is None, which passes the additional arguments in the animation.extra_args rcParam.

metadata: Dict[str, str] or None A dictionary of keys and values for metadata to include in the output file. Some keys that may be of use include: title, artist, genre, subject, copyright, srcform, comment.

classmethod bin_path()

Returns the binary path to the commandline tool used by a specific subclass. This is a class method so that the tool can be looked for before making a particular MovieWriter subclass available.

cleanup()

Clean-up and collect the process used to write the movie file.

finish()

Finish any processing for writing the movie.

`frame_size`

A tuple (width, height) in pixels of a movie frame.

`grab_frame(**savefig_kwargs)`

Grab the image information from the figure and save as a movie frame.

All keyword arguments in `savefig_kwargs` are passed on to the `savefig` command that saves the figure.

`classmethod isAvailable()`

Check to see if a `MovieWriter` subclass is actually available by running the commandline tool.

`saving(fig, outfile, dpi, *args, **kwargs)`

Context manager to facilitate writing the movie file.

`*args, **kw` are any parameters that should be passed to `setup`.

`setup(fig, outfile, dpi=None)`

Perform setup for writing the movie file.

Parameters

`fig` [`matplotlib.figure.Figure`] The figure object that contains the information for frames

`outfile` [`string`] The filename of the resulting movie file

`dpi` [`int`, optional] The DPI (or resolution) for the file. This controls the size in pixels of the resulting movie file. Default is `fig.dpi`.

`matplotlib.animation.FileMovieWriter`

`class matplotlib.animation.FileMovieWriter(*args, **kwargs)`

Bases: `matplotlib.animation.MovieWriter`

`MovieWriter` for writing to individual files and stitching at the end.

This must be sub-classed to be useful.

`classmethod bin_path()`

Returns the binary path to the commandline tool used by a specific subclass. This is a class method so that the tool can be looked for before making a particular `MovieWriter` subclass available.

`cleanup()`

Clean-up and collect the process used to write the movie file.

`finish()`

Finish any processing for writing the movie.

`frame_format`

Format (png, jpeg, etc.) to use for saving the frames, which can be decided by the individual subclasses.

`frame_size`

A tuple (width, height) in pixels of a movie frame.

`grab_frame(**savefig_kwargs)`

Grab the image information from the figure and save as a movie frame. All keyword arguments in `savefig_kwargs` are passed on to the `savefig` command that saves the figure.

`classmethod isAvailable()`

Check to see if a `MovieWriter` subclass is actually available by running the commandline tool.

`saving(fig, outfile, dpi, *args, **kwargs)`

Context manager to facilitate writing the movie file.

`*args, **kw` are any parameters that should be passed to setup.

`setup(fig, outfile, dpi=None, frame_prefix='_tmp', clear_temp=True)`

Perform setup for writing the movie file.

Parameters

`fig` [`matplotlib.figure.Figure`] The figure to grab the rendered frames from.

`outfile` [`str`] The filename of the resulting movie file.

`dpi` [`number`, optional] The dpi of the output file. This, with the figure size, controls the size in pixels of the resulting movie file. Default is `fig.dpi`.

`frame_prefix` [`str`, optional] The filename prefix to use for temporary files. Defaults to `'_tmp'`.

`clear_temp` [`bool`, optional] If the temporary files should be deleted after stitching the final result. Setting this to `False` can be useful for debugging. Defaults to `True`.

and mixins

<code>AVConvBase</code>	Mixin class for <code>avconv</code> output.
<code>FFMpegBase</code>	Mixin class for <code>FFMpeg</code> output.
<code>ImageMagickBase</code>	Mixin class for <code>ImageMagick</code> output.

`matplotlib.animation.AVConvBase`

`class matplotlib.animation.AVConvBase`

Bases: `matplotlib.animation.FFMpegBase`

Mixin class for `avconv` output.

To be useful this must be multiply-inherited from with a `MovieWriterBase` sub-class.

`args_key = 'animation.avconv_args'`

```
exec_key = 'animation.avconv_path'
```

```
output_args
```

`matplotlib.animation.FFMpegBase`

```
class matplotlib.animation.FFMpegBase
```

```
    Bases: object
```

```
    Mixin class for FFMpeg output.
```

```
    To be useful this must be multiply-inherited from with a MovieWriterBase sub-class.
```

```
    args_key = 'animation.ffmpeg_args'
```

```
    exec_key = 'animation.ffmpeg_path'
```

```
    output_args
```

`matplotlib.animation.ImageMagickBase`

```
class matplotlib.animation.ImageMagickBase
```

```
    Bases: object
```

```
    Mixin class for ImageMagick output.
```

```
    To be useful this must be multiply-inherited from with a MovieWriterBase sub-class.
```

```
    args_key = 'animation.convert_args'
```

```
    delay
```

```
    exec_key = 'animation.convert_path'
```

```
    classmethod isAvailable()
```

```
        Check to see if a ImageMagickWriter is actually available.
```

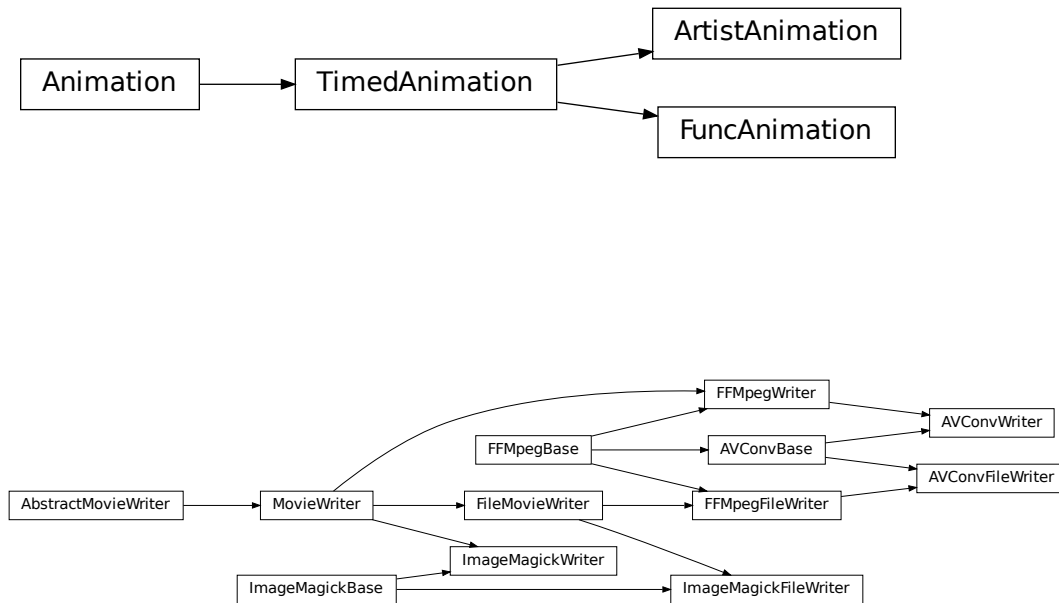
```
        Done by first checking the windows registry (if applicable) and then running the com-  
        mandline tool.
```

```
    output_args
```

are provided.

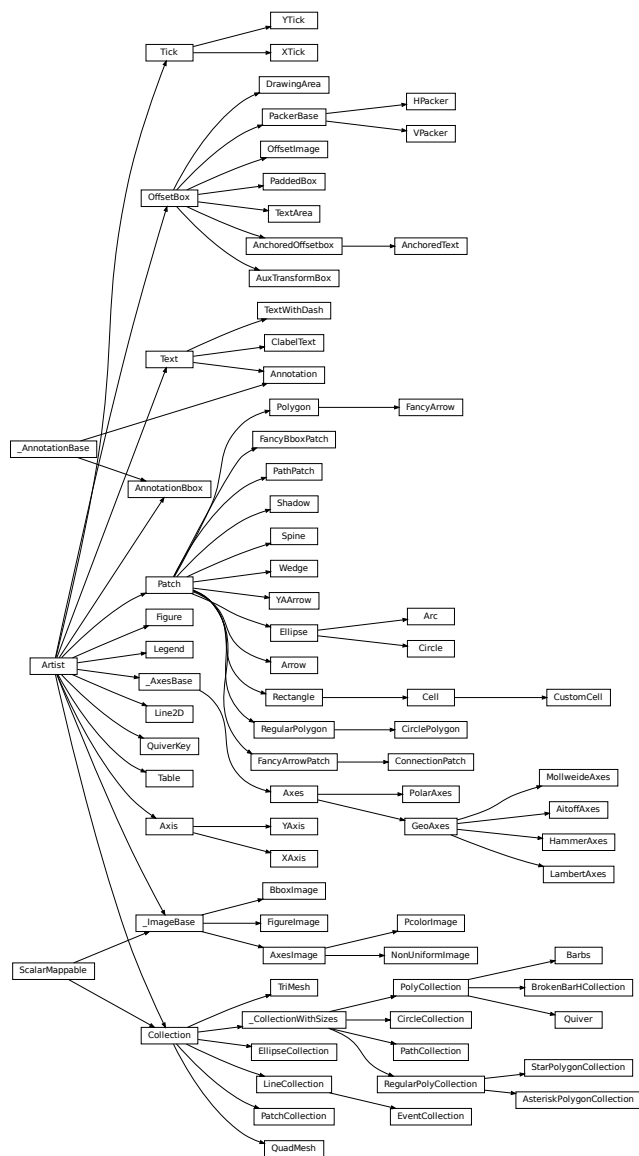
See the source code for how to easily implement new MovieWriter classes.

33.4 Inheritance Diagrams



CHAPTER
THIRTYFOUR

ARTIST MODULE



34.1 Artist class

`class matplotlib.artist.Artist`

Abstract base class for someone who renders into a `FigureCanvas`.

34.1.1 Interactive

<code>Artist.add_callback</code>	Adds a callback function that will be called whenever one of the Artist's properties changes.
<code>Artist.format_cursor_data</code>	Return cursor data string formatted.
<code>Artist.get_contains</code>	Return the <code>_contains</code> test used by the artist, or <code>None</code> for default.
<code>Artist.get_cursor_data</code>	Get the cursor data for a given event.
<code>Artist.get_picker</code>	Return the picker object used by this artist.
<code>Artist.mouseover</code>	
<code>Artist.pchanged</code>	Fire an event when property changed, calling all of the registered callbacks.
<code>Artist.pick</code>	Process pick event
<code>Artist.pickable</code>	Return <code>True</code> if Artist is pickable.
<code>Artist.remove_callback</code>	Remove a callback based on its id.
<code>Artist.set_contains</code>	Replace the contains test used by this artist.
<code>Artist.set_picker</code>	Set the epsilon for picking used by this artist
<code>Artist.contains</code>	Test whether the artist contains the mouse event.

`matplotlib.artist.Artist.add_callback`

`Artist.add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`matplotlib.artist.Artist.format_cursor_data`

`Artist.format_cursor_data(data)`

Return cursor data string formatted.

`matplotlib.artist.Artist.get_contains`

`Artist.get_contains()`

Return the `_contains` test used by the artist, or `None` for default.

`matplotlib.artist.Artist.get_cursor_data`

`Artist.get_cursor_data(event)`

Get the cursor data for a given event.

`matplotlib.artist.Artist.get_picker`

`Artist.get_picker()`

Return the picker object used by this artist.

`matplotlib.artist.Artist.mouseover`

`Artist.mouseover`

`matplotlib.artist.Artist.pchanged`

`Artist.pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`matplotlib.artist.Artist.pick`

`Artist.pick(mouseevent)`

Process pick event

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

`matplotlib.artist.Artist.pickable`

`Artist.pickable()`

Return True if Artist is pickable.

`matplotlib.artist.Artist.remove_callback`

`Artist.remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`matplotlib.artist.Artist.set__contains``Artist.set__contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

`picker` [callable]

`matplotlib.artist.Artist.set__picker``Artist.set__picker(picker)`

Set the epsilon for picking used by this artist

`picker` can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if `picker` is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if `picker` is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

to determine the hit test. if the mouse event is over the artist, return `hit=True` and `props` is a dictionary of properties you want added to the `PickEvent` attributes.

Parameters

`picker` [None or bool or float or callable]

`matplotlib.artist.Artist.contains``Artist.contains(mouseevent)`

Test whether the artist contains the mouse event.

Returns the truth value and a dictionary of artist specific details of selection, such as which points are contained in the pick radius. See individual artists for details.

34.1.2 Margins and Autoscaling

<code>Artist.sticky_edges</code>	x and y sticky edge lists.
----------------------------------	----------------------------

`matplotlib.artist.Artist.sticky_edges`

`Artist.sticky_edges`
x and y sticky edge lists.

When performing autoscaling, if a data limit coincides with a value in the corresponding `sticky_edges` list, then no margin will be added—the view limit “sticks” to the edge. A typical usecase is histograms, where one usually expects no margin on the bottom edge (0) of the histogram.

This attribute cannot be assigned to; however, the x and y lists can be modified in place as needed.

Examples

```
>>> artist.sticky_edges.x[:] = (xmin, xmax)
>>> artist.sticky_edges.y[:] = (ymin, ymax)
```

34.1.3 Clipping

<code>Artist.get_clip_box</code>	Return artist clipbox
<code>Artist.get_clip_on</code>	Return whether artist uses clipping
<code>Artist.get_clip_path</code>	Return artist clip path
<code>Artist.set_clip_box</code>	Set the artist’s clip Bbox.
<code>Artist.set_clip_on</code>	Set whether artist uses clipping.
<code>Artist.set_clip_path</code>	Set the artist’s clip path, which may be:

`matplotlib.artist.Artist.get_clip_box`

`Artist.get_clip_box()`
Return artist clipbox

`matplotlib.artist.Artist.get_clip_on`

`Artist.get_clip_on()`
Return whether artist uses clipping

`matplotlib.artist.Artist.get_clip_path`

`Artist.get_clip_path()`
Return artist clip path

`matplotlib.artist.Artist.set_clip_box`

`Artist.set_clip_box(clipbox)`
Set the artist's clip Bbox.

Parameters
clipbox [Bbox]

`matplotlib.artist.Artist.set_clip_on`

`Artist.set_clip_on(b)`
Set whether artist uses clipping.
When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters
b [bool]

`matplotlib.artist.Artist.set_clip_path`

`Artist.set_clip_path(path, transform=None)`
Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

34.1.4 Bulk Properties

<code>Artist.update</code>	Update this artist's properties from the dictionary prop.
----------------------------	---

<code>Artist.update_from</code>	Copy properties from other to self.
---------------------------------	-------------------------------------

Continued on next page

Table 4 – continued from previous page

<code>Artist.properties</code>	return a dictionary mapping property name -> value for all Artist props
<code>Artist.set</code>	A property batch setter.

`matplotlib.artist.Artist.update`

`Artist.update(props)`

Update this artist's properties from the dictionary prop.

`matplotlib.artist.Artist.update_from`

`Artist.update_from(other)`

Copy properties from other to self.

`matplotlib.artist.Artist.properties`

`Artist.properties()`

return a dictionary mapping property name -> value for all Artist props

`matplotlib.artist.Artist.set`

`Artist.set(**kwargs)`

A property batch setter. Pass kwargs to set properties.

34.1.5 Drawing

<code>Artist.draw</code>	Derived classes drawing method
<code>Artist.get_animated</code>	Return the artist's animated state
<code>Artist.set_animated</code>	Set the artist's animation state.
<code>Artist.get_agg_filter</code>	Return filter function to be used for agg filter.
<code>Artist.get_alpha</code>	Return the alpha value used for blending - not supported on all backends
<code>Artist.get_snap</code>	Returns the snap setting which may be:
<code>Artist.get_visible</code>	Return the artist's visibility
<code>Artist.get_zorder</code>	Return the artist's zorder.
<code>Artist.set_agg_filter</code>	Set the agg filter.
<code>Artist.set_alpha</code>	Set the alpha value used for blending - not supported on all backends.
<code>Artist.set_sketch_params</code>	Sets the sketch parameters.
<code>Artist.set_snap</code>	Sets the snap setting which may be:
<code>Artist.get_rasterized</code>	Return whether the artist is to be rasterized.
<code>Artist.get_sketch_params</code>	Returns the sketch parameters for the artist.

Continued on next page

Table 5 – continued from previous page

<code>Artist.set_path_effects</code>	Set the path effects.
<code>Artist.set_rasterized</code>	Force rasterized (bitmap) drawing in vector backend output.
<code>Artist.zorder</code>	
<code>Artist.set_visible</code>	Set the artist's visibility.
<code>Artist.set_zorder</code>	Set the zorder for the artist.
<code>Artist.get_window_extent</code>	Get the axes bounding box in display space.
<code>Artist.get_path_effects</code>	
<code>Artist.get_transformed_clip_path_and_affine</code>	Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`matplotlib.artist.Artist.draw`

`Artist.draw(renderer, *args, **kwargs)`
Derived classes drawing method

`matplotlib.artist.Artist.get_animated`

`Artist.get_animated()`
Return the artist's animated state

`matplotlib.artist.Artist.set_animated`

`Artist.set_animated(b)`
Set the artist's animation state.

Parameters

`b` [bool]

`matplotlib.artist.Artist.get_agg_filter`

`Artist.get_agg_filter()`
Return filter function to be used for agg filter.

`matplotlib.artist.Artist.get_alpha`

`Artist.get_alpha()`
Return the alpha value used for blending - not supported on all backends

`matplotlib.artist.Artist.get_snap`

`Artist.get_snap()`

Returns the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`matplotlib.artist.Artist.get_visible`

`Artist.get_visible()`

Return the artist's visibility

`matplotlib.artist.Artist.get_zorder`

`Artist.get_zorder()`

Return the artist's zorder.

`matplotlib.artist.Artist.set_agg_filter`

`Artist.set_agg_filter(filter_func)`

Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`matplotlib.artist.Artist.set_alpha`

`Artist.set_alpha(alpha)`

Set the alpha value used for blending - not supported on all backends.

Parameters

`alpha` [float]

`matplotlib.artist.Artist.set_sketch_params`

`Artist.set_sketch_params(scale=None, length=None, randomness=None)`

Sets the sketch parameters.

Parameters

scale [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If scale is `None`, or not provided, no sketch filter will be provided.

length [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

randomness [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

`matplotlib.artist.Artist.set_snap`

`Artist.set_snap(snap)`

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

snap [bool or None]

`matplotlib.artist.Artist.get_rasterized`

`Artist.get_rasterized()`

Return whether the artist is to be rasterized.

`matplotlib.artist.Artist.get_sketch_params`

`Artist.get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

sketch_params [tuple or `None`]

A 3-tuple with the following elements:

- scale: The amplitude of the wiggle perpendicular to the source line.
- length: The length of the wiggle along the line.
- randomness: The scale factor by which the length is shrunk or expanded.

May return 'None' if no sketch parameters were set.

`matplotlib.artist.Artist.set_path_effects`

`Artist.set_path_effects(path_effects)`

Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`matplotlib.artist.Artist.set_rasterized`

`Artist.set_rasterized(rasterized)`

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

`rasterized` [bool or None]

`matplotlib.artist.Artist.zorder`

`Artist.zorder = 0`

`matplotlib.artist.Artist.set_visible`

`Artist.set_visible(b)`

Set the artist's visibility.

Parameters

`b` [bool]

`matplotlib.artist.Artist.set_zorder`

`Artist.set_zorder(level)`

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

`level` [float]

`matplotlib.artist.Artist.get_window_extent`

`Artist.get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`matplotlib.artist.Artist.get_path_effects`

`Artist.get_path_effects()`

`matplotlib.artist.Artist.get_transformed_clip_path_and_affine`

`Artist.get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

34.1.6 Figure and Axes

<code>Artist.remove</code>	Remove the artist from the figure if possible.
<code>Artist.axes</code>	The Axes instance the artist resides in, or None.
<code>Artist.set_figure</code>	Set the Figure instance the artist belongs to.
<code>Artist.get_figure</code>	Return the Figure instance the artist belongs to.

`matplotlib.artist.Artist.remove`

`Artist.remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`matplotlib.artist.Artist.axes`

`Artist.axes`

The Axes instance the artist resides in, or None.

`matplotlib.artist.Artist.set_figure`

`Artist.set_figure(fig)`

Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`matplotlib.artist.Artist.get_figure`

`Artist.get_figure()`

Return the Figure instance the artist belongs to.

34.1.7 Children

`Artist.get_children`

Return a list of the child Artist`s this :class:`Artist` contains.

`Artist.findobj`

Find artist objects.

`matplotlib.artist.Artist.get_children`

`Artist.get_children()`

Return a list of the child Artist`s this :class:`Artist` contains.

`matplotlib.artist.Artist.findobj`

`Artist.findobj(match=None, include_self=True)`

Find artist objects.

Recursively find all Artist instances contained in self.

`match` can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is True (default), include self in the list to be checked for a match.

34.1.8 Transform

`Artist.set_transform`

Set the artist transform.

Continued on next page

Table 8 – continued from previous page

<code>Artist.get_transform</code>	Return the Transform instance used by this artist.
<code>Artist.is_transform_set</code>	Returns True if Artist has a transform explicitly set.

`matplotlib.artist.Artist.set_transform`

`Artist.set_transform(t)`
Set the artist transform.

Parameters

t [Transform]

`matplotlib.artist.Artist.get_transform`

`Artist.get_transform()`
Return the Transform instance used by this artist.

`matplotlib.artist.Artist.is_transform_set`

`Artist.is_transform_set()`
Returns True if Artist has a transform explicitly set.

34.1.9 Units

<code>Artist.convert_xunits</code>	For artists in an axes, if the xaxis has units support, convert x using xaxis unit type
<code>Artist.convert_yunits</code>	For artists in an axes, if the yaxis has units support, convert y using yaxis unit type
<code>Artist.have_units</code>	Return True if units are set on the x or y axes

`matplotlib.artist.Artist.convert_xunits`

`Artist.convert_xunits(x)`
For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`matplotlib.artist.Artist.convert_yunits`

`Artist.convert_yunits(y)`
For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`matplotlib.artist.Artist.have_units`

`Artist.have_units()`

Return True if units are set on the x or y axes

34.1.10 Metadata

<code>Artist.get_gid</code>	Returns the group id.
<code>Artist.get_label</code>	Get the label used for this artist in the legend.
<code>Artist.set_gid</code>	Sets the (group) id for the artist.
<code>Artist.set_label</code>	Set the label to s for auto legend.
<code>Artist.get_url</code>	Returns the url.
<code>Artist.set_url</code>	Sets the url for the artist.
<code>Artist.aname</code>	

`matplotlib.artist.Artist.get_gid`

`Artist.get_gid()`

Returns the group id.

`matplotlib.artist.Artist.get_label`

`Artist.get_label()`

Get the label used for this artist in the legend.

`matplotlib.artist.Artist.set_gid`

`Artist.set_gid(gid)`

Sets the (group) id for the artist.

Parameters

gid [str]

`matplotlib.artist.Artist.set_label`

`Artist.set_label(s)`

Set the label to s for auto legend.

Parameters

s [object] s will be converted to a string by calling `str` (unicode on Py2).

matplotlib.artist.Artist.get_url

Artist.get_url()
Returns the url.

matplotlib.artist.Artist.set_url

Artist.set_url(url)
Sets the url for the artist.

Parameters
url [str]

matplotlib.artist.Artist.aname

Artist.aname = 'Artist'

34.1.11 Stale

Artist.stale	If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.
--------------	---

matplotlib.artist.Artist.stale

Artist.stale
If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

34.2 Functions

allow_rasterization	Decorator for Artist.draw method.
get	Return the value of object's property.
getp	Return the value of object's property.
setp	Set a property on an artist object.
kwdoc	
ArtistInspector	A helper class to inspect an Artist and return information about it's settable properties and their current values.

34.2.1 matplotlib.artist.allow_rasterization

matplotlib.artist.allow_rasterization(draw)

Decorator for Artist.draw method. Provides routines that run before and after the draw call. The before and after functions are useful for changing artist-dependent renderer attributes or making other setup function calls, such as starting and flushing a mixed-mode renderer.

34.2.2 matplotlib.artist.get

matplotlib.artist.get(obj, property=None)

Return the value of object's property. property is an optional string for the property you want to return

Example usage:

```
getp(obj) # get all the object properties
getp(obj, 'linestyle') # get the linestyle property
```

obj is a Artist instance, e.g., Line2D or an instance of a Axes or matplotlib.text.Text. If the property is 'somename', this function returns

```
obj.get_somename()
```

getp() can be used to query all the gettable properties with getp(obj). Many properties have aliases for shorter typing, e.g. 'lw' is an alias for 'linewidth'. In the output, aliases and full property names will be listed as:

```
property or alias = value
```

e.g.:

```
linewidth or lw = 2
```

34.2.3 matplotlib.artist.getp

matplotlib.artist.getp(obj, property=None)

Return the value of object's property. property is an optional string for the property you want to return

Example usage:

```
getp(obj) # get all the object properties
getp(obj, 'linestyle') # get the linestyle property
```

obj is a Artist instance, e.g., Line2D or an instance of a Axes or matplotlib.text.Text. If the property is 'somename', this function returns

```
obj.get_somename()
```

`getp()` can be used to query all the gettable properties with `getp(obj)`. Many properties have aliases for shorter typing, e.g. `'lw'` is an alias for `'linewidth'`. In the output, aliases and full property names will be listed as:

property or alias = value

e.g.:

linewidth or lw = 2

34.2.4 matplotlib.artist.setp

`matplotlib.artist.setp(obj, *args, **kwargs)`

Set a property on an artist object.

matplotlib supports the use of `setp()` ("set property") and `getp()` to set and get object properties, as well as to do introspection on the object. For example, to set the linestyle of a line to be dashed, you can do:

```
>>> line, = plot([1,2,3])
>>> setp(line, linestyle='--')
```

If you want to know the valid types of arguments, you can provide the name of the property you want to set without a value:

```
>>> setp(line, 'linestyle')
linestyle: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' ]
```

If you want to see all the properties that can be set, and their possible values, you can do:

```
>>> setp(line)
... long output listing omitted
```

You may specify another output file to `setp` if `sys.stdout` is not acceptable for some reason using the file keyword-only argument:

```
>>> with fopen('output.log') as f:
>>>     setp(line, file=f)
```

`setp()` operates on a single instance or a iterable of instances. If you are in query mode introspecting the possible values, only the first instance in the sequence is used. When actually setting values, all the instances will be set. e.g., suppose you have a list of two lines, the following will make both lines thicker and red:

```
>>> x = arange(0,1.0,0.01)
>>> y1 = sin(2*pi*x)
>>> y2 = sin(4*pi*x)
>>> lines = plot(x, y1, x, y2)
>>> setp(lines, linewidth=2, color='r')
```

setp() works with the MATLAB style string/value pairs or with python kwargs. For example, the following are equivalent:

```
>>> setp(lines, 'linewidth', 2, 'color', 'r') # MATLAB style
>>> setp(lines, linewidth=2, color='r')      # python style
```

34.2.5 matplotlib.artist.kwdoc

matplotlib.artist.kwdoc(a)

34.2.6 matplotlib.artist.ArtistInspector

class matplotlib.artist.ArtistInspector(o)

A helper class to inspect an Artist and return information about it's settable properties and their current values.

Initialize the artist inspector with an Artist or an iterable of Artists. If an iterable is used, we assume it is a homogeneous sequence (all Artists are of the same type) and it is your responsibility to make sure this is so.

__init__(o)

Initialize the artist inspector with an Artist or an iterable of Artists. If an iterable is used, we assume it is a homogeneous sequence (all Artists are of the same type) and it is your responsibility to make sure this is so.

Methods

__init__(o)	Initialize the artist inspector with an Artist or an iterable of Artists.
aliased_name(s)	return 'PROPNAME or alias' if s has an alias, else return PROPNAME.
aliased_name_rest(s, target)	return 'PROPNAME or alias' if s has an alias, else return PROPNAME formatted for ReST
get_aliases()	Get a dict mapping fullname -> alias for each alias in the ArtistInspector.
get_setters()	Get the attribute strings with setters for object.
get_valid_values(attr)	Get the legal arguments for the setter associated with attr.
is_alias(o)	Return True if method object o is an alias for another function.

Continued on next page

Table 13 – continued from previous page

<code>pprint_getters()</code>	Return the getters and actual values as list of strings.
<code>pprint_setters([prop, leadingspace])</code>	If prop is None, return a list of strings of all settable properties and their valid values.
<code>pprint_setters_rest([prop, leadingspace])</code>	If prop is None, return a list of strings of all settable properties and their valid values.
<code>properties()</code>	return a dictionary mapping property name -> value

`aliased_name(s)`

return 'PROPNAME or alias' if s has an alias, else return PROPNAME.

e.g., for the line `markerfacecolor` property, which has an alias, return 'markerfacecolor or mfc' and for the transform property, which does not, return 'transform'

`aliased_name_rest(s, target)`

return 'PROPNAME or alias' if s has an alias, else return PROPNAME formatted for ReST

e.g., for the line `markerfacecolor` property, which has an alias, return 'markerfacecolor or mfc' and for the transform property, which does not, return 'transform'

`get_aliases()`

Get a dict mapping fullname -> alias for each alias in the ArtistInspector.

e.g., for lines:

```
{'markerfacecolor': 'mfc',
 'linewidth'      : 'lw',
 }
```

`get_setters()`

Get the attribute strings with setters for object. e.g., for a line, return ['markerfacecolor', 'linewidth',].

`get_valid_values(attr)`

Get the legal arguments for the setter associated with attr.

This is done by querying the docstring of the function `set_attr` for a line that begins with "ACCEPTS" or ".. ACCEPTS":

e.g., for a line `linestyle`, return "['-' | '--' | '-.' | ':' | 'steps' | 'None']"

`is_alias(o)`

Return True if method object o is an alias for another function.

`pprint_getters()`

Return the getters and actual values as list of strings.

`pprint_setters(prop=None, leadingspace=2)`

If prop is None, return a list of strings of all settable properties and their valid values.

If prop is not None, it is a valid property name and that property will be returned as a string of property : valid values.

`pprint__setters__rest(prop=None, leadingspace=4)`

If prop is None, return a list of strings of all settable properties and their valid values.
Format the output for ReST

If prop is not None, it is a valid property name and that property will be returned as a string of property : valid values.

`properties()`

return a dictionary mapping property name -> value

AXES CLASS

```
class matplotlib.axes.Axes(fig, rect, facecolor=None, frameon=True, sharex=None,  
                           sharey=None, label="", xscale=None, yscale=None, **kwargs)
```

The Axes contains most of the figure elements: Axis, Tick, Line2D, Text, Polygon, etc., and sets the coordinate system.

The Axes instance supports callbacks through a `callbacks` attribute which is a `CallbackRegistry` instance. The events you can connect to are `'xlim_changed'` and `'ylim_changed'` and the callback will be called with `func(ax)` where `ax` is the Axes instance.

Build an Axes instance in Figure `fig` with `rect=[left, bottom, width, height]` in Figure coordinates

Optional keyword arguments:

Keyword	Description
adjustable	['box' 'datalim']
alpha	float: the alpha transparency (can be None)
anchor	['C', 'SW', 'S', 'SE', 'E', 'NE', 'N', 'NW', 'W']
aspect	['auto' 'equal' aspect_ratio]
autoscale_on	bool; whether to autoscale the viewlim
axisbelow	[bool 'line'] draw the grids and ticks below or above most other artists, or below lines but above patches
cursor_props	a (float, color) tuple
figure	a Figure instance
frame_on	bool; whether to draw the axes frame
label	the axes label
navigate	bool
navigate_mode	['PAN' 'ZOOM' None] the navigation toolbar button status
position	[left, bottom, width, height] in class:Figure coords
sharex	an class:Axes instance to share the x-axis with
sharey	an class:Axes instance to share the y-axis with
title	the title string
visible	bool, whether the axes is visible
xlabel	the xlabel
xlim	(xmin, xmax) view limits
xscale	['linear' 'log' 'logit' 'symlog']
xticklabels	sequence of strings
xticks	sequence of floats
ylabel	the ylabel strings
ylim	(ymin, ymax) view limits
yscale	['linear' 'log' 'logit' 'symlog']
yticklabels	sequence of strings
yticks	sequence of floats

Table of Contents

- Plotting
 - Basic
 - Spans

- Spectral
 - Statistics
 - Binned
 - Contours
 - Array
 - Unstructured Triangles
 - Text and Annotations
 - Fields
- Clearing
- Appearance
- Property cycle
- Axis / limits
 - Axis Limits and direction
 - Axis Labels, title, and legend
 - Axis scales
 - Autoscaling and margins
 - Aspect ratio
 - Ticks and tick labels
- Units
- Adding Artists
- Twinning
- Axes Position
- Async/Event based
- Interactive
- Children
- Drawing
- Bulk property manipulation
- General Artist Properties
- Artist Methods
- Projection
- Other

- Inheritance

35.1 Plotting

35.1.1 Basic

<code>Axes.plot</code>	Plot y versus x as lines and/or markers.
<code>Axes.errorbar</code>	Plot y versus x as lines and/or markers with attached errorbars.
<code>Axes.scatter</code>	A scatter plot of y vs x with varying marker size and/or color.
<code>Axes.plot_date</code>	Plot data that contains dates.
<code>Axes.step</code>	Make a step plot.
<code>Axes.loglog</code>	Make a plot with log scaling on both the x and y axis.
<code>Axes.semilogx</code>	Make a plot with log scaling on the x axis.
<code>Axes.semilogy</code>	Make a plot with log scaling on the y axis.
<code>Axes.fill_between</code>	Fill the area between two horizontal curves.
<code>Axes.fill_betweenx</code>	Fill the area between two vertical curves.
<code>Axes.bar</code>	Make a bar plot.
<code>Axes.barh</code>	Make a horizontal bar plot.
<code>Axes.stem</code>	Create a stem plot.
<code>Axes.eventplot</code>	Plot identical parallel lines at the given positions.
<code>Axes.pie</code>	Plot a pie chart.
<code>Axes.stackplot</code>	Draws a stacked area plot.
<code>Axes.broken_barh</code>	Plot a horizontal sequence of rectangles.
<code>Axes.vlines</code>	Plot vertical lines.
<code>Axes.hlines</code>	Plot horizontal lines at each y from xmin to xmax.
<code>Axes.fill</code>	Plot filled polygons.

matplotlib.axes.Axes.plot

`Axes.plot(*args, data=None, **kwargs)`
 Plot y versus x as lines and/or markers.

Call signatures:

```
plot([x], y, [fmt], data=None, **kwargs)
plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

The coordinates of the points or line nodes are given by x, y.

The optional parameter `fmt` is a convenient way for defining basic formatting like color, marker

and linestyle. It's a shortcut string notation described in the Notes section below.

```
>>> plot(x, y)      # plot x and y using default line style and color
>>> plot(x, y, 'bo') # plot x and y using blue circle markers
>>> plot(y)         # plot y using x as index array 0..N-1
>>> plot(y, 'r+')    # ditto, but with red plusses
```

You can use Line2D properties as keyword arguments for more control on the appearance. Line properties and fmt can be mixed. The following two calls yield identical results:

```
>>> plot(x, y, 'go--', linewidth=2, markersize=12)
>>> plot(x, y, color='green', marker='o', linestyle='dashed',
        linewidth=2, markersize=12)
```

When conflicting with fmt, keyword arguments take precedence.

Plotting labelled data

There's a convenient way for plotting objects with labelled data (i.e. data that can be accessed by index obj['y']). Instead of giving the data in x and y, you can provide the object in the data parameter and just give the labels for x and y:

```
>>> plot('xlabel', 'ylabel', data=obj)
```

All indexable objects are supported. This could e.g. be a `dict`, a `pandas.DataFrame` or a structured numpy array.

Plotting multiple sets of data

There are various ways to plot multiple sets of data.

- The most straight forward way is just to call plot multiple times. Example:

```
>>> plot(x1, y1, 'bo')
>>> plot(x2, y2, 'go')
```

- Alternatively, if your data is already a 2d array, you can pass it directly to x, y. A separate data set will be drawn for every column.

Example: an array a where the first column represents the x values and the other columns are the y columns:

```
>>> plot(a[0], a[1:])
```

- The third way is to specify multiple sets of [x], y, [fmt] groups:

```
>>> plot(x1, y1, 'g^', x2, y2, 'g-')
```

In this case, any additional keyword argument applies to all datasets. Also this syntax cannot be combined with the data parameter.

By default, each line is assigned a different style specified by a 'style cycle'. The `fmt` and line property parameters are only necessary if you want explicit deviations from these defaults. Alternatively, you can also change the style cycle using the `'axes.prop_cycle'` rcParam.

Parameters

`x, y` [array-like or scalar] The horizontal / vertical coordinates of the data points. `x` values are optional. If not given, they default to `[0, ..., N-1]`.

Commonly, these parameters are arrays of length `N`. However, scalars are supported as well (equivalent to an array with constant value).

The parameters can also be 2-dimensional. Then, the columns represent separate data sets.

`fmt` [str, optional] A format string, e.g. `'ro'` for red circles. See the Notes section for a full description of the format strings.

Format strings are just an abbreviation for quickly setting basic line properties. All of these and more can also be controlled by keyword arguments.

`data` [indexable object, optional] An object with labelled data. If given, provide the label names to plot in `x` and `y`.

Note: Technically there's a slight ambiguity in calls where the second label is a valid `fmt`. `plot('n', 'o', data=obj)` could be `plt(x, y)` or `plt(y, fmt)`. In such cases, the former interpretation is chosen, but a warning is issued. You may suppress the warning by adding an empty format string `plot('n', 'o', '', data=obj)`.

Returns

`lines` A list of `Line2D` objects representing the plotted data.

Other Parameters

`scalex, scaley` [bool, optional, default: `True`] These parameters determined if the view limits are adapted to the data limits. The values are passed on to `autoscale_view`.

`**kwargs` [`Line2D` properties, optional] `kwargs` are used to specify properties like a line label (for auto legends), `linewidth`, `antialiased`, `marker face color`. Example:

```
>>> plot([1,2,3], [1,2,3], 'go-', label='line 1', linewidth=2)
>>> plot([1,2,3], [1,4,9], 'rs', label='line 2')
```

If you make multiple lines with one plot command, the `kwargs` apply to all those lines.

Here is a list of available `Line2D` properties:

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) float (0.0 transparent through 1.0 opaque)
alpha		float (0.0 transparent through 1.0 opaque)
animated		bool
antialiased or aa		bool
clip_box		a Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
color or c		any matplotlib color
contains		a callable function
dash_capstyle		['butt' 'round' 'projecting']
dash_joinstyle		['miter' 'round' 'bevel']
dashes		sequence of on/off ink in points
drawstyle		['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']
figure		a Figure instance
fillstyle		['full' 'left' 'right' 'bottom' 'top' 'none']
gid		an id string
label		object
linestyle or ls		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']
linewidth or lw		float value in points
marker		A valid marker style
markeredgecolor or mec		any matplotlib color
markeredgewidth or mew		float value in points
markerfacecolor or mfc		any matplotlib color
markerfacecoloralt or mfcalt		any matplotlib color
markersize or ms		float
markevery		[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]
path_effects		AbstractPathEffect
picker		float distance in points or callable pick function fn(artist, event)
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
solid_capstyle		['butt' 'round' 'projecting']
solid_joinstyle		['miter' 'round' 'bevel']
transform		a matplotlib.transforms.Transform instance
url		a url string
visible		bool
xdata		1D array
ydata		1D array
zorder		float

See also:

`scatter` XY scatter plot with markers of varying size and/or color (sometimes also called bubble chart).

Notes

Format Strings

A format string consists of a part for color, marker and line:

```
fmt = '[color][marker][line]'
```

Each of them is optional. If not provided, the value from the style cycle is used. Exception: If line is given, but no marker, the data will be a line without markers.

Colors

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

If the color is the only part of the format string, you can additionally use any `matplotlib.colors` spec, e.g. full names ('green') or hex strings ('#008000').

Markers

character	description
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
' '	vline marker
'_'	hline marker

Line Styles

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style

Example format strings:

```
'b'  # blue markers with default shape
'ro' # red circles
'g-'  # green solid line
'--'  # dashed line with default color
'k^:' # black triangle_up markers connected by a dotted line
```

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'x', 'y'.

Examples using `matplotlib.axes.Axes.plot`

- `sphinx_glr_gallery_color_color_by_yvalue.py`
- `sphinx_glr_gallery_color_color_demo.py`
- `sphinx_glr_gallery_color_color_cycler.py`
- `sphinx_glr_gallery_lines_bars_and_markers_simple_plot.py`
- `sphinx_glr_gallery_images_contours_and_fields_triinterp_demo.py`
- `sphinx_glr_gallery_shapes_and_collections_marker_path.py`
- `sphinx_glr_gallery_pie_and_polar_charts_polar_demo.py`
- `sphinx_glr_gallery_pie_and_polar_charts_polar_legend.py`

`matplotlib.axes.Axes.errorbar`

`Axes.errorbar(x, y, yerr=None, xerr=None, fmt=" ", ecol=None, elinewidth=None, capsize=None, barsabove=False, lolims=False, uplims=False, xlolims=False, xuplims=False, errorevery=1, capthick=None, *, data=None, **kwargs)`

Plot *y* versus *x* as lines and/or markers with attached errorbars.

x, *y* define the data locations, *xerr*, *yerr* define the errorbar sizes. By default, this draws the data markers/lines as well the errorbars. Use `fmt='none'` to draw errorbars without any data markers.

Parameters

x, *y* [scalar or array-like] The data positions.

xerr, *yerr* [scalar or array-like, `shape(N,)` or `shape(2,N)`, optional] The errorbar sizes:

- scalar: Symmetric +/- values for all data points.
- `shape(N,)`: Symmetric +/-values for each data point.
- `shape(2,N)`: Separate - and + values for each bar. First row contains the lower errors, the second row contains the upper errors.
- None: No errorbar.

See `/gallery/statistics/errorbar_features` for an example on the usage of *xerr* and *yerr*.

fmt [plot format string, optional, default: " "] The format for the data points / data lines. See `plot` for details.

Use 'none' (case insensitive) to plot errorbars without any data markers.

- `ecolor` [mpl color, optional, default: None] A matplotlib color arg which gives the color the errorbar lines. If None, use the color of the line connecting the markers.
- `linewidth` [scalar, optional, default: None] The linewidth of the errorbar lines. If None, the linewidth of the current style is used.
- `capsize` [scalar, optional, default: None] The length of the error bar caps in points. If None, it will take the value from `rcParams["errorbar.capsize"]`.
- `capthick` [scalar, optional, default: None] An alias to the keyword argument `markedgedwidth` (a.k.a. `mew`). This setting is a more sensible name for the property that controls the thickness of the error bar cap in points. For backwards compatibility, if `mew` or `markedgedwidth` are given, then they will over-ride `capthick`. This may change in future releases.
- `barsabove` [bool, optional, default: False] If True, will plot the errorbars above the plot symbols. Default is below.
- `lolims`, `uplims`, `xlolims`, `xuplims` [bool, optional, default: None] These arguments can be used to indicate that a value gives only upper/lower limits. In that case a caret symbol is used to indicate this. `lims`-arguments may be of the same type as `xerr` and `yerr`. To use limits with inverted axes, `set_xlim()` or `set_ylim()` must be called before `errorbar()`.
- `errorevery` [positive integer, optional, default: 1] Subsamples the errorbars. e.g., if `errorevery=5`, errorbars for every 5-th datapoint will be plotted. The data plot itself still shows all data points.

Returns

- `container` [ErrorbarContainer] The container contains:
- `plotline`: Line2D instance of x, y plot markers and/or line.
 - `caplines`: A tuple of Line2D instances of the error bar caps.
 - `barlinecols`: A tuple of LineCollection with the horizontal and vertical error ranges.

Other Parameters

- **kwargs** : All other keyword arguments are passed on to the plot command for the markers. For example, this code makes big red squares with thick green edges:

```
x,y,yerr = rand(3,10)
errorbar(x, y, yerr, marker='s', mfc='red',
         mec='green', ms=20, mew=4)
```

where `mfc`, `mec`, `ms` and `mew` are aliases for the longer property names, `markerfacecolor`, `markeredgecolor`, `markersize` and `markedgedwidth`.

Valid kwargs for the marker properties are Lines2D properties:

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)
alpha		float (0.0 transparent through 1.0 opaque)
animated		bool
antialiased or aa		bool
clip_box		a Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
color or c		any matplotlib color
contains		a callable function
dash_capstyle		['butt' 'round' 'projecting']
dash_joinstyle		['miter' 'round' 'bevel']
dashes		sequence of on/off ink in points
drawstyle		['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']
figure		a Figure instance
fillstyle		['full' 'left' 'right' 'bottom' 'top' 'none']
gid		an id string
label		object
linestyle or ls		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']
linewidth or lw		float value in points
marker		A valid marker style
markeredgecolor or mec		any matplotlib color
markeredgewidth or mew		float value in points
markerfacecolor or mfc		any matplotlib color
markerfacecoloralt or mfcalt		any matplotlib color
markersize or ms		float
markevery		[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]
path_effects		AbstractPathEffect
picker		float distance in points or callable pick function fn(artist, event)
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
solid_capstyle		['butt' 'round' 'projecting']
solid_joinstyle		['miter' 'round' 'bevel']
transform		a matplotlib.transforms.Transform instance
url		a url string
visible		bool
xdata		1D array
ydata		1D array
zorder		float

Notes

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'x', 'xerr', 'y', 'yerr'.
-

`matplotlib.axes.Axes.scatter`

`Axes.scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, verts=None, edgecolors=None, *, data=None, **kwargs)`

A scatter plot of y vs x with varying marker size and/or color.

Parameters

- `x, y` [array_like, shape (n,)] The data positions.
- `s` [scalar or array_like, shape (n,), optional] The marker size in points**2. Default is `rcParams['lines.markersize'] ** 2`.
- `c` [color, sequence, or sequence of color, optional, default: 'b'] The marker color. Possible values:
 - A single color format string.
 - A sequence of color specifications of length n.
 - A sequence of n numbers to be mapped to colors using `cmap` and `norm`.
 - A 2-D array in which the rows are RGB or RGBA.

Note that `c` should not be a single numeric RGB or RGBA sequence because that is indistinguishable from an array of values to be colormapped. If you want to specify the same RGB or RGBA value for all points, use a 2-D array with a single row.

- `marker` [MarkerStyle, optional, default: 'o'] The marker style. `marker` can be either an instance of the class or the text shorthand for a particular marker. See markers for more information marker styles.
- `cmap` [Colormap, optional, default: None] A Colormap instance or registered colormap name. `cmap` is only used if `c` is an array of floats. If None, defaults to `rc image.cmap`.
- `norm` [Normalize, optional, default: None] A Normalize instance is used to scale luminance data to 0, 1. `norm` is only used if `c` is an array of floats. If None, use the default `colors.Normalize`.
- `vmin, vmax` [scalar, optional, default: None] `vmin` and `vmax` are used in conjunction with `norm` to normalize luminance data. If None, the respective min and max of the color array is used. `vmin` and `vmax` are ignored if you pass a `norm` instance.

`alpha` [scalar, optional, default: None] The alpha blending value, between 0 (transparent) and 1 (opaque).

`linewidths` [scalar or array_like, optional, default: None] The linewidth of the marker edges. Note: The default edgecolors is 'face'. You may want to change this as well. If None, defaults to rcParams lines.linewidth.

`verts` [sequence of (x, y), optional] If marker is None, these vertices will be used to construct the marker. The center of the marker is located at (0, 0) in normalized units. The overall marker is rescaled by s.

`edgecolors` [color or sequence of color, optional, default: 'face'] The edge color of the marker. Possible values:

- 'face': The edge color will always be the same as the face color.
- 'none': No patch boundary will be drawn.
- A matplotlib color.

For non-filled markers, the edgecolors kwarg is ignored and forced to 'face' internally.

Returns

`paths` [PathCollection]

Other Parameters

`**kwargs` [Collection properties]

See also:

`plot` To plot scatter plots when markers are identical in size and color.

Notes

- The plot function will be faster for scatterplots where markers don't vary in size or color.
- Any or all of x, y, s, and c may be masked arrays, in which case all masks will be combined and only unmasked points will be plotted.
- Fundamentally, scatter works with 1-D arrays; x, y, s, and c may be input as 2-D arrays, but within scatter they will be flattened. The exception is c, which will be flattened only if its size matches the size of x and y.

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All arguments with the following names: 'c', 'color', 'edgecolors', 'facecolor', 'facecolors', 'linewidths', 's', 'x', 'y'.
-

Examples using `matplotlib.axes.Axes.scatter`

- `sphx_glr_gallery_shapes_and_collections_scatter.py`
- `sphx_glr_gallery_pie_and_polar_charts_polar_scatter.py`

`matplotlib.axes.Axes.plot_date`

`Axes.plot_date(x, y, fmt='o', tz=None, xdate=True, ydate=False, *, data=None, **kwargs)`
Plot data that contains dates.

Similar to `plot`, this plots `y` vs. `x` as lines or markers. However, the axis labels are formatted as dates depending on `xdate` and `ydate`.

Parameters

- `x, y` [array-like] The coordinates of the data points. If `xdate` or `ydate` is `True`, the respective values `x` or `y` are interpreted as Matplotlib dates.
- `fmt` [str, optional] The plot format string. For details, see the corresponding parameter in `plot`.
- `tz` [[None | timezone string | tzinfo instance]] The time zone to use in labeling dates. If `None`, defaults to rcParam timezone.
- `xdate` [bool, optional, default: `True`] If `True`, the x-axis will be interpreted as Matplotlib dates.
- `ydate` [bool, optional, default: `False`] If `True`, the y-axis will be interpreted as Matplotlib dates.

Returns

`lines` A list of `Line2D` objects representing the plotted data.

Other Parameters

`**kwargs` Keyword arguments control the `Line2D` properties:

	<table><tr><th>Property</th><th>Description</th></tr></table>	Property	Description
Property	Description		
<code>agg_filter</code>	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)		
<code>alpha</code>	float (0.0 transparent through 1.0 opaque)		
<code>animated</code>	bool		
<code>antialiased</code> or <code>aa</code>	bool		
<code>clip_box</code>	a Bbox instance		
<code>clip_on</code>	bool		
<code>clip_path</code>	[(Path, Transform) Patch None]		
<code>color</code> or <code>c</code>	any matplotlib color		
<code>contains</code>	a callable function		
<code>dash_capstyle</code>	['butt' 'round' 'projecting']		
<code>dash_joinstyle</code>	['miter' 'round' 'bevel']		

Continued on next page

Table 4 – continued from previous page

	Property	Description
dashes	sequence of on/off ink in points	
drawstyle	['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']	
figure	a Figure instance	
fillstyle	['full' 'left' 'right' 'bottom' 'top' 'none']	
gid	an id string	
label	object	
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']	
linewidth or lw	float value in points	
marker	A valid marker style	
markeredgcolor or mec	any matplotlib color	
markeredgewidth or mew	float value in points	
markerfacecolor or mfc	any matplotlib color	
markerfacecoloralt or mfcalt	any matplotlib color	
markersize or ms	float	
markevery	[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]	
path_effects	AbstractPathEffect	
picker	float distance in points or callable pick function fn(artist, event)	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
solid_capstyle	['butt' 'round' 'projecting']	
solid_joinstyle	['miter' 'round' 'bevel']	
transform	a matplotlib.transforms.Transform instance	
url	a url string	
visible	bool	
xdata	1D array	
ydata	1D array	
zorder	float	

See also:

`matplotlib.dates` Helper functions on dates.

`matplotlib.dates.date2num` Convert dates to num.

`matplotlib.dates.num2date` Convert num to dates.

`matplotlib.dates.drange` Create an equally spaced sequence of dates.

Notes

If you are using custom date tickers and formatters, it may be necessary to set the formatters/locators after the call to `plot_date`. `plot_date` will set the default tick locator to `AutoDateLocator` (if the tick locator is not already set to a `DateLocator` instance) and the

default tick formatter to `AutoDateFormatter` (if the tick formatter is not already set to a `DateFormatter` instance).

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'x', 'y'.
-

matplotlib.axes.Axes.step

`Axes.step(x, y, *args, data=None, **kwargs)`

Make a step plot.

Call signatures:

```
step(x, y, [fmt], *, data=None, where='pre', **kwargs)
step(x, y, [fmt], x2, y2, [fmt2], ..., *, where='pre', **kwargs)
```

This is just a thin wrapper around `plot` which changes some formatting options. Most of the concepts and parameters of `plot` can be used here as well.

Parameters

x [array_like] 1-D sequence of x positions. It is assumed, but not checked, that it is uniformly increasing.

y [array_like] 1-D sequence of y levels.

fmt [str, optional] A format string, e.g. 'g' for a green line. See `plot` for a more detailed description.

Note: While full format strings are accepted, it is recommended to only specify the color. Line styles are currently ignored (use the keyword argument `linestyle` instead). Markers are accepted and plotted on the given positions, however, this is a rarely needed feature for step plots.

data [indexable object, optional] An object with labelled data. If given, provide the label names to plot in x and y.

where [{ 'pre', 'post', 'mid' }, optional, default 'pre'] Define where the steps should be placed:

- 'pre': The y value is continued constantly to the left from every x position, i.e. the interval $(x[i-1], x[i])$ has the value $y[i]$.
- 'post': The y value is continued constantly to the right from every x position, i.e. the interval $[x[i], x[i+1])$ has the value $y[i]$.
- 'mid': Steps occur half-way between the x positions.

Returns

lines A list of Line2D objects representing the plotted data.

Other Parameters

****kwargs** Additional parameters are the same as those for plot.

Notes

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All arguments with the following names: 'x', 'y'.
-

matplotlib.axes.Axes.loglog

Axes.loglog(*args, **kwargs)

Make a plot with log scaling on both the x and y axis.

Call signatures:

```
loglog([x], y, [fmt], data=None, **kwargs)
loglog([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

This is just a thin wrapper around plot which additionally changes both the x-axis and the y-axis to log scaling. All of the concepts and parameters of plot can be used here as well.

The additional parameters basex/y, subsx/y and nonposx/y control the x/y-axis properties. They are just forwarded to Axes.set_xscale and Axes.set_yscale.

Parameters

basex, basey [scalar, optional, default 10] Base of the x/y logarithm.

subsx, subsy [sequence, optional] The location of the minor x/y ticks. If None, reasonable locations are automatically chosen depending on the number of decades in the plot. See Axes.set_xscale / Axes.set_yscale for details.

nonposx, nonposy [{ 'mask', 'clip' }, optional, default 'mask'] Non-positive values in x or y can be masked as invalid, or clipped to a very small positive number.

Returns

lines A list of Line2D objects representing the plotted data.

Other Parameters

****kwargs** All parameters supported by plot.

`matplotlib.axes.Axes.semilogx``Axes.semilogx(*args, **kwargs)`

Make a plot with log scaling on the x axis.

Call signatures:

```
semilogx([x], y, [fmt], data=None, **kwargs)
semilogx([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

This is just a thin wrapper around plot which additionally changes the x-axis to log scaling. All of the concepts and parameters of plot can be used here as well.

The additional parameters `basex`, `subsx` and `nonposx` control the x-axis properties. They are just forwarded to `Axes.set_xscale`.

Parameters

`basex` [scalar, optional, default 10] Base of the x logarithm.

`subsx` [array_like, optional] The location of the minor xticks. If None, reasonable locations are automatically chosen depending on the number of decades in the plot. See `Axes.set_xscale` for details.

`nonposx` [{`'mask'`, `'clip'`}, optional, default `'mask'`] Non-positive values in x can be masked as invalid, or clipped to a very small positive number.

Returns

`lines` A list of `Line2D` objects representing the plotted data.

Other Parameters

`**kwargs` All parameters supported by plot.

`matplotlib.axes.Axes.semilogy``Axes.semilogy(*args, **kwargs)`

Make a plot with log scaling on the y axis.

Call signatures:

```
semilogy([x], y, [fmt], data=None, **kwargs)
semilogy([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

This is just a thin wrapper around plot which additionally changes the y-axis to log scaling. All of the concepts and parameters of plot can be used here as well.

The additional parameters `basey`, `subsy` and `nonposy` control the y-axis properties. They are just forwarded to `Axes.set_yscale`.

Parameters

`basey` [scalar, optional, default 10] Base of the y logarithm.

`subsy` [array_like, optional] The location of the minor yticks. If None, reasonable locations are automatically chosen depending on the number of decades in the plot. See `Axes.set_yscale` for details.

`nonposy` [{`'mask'`, `'clip'`}, optional, default `'mask'`] Non-positive values in y can be masked as invalid, or clipped to a very small positive number.

Returns

`lines` A list of `Line2D` objects representing the plotted data.

Other Parameters

`**kwargs` All parameters supported by `plot`.

`matplotlib.axes.Axes.fill_between`

`Axes.fill_between(x, y1, y2=0, where=None, interpolate=False, step=None, *, data=None, **kwargs)`

Fill the area between two horizontal curves.

The curves are defined by the points $(x, y1)$ and $(x, y2)$. This creates one or multiple polygons describing the filled area.

You may exclude some horizontal sections from filling using `where`.

By default, the edges connect the given points directly. Use `step` if the filling should be a step function, i.e. constant in between `x`.

Parameters

`x` [array (length N)] The x coordinates of the nodes defining the curves.

`y1` [array (length N) or scalar] The y coordinates of the nodes defining the first curve.

`y2` [array (length N) or scalar, optional, default: 0] The y coordinates of the nodes defining the second curve.

`where` [array of bool (length N), optional, default: None] Define where to exclude some horizontal regions from being filled. The filled regions are defined by the coordinates `x[where]`. More precisely, fill between `x[i]` and `x[i+1]` if `where[i]` and `where[i+1]`. Note that this definition implies that an isolated True value between two False values in `where` will not result in filling. Both sides of the True position remain unfilled due to the adjacent False values.

`interpolate` [bool, optional] This option is only relevant if `where` is used and the two curves are crossing each other.

Semantically, `where` is often used for $y1 > y2$ or similar. By default, the nodes of the polygon defining the filled region will only be placed at the positions in the `x` array. Such a polygon cannot describe the above semantics close to the intersection. The x-sections containing the intersection are simply clipped.

Setting `interpolate` to `True` will calculate the actual intersection point and extend the filled region up to this point.

`step` [{`'pre'`, `'post'`, `'mid'`}, optional] Define step if the filling should be a step function, i.e. constant in between `x`. The value determines where the step will occur:

- `'pre'`: The `y` value is continued constantly to the left from every `x` position, i.e. the interval $(x[i-1], x[i])$ has the value `y[i]`.
- `'post'`: The `y` value is continued constantly to the right from every `x` position, i.e. the interval $[x[i], x[i+1])$ has the value `y[i]`.
- `'mid'`: Steps occur half-way between the `x` positions.

Returns

‘PolyCollection’ A `PolyCollection` containing the plotted polygons.

Other Parameters

****kwargs** All other keyword arguments are passed on to `PolyCollection`. They control the `Polygon` properties:

	Property	Description
<code>agg_filter</code>		a filter function, which takes a $(m, n, 3)$ float array and a dpi value, and returns a
<code>alpha</code>		float or None
<code>animated</code>		bool
<code>antialiased</code> or <code>antialiaseds</code>		Boolean or sequence of booleans
<code>array</code>		ndarray
<code>capstyle</code>		unknown
<code>clim</code>		a length 2 sequence of floats; may be overridden in methods that have <code>vmin</code> and <code>vm</code>
<code>clip_box</code>		a <code>Bbox</code> instance
<code>clip_on</code>		bool
<code>clip_path</code>		$[(\text{Path}, \text{Transform}) \mid \text{Patch} \mid \text{None}]$
<code>cmap</code>		a colormap or registered colormap name
<code>color</code>		matplotlib color arg or sequence of rgba tuples
<code>contains</code>		a callable function
<code>edgecolor</code> or <code>edgecolors</code>		matplotlib color spec or sequence of specs
<code>facecolor</code> or <code>facecolors</code>		matplotlib color spec or sequence of specs
<code>figure</code>		a <code>Figure</code> instance
<code>gid</code>		an id string
<code>hatch</code>		$[\text{'/'} \mid \text{'\'} \mid \text{' '} \mid \text{'-'} \mid \text{'+'} \mid \text{'x'} \mid \text{'o'} \mid \text{'O'} \mid \text{'.'} \mid \text{'*'}]$
<code>joinstyle</code>		unknown
<code>label</code>		object
<code>linestyle</code> or <code>dashes</code> or <code>linestyles</code>		$[\text{'solid'} \mid \text{'dashed'}, \text{'dashdot'}, \text{'dotted'} \mid (\text{offset}, \text{on-off-dash-seq}) \mid \text{'-'} \mid \text{'--'} \mid \text{'-.'} \mid \text{'.'}]$
<code>linewidth</code> or <code>linewidths</code> or <code>lw</code>		float or sequence of floats
<code>norm</code>		Normalize
<code>offset_position</code>		$[\text{'screen'} \mid \text{'data'}]$

Continued on next page

Table 5 – continued from previous page

	Property	Description
offsets		float or sequence of floats
path_effects		AbstractPathEffect
picker		[None bool float callable]
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
transform		Transform
url		a url string
urls		List[str] or None
visible		bool
zorder		float

See also:

`fill_betweenx` Fill between two sets of x-values.

Notes

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'where', 'x', 'y1', 'y2'.
-

`matplotlib.axes.Axes.fill_betweenx`

`Axes.fill_betweenx(y, x1, x2=0, where=None, step=None, interpolate=False, *, data=None, **kwargs)`

Fill the area between two vertical curves.

The curves are defined by the points $(x1, y)$ and $(x2, y)$. This creates one or multiple polygons describing the filled area.

You may exclude some vertical sections from filling using `where`.

By default, the edges connect the given points directly. Use `step` if the filling should be a step function, i.e. constant in between `y`.

Parameters

`y` [array (length N)] The y coordinates of the nodes defining the curves.

`x1` [array (length N) or scalar] The x coordinates of the nodes defining the first curve.

`x2` [array (length N) or scalar, optional, default: 0] The x coordinates of the nodes defining the second curve.

`where` [array of bool (length N), optional, default: None] Define where to exclude some vertical regions from being filled. The filled regions are defined by the coordinates `y[where]`. More precisely, fill between `y[i]` and `y[i+1]` if `where[i]` and `where[i+1]`. Note that this definition implies that an isolated True value between two False values in `where` will not result in filling. Both sides of the True position remain unfilled due to the adjacent False values.

`interpolate` [bool, optional] This option is only relevant if `where` is used and the two curves are crossing each other.

Semantically, `where` is often used for `x1 > x2` or similar. By default, the nodes of the polygon defining the filled region will only be placed at the positions in the `y` array. Such a polygon cannot describe the above semantics close to the intersection. The y-sections containing the intersection are simply clipped.

Setting `interpolate` to True will calculate the actual intersection point and extend the filled region up to this point.

`step` [{`'pre'`, `'post'`, `'mid'`}, optional] Define step if the filling should be a step function, i.e. constant in between `y`. The value determines where the step will occur:

- `'pre'`: The y value is continued constantly to the left from every x position, i.e. the interval `(x[i-1], x[i]]` has the value `y[i]`.
- `'post'`: The y value is continued constantly to the right from every x position, i.e. the interval `[x[i], x[i+1))` has the value `y[i]`.
- `'mid'`: Steps occur half-way between the x positions.

Returns

`'PolyCollection'` A `PolyCollection` containing the plotted polygons.

Other Parameters

`**kwargs` All other keyword arguments are passed on to `PolyCollection`. They control the `Polygon` properties:

	Property	Description
<code>agg_filter</code>		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) float array
<code>alpha</code>		float or None
<code>animated</code>		bool
<code>antialiased</code> or <code>antialiaseds</code>		Boolean or sequence of booleans
<code>array</code>		ndarray
<code>capstyle</code>		unknown
<code>clim</code>		a length 2 sequence of floats; may be overridden in methods that have <code>vmin</code> and <code>vmax</code>
<code>clip_box</code>		a <code>Bbox</code> instance

Continued on next page

Table 6 – continued from previous page

	Property	Description
clip_on	bool	
clip_path	[(Path, Transform) Patch None]	
cmap	a colormap or registered colormap name	
color	matplotlib color arg or sequence of rgba tuples	
contains	a callable function	
edgecolor or edgecolors	matplotlib color spec or sequence of specs	
facecolor or facecolors	matplotlib color spec or sequence of specs	
figure	a Figure instance	
gid	an id string	
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']	
joinstyle	unknown	
label	object	
linestyle or dashes or linestyles	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' '']	
linewidth or linewidths or lw	float or sequence of floats	
norm	Normalize	
offset_position	['screen' 'data']	
offsets	float or sequence of floats	
path_effects	AbstractPathEffect	
picker	[None bool float callable]	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
transform	Transform	
url	a url string	
urls	List[str] or None	
visible	bool	
zorder	float	

See also:

`fill_between` Fill between two sets of y-values.

Notes

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'where', 'x1', 'x2', 'y'.

matplotlib.axes.Axes.bar

Axes.bar(*args, data=None, **kwargs)

Make a bar plot.

Call signatures:

```
bar(x, height, *, align='center', **kwargs)
bar(x, height, width, *, align='center', **kwargs)
bar(x, height, width, bottom, *, align='center', **kwargs)
```

The bars are positioned at x with the given alignment. Their dimensions are given by width and height. The vertical baseline is bottom (default 0).

Each of x, height, width, and bottom may either be a scalar applying to all bars, or it may be a sequence of length N providing a separate value for each bar.

Parameters

x [sequence of scalars] The x coordinates of the bars. See also align for the alignment of the bars to the coordinates.

height [scalar or sequence of scalars] The height(s) of the bars.

width [scalar or array-like, optional] The width(s) of the bars (default: 0.8).

bottom [scalar or array-like, optional] The y coordinate(s) of the bars bases (default: 0).

align [{‘center’, ‘edge’}, optional, default: ‘center’] Alignment of the bars to the x coordinates:

- ‘center’: Center the base on the x positions.
- ‘edge’: Align the left edges of the bars with the x positions.

To align the bars on the right edge pass a negative width and align=‘edge’.

Returns

container [BarContainer] Container with all the bars and optionally errorbars.

Other Parameters

color [scalar or array-like, optional] The colors of the bar faces.

edgecolor [scalar or array-like, optional] The colors of the bar edges.

linewidth [scalar or array-like, optional] Width of the bar edge(s). If 0, don’t draw edges.

tick_label [string or array-like, optional] The tick labels of the bars. Default: None (Use default numeric labels.)

xerr, yerr [scalar or array-like of shape(N,) or shape(2,N), optional] If not None, add horizontal / vertical errorbars to the bar tips. The values are +/- sizes relative to the data:

- scalar: symmetric +/- values for all bars
- shape(N,): symmetric +/- values for each bar
- shape(2,N): Separate - and + values for each bar. First row contains the lower errors, the second row contains the upper errors.
- None: No errorbar. (Default)

See /gallery/statistics/errorbar_features for an example on the usage of `xerr` and `yerr`.

`ecolor` [scalar or array-like, optional, default: 'black'] The line color of the errorbars.

`capsize` [scalar, optional] The length of the error bar caps in points. Default: None, which will take the value from `rcParams["errorbar.capsize"]`.

`error_kw` [dict, optional] Dictionary of kwargs to be passed to the `errorbar` method. Values of `ecolor` or `capsize` defined here take precedence over the independent kwargs.

`log` [bool, optional, default: False] If True, set the y-axis to be log scale.

`orientation` [{'vertical', 'horizontal'}, optional] This is for internal use only. Please use `barh` for horizontal bar plots. Default: 'vertical'.

See also:

`barh` Plot a horizontal bar plot.

Notes

The optional arguments `color`, `edgecolor`, `linewidth`, `xerr`, and `yerr` can be either scalars or sequences of length equal to the number of bars. This enables you to use `bar` as the basis for stacked bar charts, or candlestick plots. Detail: `xerr` and `yerr` are passed directly to `errorbar()`, so they can also have shape `2xN` for independent specification of lower and upper errors.

Other optional kwargs:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\ ' ' ' '- ' '+ ' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All arguments with the following names: 'bottom', 'color', 'ecolor', 'edgecolor', 'height', 'left', 'linewidth', 'tick_label', 'width', 'x', 'xerr', 'y', 'yerr'.
- All positional arguments.

Examples using `matplotlib.axes.Axes.bar`

- `sphinx_glr_gallery_shapes_and_collections_hatch_demo.py`
- `sphinx_glr_gallery_pie_and_polar_charts_polar_bar.py`
- `sphinx_glr_gallery_pie_and_polar_charts_nested_pie.py`

`matplotlib.axes.Axes.barh``Axes.barh(*args, **kwargs)`

Make a horizontal bar plot.

Call signatures:

```
bar(y, width, *, align='center', **kwargs)
bar(y, width, height, *, align='center', **kwargs)
bar(y, width, height, left, *, align='center', **kwargs)
```

The bars are positioned at `y` with the given `align`. Their dimensions are given by `width` and `height`. The horizontal baseline is left (default 0).

Each of `y`, `width`, `height`, and `left` may either be a scalar applying to all bars, or it may be a sequence of length `N` providing a separate value for each bar.

Parameters

`y` [scalar or array-like] The `y` coordinates of the bars. See also `align` for the alignment of the bars to the coordinates.

`width` [scalar or array-like] The width(s) of the bars.

`height` [sequence of scalars, optional, default: 0.8] The heights of the bars.

`left` [sequence of scalars] The `x` coordinates of the left sides of the bars (default: 0).

`align` [{`'center'`, `'edge'`}, optional, default: `'center'`] Alignment of the base to the `y` coordinates*:

- `'center'`: Center the bars on the `y` positions.
- `'edge'`: Align the bottom edges of the bars with the `y` positions.

To align the bars on the top edge pass a negative height and `align='edge'`.

Returns

`container` [BarContainer] Container with all the bars and optionally errorbars.

Other Parameters

`color` [scalar or array-like, optional] The colors of the bar faces.

`edgecolor` [scalar or array-like, optional] The colors of the bar edges.

`linewidth` [scalar or array-like, optional] Width of the bar edge(s). If 0, don't draw edges.

`tick_label` [string or array-like, optional] The tick labels of the bars. Default: None (Use default numeric labels.)

`xerr, yerr` [scalar or array-like of shape(N,) or shape(2,N), optional] If not None, add horizontal / vertical errorbars to the bar tips. The values are +/- sizes relative to the data:

- scalar: symmetric +/- values for all bars
- shape(N,): symmetric +/- values for each bar
- shape(2,N): Separate - and + values for each bar. First row contains the lower errors, the second row contains the upper errors.
- None: No errorbar. (default)

See /gallery/statistics/errorbar_features for an example on the usage of `xerr` and `yerr`.

`ecolor` [scalar or array-like, optional, default: 'black'] The line color of the errorbars.

`capsize` [scalar, optional] The length of the error bar caps in points. Default: None, which will take the value from `rcParams["errorbar.capsize"]`.

`error_kw` [dict, optional] Dictionary of kwargs to be passed to the errorbar method. Values of `ecolor` or `capsize` defined here take precedence over the independent kwargs.

`log` [bool, optional, default: False] If True, set the x-axis to be log scale.

See also:

`bar` Plot a vertical bar plot.

Notes

The optional arguments `color`, `edgecolor`, `linewidth`, `xerr`, and `yerr` can be either scalars or sequences of length equal to the number of bars. This enables you to use `bar` as the basis for stacked bar charts, or candlestick plots. Detail: `xerr` and `yerr` are passed directly to `errorbar()`, so they can also have shape `2xN` for independent specification of lower and upper errors.

Other optional kwargs:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '.' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

matplotlib.axes.Axes.stem

Axes.stem(*args, data=None, **kwargs)

Create a stem plot.

A stem plot plots vertical lines at each x location from the baseline to y, and places a marker there.

Call signature:

```
stem([x,] y, linefmt=None, markerfmt=None, basefmt=None)
```

The x-positions are optional. The formats may be provided either as positional or as keyword-arguments.

Parameters

- x** [array-like, optional] The x-positions of the stems. Default: (0, 1, ..., len(y) - 1).
- y** [array-like] The y-values of the stem heads.
- linefmt** [str, optional] A string defining the properties of the vertical lines. Usually, this will be a color or a color and a linestyle:

Character	Line Style
'-'	solid line
'--'	dashed line
'-.'	dash-dot line
'.'	dotted line

Default: 'C0-', i.e. solid line with the first color of the color cycle.

Note: While it is technically possible to specify valid formats other than color or color and linestyle (e.g. 'rx' or '-.'), this is beyond the intention of the method and will most likely not result in a reasonable reasonable plot.

markerfmt [str, optional] A string defining the properties of the markers at the stem heads. Default: 'C0o', i.e. filled circles with the first color of the color cycle.

basefmt [str, optional] A format string defining the properties of the baseline.

Default: 'C3-' ('C2-' in classic mode).

bottom [float, optional, default: 0] The y-position of the baseline.

label [str, optional, default: None] The label to use for the stems in legends.

Returns

container [StemContainer] The container may be treated like a tuple (markerline, stemlines, baseline)

Other Parameters

****kwargs** No other parameters are supported. They are currently ignored silently for backward compatibility. This behavior is deprecated. Future versions will not accept any other parameters and will raise a `TypeError` instead.

Notes

See also:

The MATLAB function `stem` which inspired this method.

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All positional and all keyword arguments.
-

`matplotlib.axes.Axes.eventplot`

`Axes.eventplot(positions, orientation='horizontal', lineoffsets=1, linelengths=1, linewidths=None, colors=None, linestyle='solid', *, data=None, **kwargs)`
Plot identical parallel lines at the given positions.

`positions` should be a 1D or 2D array-like object, with each row corresponding to a row or column of lines.

This type of plot is commonly used in neuroscience for representing neural events, where it is usually called a spike raster, dot raster, or raster plot.

However, it is useful in any situation where you wish to show the timing or position of multiple sets of discrete events, such as the arrival times of people to a business on each day of the month or the date of hurricanes each year of the last century.

Parameters

`positions` [1D or 2D array-like object] Each value is an event. If `positions` is a 2D array-like, each row corresponds to a row or a column of lines (depending on the `orientation` parameter).

`orientation` [{`'horizontal'`, `'vertical'`}, optional] Controls the direction of the event collections:

- `'horizontal'` : the lines are arranged horizontally in rows, and are vertical.
- `'vertical'` : the lines are arranged vertically in columns, and are horizontal.

`lineoffsets` [scalar or sequence of scalars, optional, default: 1] The offset of the center of the lines from the origin, in the direction orthogonal to `orientation`.

`linelengths` [scalar or sequence of scalars, optional, default: 1] The total height of the lines (i.e. the lines stretches from `lineoffset - linelength/2` to `lineoffset + linelength/2`).

`linewidths` [scalar, scalar sequence or None, optional, default: None] The line width(s) of the event lines, in points. If it is None, defaults to its `rcParams` setting.

colors [color, sequence of colors or None, optional, default: None] The color(s) of the event lines. If it is None, defaults to its rcParams setting.

linestyles [str or tuple or a sequence of such values, optional] Default is 'solid'. Valid strings are ['solid', 'dashed', 'dashdot', 'dotted', '-', '--', '-.', ':']. Dash tuples should be of the form:

(offset, onoffseq),

where onoffseq is an even length tuple of on and off ink in points.

****kwargs** [optional] Other keyword arguments are line collection properties. See LineCollection for a list of the valid properties.

Returns

list [A list of EventCollection objects.] Contains the EventCollection that were added.

Notes

For `linelengths`, `linewidths`, `colors`, and `linestyles`, if only a single value is given, that value is applied to all lines. If an array-like is given, it must have the same length as `positions`, and each value will be applied to the corresponding row of the array.

Examples

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

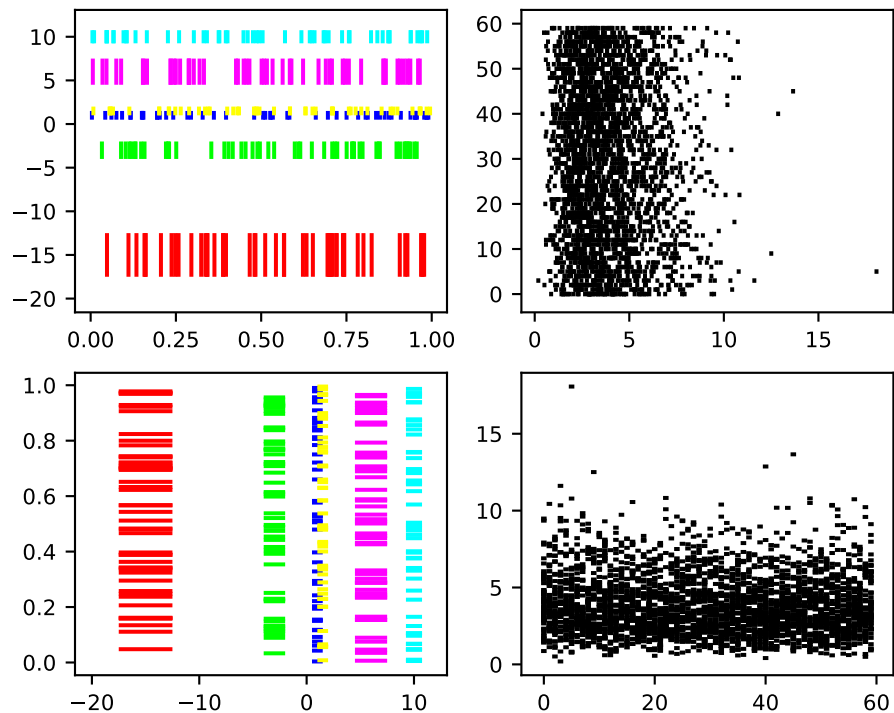
- All arguments with the following names: 'colors', 'linelengths', 'lineoffsets', 'linestyles', 'linewidths', 'positions'.
-

matplotlib.axes.Axes.pie

`Axes.pie(x, explode=None, labels=None, colors=None, autopct=None, pctdistance=0.6, shadow=False, labeldistance=1.1, startangle=None, radius=None, counter-clock=True, wedgeprops=None, textprops=None, center=(0, 0), frame=False, rotatelabels=False, *, data=None)`
Plot a pie chart.

Make a pie chart of array `x`. The fractional area of each wedge is given by `x/sum(x)`. If `sum(x) < 1`, then the values of `x` give the fractional area directly and the array will not be normalized. The resulting pie will have an empty wedge of size `1 - sum(x)`.

The wedges are plotted counterclockwise, by default starting from the x-axis.



Parameters

`x` [array-like] The wedge sizes.

`explode` [array-like, optional, default: None] If not None, is a `len(x)` array which specifies the fraction of the radius with which to offset each wedge.

`labels` [list, optional, default: None] A sequence of strings providing the labels for each wedge

`colors` [array-like, optional, default: None] A sequence of matplotlib color args through which the pie chart will cycle. If None, will use the colors in the currently active cycle.

`autopct` [None (default), string, or function, optional] If not None, is a string or function used to label the wedges with their numeric value. The label will be placed inside the wedge. If it is a format string, the label will be `fmt%pct`. If it is a function, it will be called.

`pctdistance` [float, optional, default: 0.6] The ratio between the center of each pie slice and the start of the text generated by `autopct`. Ignored if `autopct` is None.

`shadow` [bool, optional, default: False] Draw a shadow beneath the pie.

`labeldistance` [float, optional, default: 1.1] The radial distance at which the

pie labels are drawn

`startangle` [float, optional, default: None] If not None, rotates the start of the pie chart by angle degrees counterclockwise from the x-axis.

`radius` [float, optional, default: None] The radius of the pie, if radius is None it will be set to 1.

`counterclock` [bool, optional, default: True] Specify fractions direction, clockwise or counterclockwise.

`wedgeprops` [dict, optional, default: None] Dict of arguments passed to the wedge objects making the pie. For example, you can pass in `wedgeprops = {'linewidth': 3}` to set the width of the wedge border lines equal to 3. For more details, look at the doc/arguments of the wedge object. By default `clip_on=False`.

`textprops` [dict, optional, default: None] Dict of arguments to pass to the text objects.

`center` [list of float, optional, default: (0, 0)] Center position of the chart. Takes value (0, 0) or is a sequence of 2 scalars.

`frame` [bool, optional, default: False] Plot axes frame with the chart if true.

`rotatelabels` [bool, optional, default: False] Rotate each label to the angle of the corresponding slice if true.

Returns

`patches` [list] A sequence of `matplotlib.patches.Wedge` instances

`texts` [list] A list of the label `matplotlib.text.Text` instances.

`autotexts` [list] A list of Text instances for the numeric labels. This will only be returned if the parameter `autopct` is not None.

Notes

The pie chart will probably look best if the figure and axes are square, or the Axes aspect is equal.

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'colors', 'explode', 'labels', 'x'.
-

Examples using `matplotlib.axes.Axes.pie`

- `sphx_glr_gallery_pie_and_polar_charts_pie_features.py`

- sphx_glr_gallery_pie_and_polar_charts_nested_pie.py
- sphx_glr_gallery_pie_and_polar_charts_pie_and_donut_labels.py

matplotlib.axes.Axes.stackplot

Axes.stackplot(x, *args, data=None, **kwargs)

Draws a stacked area plot.

Parameters

x [1d array of dimension N]

y [2d array (dimension MxN), or sequence of 1d arrays (each dimension 1xN)]
The data is assumed to be unstacked. Each of the following calls is legal:

```
stackplot(x, y)           # where y is MxN
stackplot(x, y1, y2, y3, y4) # where y1, y2, y3, y4, are all 1xNm
```

baseline [['zero' | 'sym' | 'wiggle' | 'weighted_wiggle']] Method used to calculate the baseline:

- 'zero': Constant zero baseline, i.e. a simple stacked plot.
- 'sym': Symmetric around zero and is sometimes called 'ThemeRiver'.
- 'wiggle': Minimizes the sum of the squared slopes.
- 'weighted_wiggle': Does the same but weights to account for size of each layer. It is also called 'Streamgraph'-layout. More details can be found at <http://leebyron.com/streamgraph/>.

labels [Length N sequence of strings] Labels to assign to each data series.

colors [Length N sequence of colors] A list or tuple of colors. These will be cycled through and used to colour the stacked areas.

**kwargs : All other keyword arguments are passed to Axes.fill_between().

Returns

list [list of PolyCollection] A list of PolyCollection instances, one for each element in the stacked area plot.

matplotlib.axes.Axes.broken_barh

Axes.broken_barh(xranges, yrange, *, data=None, **kwargs)

Plot a horizontal sequence of rectangles.

A rectangle is drawn for each element of xrange. All rectangles have the same vertical position and size defined by yrange.

This is a convenience function for instantiating a BrokenBarHCollection, adding it to the axes and autoscaling the view.

Parameters

`xranges` [sequence of tuples (xmin, xwidth)] The x-positions and extends of the rectangles. For each tuple (xmin, xwidth) a rectangle is drawn from xmin to xmin + xwidth.

`yranges` [(ymin, ymax)] The y-position and extend for all the rectangles.

Returns

`collection` [A `BrokenBarHCollection`]

Other Parameters

`**kwargs` [`BrokenBarHCollection` properties] Each kwarg can be either a single argument applying to all rectangles, e.g.:

```
facecolors='black'
```

or a sequence of arguments over which is cycled, e.g.:

```
facecolors=('black', 'blue')
```

would create interleaving black and blue rectangles.

Supported keywords:

	Property	Description
<code>agg_filter</code>		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a
<code>alpha</code>		float or None
<code>animated</code>		bool
<code>antialiased</code> or <code>antialiaseds</code>		Boolean or sequence of booleans
<code>array</code>		ndarray
<code>capstyle</code>		unknown
<code>clim</code>		a length 2 sequence of floats; may be overridden in methods that have <code>vmin</code> and <code>vm</code>
<code>clip_box</code>		a <code>Bbox</code> instance
<code>clip_on</code>		bool
<code>clip_path</code>		[(<code>Path</code> , <code>Transform</code>) <code>Patch</code> <code>None</code>]
<code>cmap</code>		a colormap or registered colormap name
<code>color</code>		matplotlib color arg or sequence of rgba tuples
<code>contains</code>		a callable function
<code>edgecolor</code> or <code>edgecolors</code>		matplotlib color spec or sequence of specs
<code>facecolor</code> or <code>facecolors</code>		matplotlib color spec or sequence of specs
<code>figure</code>		a <code>Figure</code> instance
<code>gid</code>		an id string
<code>hatch</code>		['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
<code>joinstyle</code>		unknown
<code>label</code>		object
<code>linestyle</code> or <code>dashes</code> or <code>linestyles</code>		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' '']
<code>linewidth</code> or <code>linewidths</code> or <code>lw</code>		float or sequence of floats

Continued on next page

Table 7 – continued from previous page

	Property	Description
norm		Normalize
offset_position		['screen' 'data']
offsets		float or sequence of floats
path_effects		AbstractPathEffect
picker		[None bool float callable]
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
transform		Transform
url		a url string
urls		List[str] or None
visible		bool
zorder		float

Notes

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All positional and all keyword arguments.

matplotlib.axes.Axes.vlines

Axes.vlines(x, ymin, ymax, colors='k', linestyle='solid', label="", *, data=None, **kwargs)
Plot vertical lines.

Plot vertical lines at each x from ymin to ymax.

Parameters

x [scalar or 1D array_like] x-indexes where to plot the lines.

ymin, ymax [scalar or 1D array_like] Respective beginning and end of each line. If scalars are provided, all lines will have same length.

colors [array_like of colors, optional, default: 'k']

linestyle [['solid' | 'dashed' | 'dashdot' | 'dotted'], optional]

label [string, optional, default: ""]

Returns

lines [LineCollection]

Other Parameters

****kwargs** [LineCollection properties.]

See also:

`hlines` horizontal lines

`axvline` vertical line across the axes

In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`: * All arguments with the following names: 'colors', 'x', 'ymax', 'ymin'.

`matplotlib.axes.Axes.hlines`

`Axes.hlines(y, xmin, xmax, colors='k', linestyle='solid', label="", *, data=None, **kwargs)`
Plot horizontal lines at each y from xmin to xmax.

Parameters

`y` [scalar or sequence of scalar] y-indexes where to plot the lines.

`xmin, xmax` [scalar or 1D array_like] Respective beginning and end of each line. If scalars are provided, all lines will have same length.

`colors` [array_like of colors, optional, default: 'k']

`linestyle` [['solid' | 'dashed' | 'dashdot' | 'dotted'], optional]

`label` [string, optional, default: ""]

Returns

`lines` [LineCollection]

Other Parameters

****kwargs** [LineCollection properties.]

See also:

`vlines` vertical lines

`axhline` horizontal line across the axes

In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`: * All arguments with the following names: 'colors', 'xmax', 'xmin', 'y'.

Examples using `matplotlib.axes.Axes.hlines`

- `sphinx_glr_gallery_color_named_colors.py`

matplotlib.axes.Axes.fill

Axes.fill(*args, data=None, **kwargs)

Plot filled polygons.

Parameters

args [sequence of x, y, [color]] Each polygon is defined by the lists of x and y positions of its nodes, optionally followed by a color specifier. See matplotlib.colors for supported color specifiers. The standard color cycle is used for polygons without a color specifier.

You can plot multiple polygons by providing multiple x, y, [color] groups.

For example, each of the following is legal:

ax.fill(x, y)	# a polygon with default color
ax.fill(x, y, "b")	# a blue polygon
ax.fill(x, y, x2, y2)	# two polygons
ax.fill(x, y, "b", x2, y2, "r")	# a blue and a red polygon

Returns

a list of :class:`~matplotlib.patches.Polygon`

Other Parameters

**kwargs [Polygon properties]

Notes

Use fill_between() if you would like to fill the region between two curves.

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All arguments with the following names: 'x', 'y'.
-

35.1.2 Spans

Axes.axhline	Add a horizontal line across the axis.
Axes.axhspan	Add a horizontal span (rectangle) across the axis.
Axes.axvline	Add a vertical line across the axes.
Axes.axvspan	Add a vertical span (rectangle) across the axes.

`matplotlib.axes.Axes.axhline``Axes.axhline(y=0, xmin=0, xmax=1, **kwargs)`

Add a horizontal line across the axis.

Parameters

`y` [scalar, optional, default: 0] `y` position in data coordinates of the horizontal line.`xmin` [scalar, optional, default: 0] Should be between 0 and 1, 0 being the far left of the plot, 1 the far right of the plot.`xmax` [scalar, optional, default: 1] Should be between 0 and 1, 0 being the far left of the plot, 1 the far right of the plot.

Returns

`line` [Line2D]

Other Parameters

`**kwargs` : Valid kwargs are Line2D properties, with the exception of 'transform':

	Property	Description
<code>agg_filter</code>		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)
<code>alpha</code>		float (0.0 transparent through 1.0 opaque)
<code>animated</code>		bool
<code>antialiased</code> or <code>aa</code>		bool
<code>clip_box</code>		a Bbox instance
<code>clip_on</code>		bool
<code>clip_path</code>		[(Path, Transform) Patch None]
<code>color</code> or <code>c</code>		any matplotlib color
<code>contains</code>		a callable function
<code>dash_capstyle</code>		['butt' 'round' 'projecting']
<code>dash_joinstyle</code>		['miter' 'round' 'bevel']
<code>dashes</code>		sequence of on/off ink in points
<code>drawstyle</code>		['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']
<code>figure</code>		a Figure instance
<code>fillstyle</code>		['full' 'left' 'right' 'bottom' 'top' 'none']
<code>gid</code>		an id string
<code>label</code>		object
<code>linestyle</code> or <code>ls</code>		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']
<code>linewidth</code> or <code>lw</code>		float value in points
<code>marker</code>		A valid marker style
<code>markeredgecolor</code> or <code>mec</code>		any matplotlib color
<code>markeredgewidth</code> or <code>mew</code>		float value in points
<code>markerfacecolor</code> or <code>mfc</code>		any matplotlib color
<code>markerfacecoloralt</code> or <code>mfcalt</code>		any matplotlib color

Continued on next page

Table 9 – continued from previous page

	Property	Description
markersize or ms	float	
markevery	[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]	
path_effects	AbstractPathEffect	
picker	float distance in points or callable pick function fn(artist, event)	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
solid_capstyle	['butt' 'round' 'projecting']	
solid_joinstyle	['miter' 'round' 'bevel']	
transform	a matplotlib.transforms.Transform instance	
url	a url string	
visible	bool	
xdata	1D array	
ydata	1D array	
zorder	float	

See also:

`hlines` Add horizontal lines in data coordinates.

`axhspan` Add a horizontal span (rectangle) across the axis.

Examples

- draw a thick red hline at `'y' = 0` that spans the xrange:

```
>>> axhline(linewidth=4, color='r')
```

- draw a default hline at `'y' = 1` that spans the xrange:

```
>>> axhline(y=1)
```

- draw a default hline at `'y' = .5` that spans the middle half of the xrange:

```
>>> axhline(y=.5, xmin=0.25, xmax=0.75)
```

Examples using `matplotlib.axes.Axes.axhline`

- `sphx_glr_gallery_color_color_cycle_default.py`

matplotlib.axes.Axes.axhspan

Axes.axhspan(ymin, ymax, xmin=0, xmax=1, **kwargs)

Add a horizontal span (rectangle) across the axis.

Draw a horizontal span (rectangle) from ymin to ymax. With the default values of xmin = 0 and xmax = 1, this always spans the xrange, regardless of the xlim settings, even if you change them, e.g., with the `set_xlim()` command. That is, the horizontal extent is in axes coords: 0=left, 0.5=middle, 1.0=right but the y location is in data coordinates.

Parameters

ymin [float] Lower limit of the horizontal span in data units.

ymax [float] Upper limit of the horizontal span in data units.

xmin [float, optional, default: 0] Lower limit of the vertical span in axes (relative 0-1) units.

xmax [float, optional, default: 1] Upper limit of the vertical span in axes (relative 0-1) units.

Returns

Polygon [Polygon]

Other Parameters

**kwargs [Polygon properties.]

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

See also:

`axvspan` Add a vertical span across the axes.

`matplotlib.axes.Axes.axvline`

`Axes.axvline(x=0, ymin=0, ymax=1, **kwargs)`

Add a vertical line across the axes.

Parameters

`x` [scalar, optional, default: 0] `x` position in data coordinates of the vertical line.

`ymin` [scalar, optional, default: 0] Should be between 0 and 1, 0 being the bottom of the plot, 1 the top of the plot.

`ymax` [scalar, optional, default: 1] Should be between 0 and 1, 0 being the bottom of the plot, 1 the top of the plot.

Returns

`line` [Line2D]

Other Parameters

`**kwargs` : Valid kwargs are Line2D properties, with the exception of 'transform':

	Property	Description
<code>agg_filter</code>		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)
<code>alpha</code>		float (0.0 transparent through 1.0 opaque)
<code>animated</code>		bool
<code>antialiased</code> or <code>aa</code>		bool
<code>clip_box</code>		a Bbox instance
<code>clip_on</code>		bool
<code>clip_path</code>		[(Path, Transform) Patch None]
<code>color</code> or <code>c</code>		any matplotlib color
<code>contains</code>		a callable function
<code>dash_capstyle</code>		['butt' 'round' 'projecting']
<code>dash_joinstyle</code>		['miter' 'round' 'bevel']
<code>dashes</code>		sequence of on/off ink in points
<code>drawstyle</code>		['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']
<code>figure</code>		a Figure instance
<code>fillstyle</code>		['full' 'left' 'right' 'bottom' 'top' 'none']
<code>gid</code>		an id string
<code>label</code>		object
<code>linestyle</code> or <code>ls</code>		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']
<code>linewidth</code> or <code>lw</code>		float value in points
<code>marker</code>		A valid marker style
<code>markeredgecolor</code> or <code>mec</code>		any matplotlib color
<code>markeredgewidth</code> or <code>mew</code>		float value in points
<code>markerfacecolor</code> or <code>mfc</code>		any matplotlib color
<code>markerfacecoloralt</code> or <code>mfcalt</code>		any matplotlib color

Continued on next page

Table 10 – continued from previous page

	Property	Description
markersize or ms	float	
markevery	[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]	
path_effects	AbstractPathEffect	
picker	float distance in points or callable pick function fn(artist, event)	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
solid_capstyle	['butt' 'round' 'projecting']	
solid_joinstyle	['miter' 'round' 'bevel']	
transform	a matplotlib.transforms.Transform instance	
url	a url string	
visible	bool	
xdata	1D array	
ydata	1D array	
zorder	float	

See also:

`vlines` Add vertical lines in data coordinates.

`axvspan` Add a vertical span (rectangle) across the axis.

Examples

- draw a thick red vline at $x = 0$ that spans the yrange:

```
>>> axvline(linewidth=4, color='r')
```

- draw a default vline at $x = 1$ that spans the yrange:

```
>>> axvline(x=1)
```

- draw a default vline at $x = .5$ that spans the middle half of the yrange:

```
>>> axvline(x=.5, ymin=0.25, ymax=0.75)
```

Examples using `matplotlib.axes.Axes.axvline`

- `sphx_glr_gallery_color_color_cycle_default.py`

`matplotlib.axes.Axes.axvspan`

`Axes.axvspan(xmin, xmax, ymin=0, ymax=1, **kwargs)`

Add a vertical span (rectangle) across the axes.

Draw a vertical span (rectangle) from `xmin` to `xmax`. With the default values of `ymin = 0` and `ymax = 1`. This always spans the yrange, regardless of the `ylim` settings, even if you change them, e.g., with the `set_ylim()` command. That is, the vertical extent is in axes coords: 0=bottom, 0.5=middle, 1.0=top but the `y` location is in data coordinates.

Parameters

`xmin` [scalar] Number indicating the first X-axis coordinate of the vertical span rectangle in data units.

`xmax` [scalar] Number indicating the second X-axis coordinate of the vertical span rectangle in data units.

`ymin` [scalar, optional] Number indicating the first Y-axis coordinate of the vertical span rectangle in relative Y-axis units (0-1). Default to 0.

`ymax` [scalar, optional] Number indicating the second Y-axis coordinate of the vertical span rectangle in relative Y-axis units (0-1). Default to 1.

Returns

rectangle [matplotlib.patches.Polygon] Vertical span (rectangle) from (`xmin`, `ymin`) to (`xmax`, `ymax`).

Other Parameters

`**kwargs` Optional parameters are properties of the class `matplotlib.patches.Polygon`.

See also:

`axhspan` Add a horizontal span across the axes.

Examples

Draw a vertical, green, translucent rectangle from `x = 1.25` to `x = 1.55` that spans the yrange of the axes.

```
>>> axvspan(1.25, 1.55, facecolor='g', alpha=0.5)
```

35.1.3 Spectral

<code>Axes.acorr</code>	Plot the autocorrelation of <code>x</code> .
<code>Axes.angle_spectrum</code>	Plot the angle spectrum.
<code>Axes.cohere</code>	Plot the coherence between <code>x</code> and <code>y</code> .

Continued on next page

Table 11 – continued from previous page

<code>Axes.csd</code>	Plot the cross-spectral density.
<code>Axes.magnitude_spectrum</code>	Plot the magnitude spectrum.
<code>Axes.phase_spectrum</code>	Plot the phase spectrum.
<code>Axes.psd</code>	Plot the power spectral density.
<code>Axes.spectrogram</code>	Plot a spectrogram.
<code>Axes.xcorr</code>	Plot the cross correlation between x and y.

`matplotlib.axes.Axes.acorr`

`Axes.acorr(x, *, data=None, **kwargs)`

Plot the autocorrelation of x.

Parameters

`x` [sequence of scalar]

`hold` [bool, optional, deprecated, default: True]

`detrend` [callable, optional, default: `mlab.detrend_none`] x is detrended by the detrend callable. Default is no normalization.

`normed` [bool, optional, default: True] If True, input vectors are normalised to unit length.

`usevlines` [bool, optional, default: True] If True, `Axes.vlines` is used to plot the vertical lines from the origin to the `acorr`. Otherwise, `Axes.plot` is used.

`maxlags` [integer, optional, default: 10] Number of lags to show. If None, will return all $2 * \text{len}(x) - 1$ lags.

Returns

`lags` [array (length $2 * \text{maxlags} + 1$)] lag vector.

`c` [array (length $2 * \text{maxlags} + 1$)] auto correlation vector.

`line` [LineCollection or Line2D] Artist added to the axes of the correlation.

LineCollection if `usevlines` is True Line2D if `usevlines` is False

`b` [Line2D or None] Horizontal line at 0 if `usevlines` is True None `usevlines` is False

Other Parameters

`linestyle` [Line2D prop, optional, default: None] Only used if `usevlines` is False.

`marker` [string, optional, default: 'o']

Notes

The cross correlation is performed with `numpy.correlate()` with `mode = 2`.

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All arguments with the following names: 'x'.
-

matplotlib.axes.Axes.angle_spectrum

```
Axes.angle_spectrum(x, Fs=None, Fc=None, window=None, pad_to=None, sides=None, *,
                    data=None, **kwargs)
```

Plot the angle spectrum.

Call signature:

```
angle_spectrum(x, Fs=2, Fc=0, window=mlab.window_hanning,
              pad_to=None, sides='default', **kwargs)
```

Compute the angle spectrum (wrapped phase spectrum) of x. Data is padded to a length of pad_to and the windowing function window is applied to the signal.

Parameters

- x [1-D array or sequence] Array or sequence containing the data
- Fs [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, freqs, in cycles per time unit. The default value is 2.
- window [callable or ndarray] A function or a vector of length NFFT. To create window vectors see window_hanning(), window_none(), [numpy.blackman\(\)](#), [numpy.hamming\(\)](#), [numpy.bartlett\(\)](#), [scipy.signal\(\)](#), [scipy.signal.get_window\(\)](#), etc. The default is window_hanning(). If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.
- sides [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.
- pad_to [integer] The number of points to which the data segment is padded when performing the FFT. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the n parameter in the call to [fft\(\)](#). The default is None, which sets pad_to equal to the length of the input signal (i.e. no padding).
- Fc [integer] The center frequency of x (defaults to 0), which offsets the x extents of the plot to reflect the frequency range used when a signal is acquired and then filtered and downsampled to baseband.

Returns

- spectrum [1-D array] The values for the angle spectrum in radians (real valued)
- freqs [1-D array] The frequencies corresponding to the elements in spectrum
- line [a Line2D instance] The line created by this function

Other Parameters

****kwargs** : Keyword arguments control the Line2D properties:

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) float array
alpha		float (0.0 transparent through 1.0 opaque)
animated		bool
antialiased or aa		bool
clip_box		a Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
color or c		any matplotlib color
contains		a callable function
dash_capstyle		['butt' 'round' 'projecting']
dash_joinstyle		['miter' 'round' 'bevel']
dashes		sequence of on/off ink in points
drawstyle		['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']
figure		a Figure instance
fillstyle		['full' 'left' 'right' 'bottom' 'top' 'none']
gid		an id string
label		object
linestyle or ls		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']
linewidth or lw		float value in points
marker		A valid marker style
markeredgecolor or mec		any matplotlib color
markeredgewidth or mew		float value in points
markerfacecolor or mfc		any matplotlib color
markerfacecoloralt or mfcalt		any matplotlib color
markersize or ms		float
markevery		[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]
path_effects		AbstractPathEffect
picker		float distance in points or callable pick function fn(artist, event)
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
solid_capstyle		['butt' 'round' 'projecting']
solid_joinstyle		['miter' 'round' 'bevel']

Continued on next page

Table 12 – continued from previous page

	Property	Description
transform		a matplotlib.transforms.Transform instance
url		a url string
visible		bool
xdata		1D array
ydata		1D array
zorder		float

See also:

`magnitude_spectrum()` `angle_spectrum()` plots the magnitudes of the corresponding frequencies.

`phase_spectrum()` `phase_spectrum()` plots the unwrapped version of this function.

`specgram()` `specgram()` can plot the angle spectrum of segments within the signal in a colormap.

Notes

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'x'.
-

matplotlib.axes.Axes.cohere

`Axes.cohere(x, y, NFFT=256, Fs=2, Fc=0, detrend=<function detrend_none>, window=<function window_hanning>, noverlap=0, pad_to=None, sides='default', scale_by_freq=None, *, data=None, **kwargs)`

Plot the coherence between x and y.

Plot the coherence between x and y. Coherence is the normalized cross spectral density:

$$C_{xy} = \frac{|P_{xy}|^2}{P_{xx}P_{yy}}$$

Parameters

`Fs` [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.

`window` [callable or ndarray] A function or a vector of length `NFFT`. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function

is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.

sides `[['default' | 'onesided' | 'twosided']]` Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.

pad_to `[integer]` The number of points to which the data segment is padded when performing the FFT. This can be different from NFFT, which specifies the number of data points used. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` equal to `NFFT`

NFFT `[integer]` The number of data points used in each block for the FFT. A power 2 is most efficient. The default value is 256. This should NOT be used to get zero padding, or the scaling of the result will be incorrect. Use `pad_to` for this instead.

detrend `[{'default', 'constant', 'mean', 'linear', 'none'} or callable]` The function applied to each segment before `fft`-ing, designed to remove the mean or linear trend. Unlike in MATLAB, where the `detrend` parameter is a vector, in matplotlib is it a function. The `pylab` module defines `detrend_none()`, `detrend_mean()`, and `detrend_linear()`, but you can use a custom function as well. You can also use a string to choose one of the functions. 'default', 'constant', and 'mean' call `detrend_mean()`. 'linear' calls `detrend_linear()`. 'none' calls `detrend_none()`.

scale_by_freq `[boolean, optional]` Specifies whether the resulting density values should be scaled by the scaling frequency, which gives density in units of Hz^{-1} . This allows for integration over the returned frequency values. The default is `True` for MATLAB compatibility.

noverlap `[integer]` The number of points of overlap between blocks. The default value is 0 (no overlap).

Fc `[integer]` The center frequency of `x` (defaults to 0), which offsets the `x` extents of the plot to reflect the frequency range used when a signal is acquired and then filtered and downsampled to baseband.

Returns

The return value is a tuple `(*Cxy*, *f*)`, where `*f*` are the frequencies of the coherence vector.

`kwargs` are applied to the lines.

Other Parameters

****kwargs** : Keyword arguments control the `Line2D` properties:

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) float array
alpha		float (0.0 transparent through 1.0 opaque)
animated		bool
antialiased or aa		bool
clip_box		a Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
color or c		any matplotlib color
contains		a callable function
dash_capstyle		['butt' 'round' 'projecting']
dash_joinstyle		['miter' 'round' 'bevel']
dashes		sequence of on/off ink in points
drawstyle		['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']
figure		a Figure instance
fillstyle		['full' 'left' 'right' 'bottom' 'top' 'none']
gid		an id string
label		object
linestyle or ls		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']
linewidth or lw		float value in points
marker		A valid marker style
markeredgecolor or mec		any matplotlib color
markeredgewidth or mew		float value in points
markerfacecolor or mfc		any matplotlib color
markerfacecoloralt or mfcalt		any matplotlib color
markersize or ms		float
markevery		[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]
path_effects		AbstractPathEffect
picker		float distance in points or callable pick function fn(artist, event)
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
solid_capstyle		['butt' 'round' 'projecting']
solid_joinstyle		['miter' 'round' 'bevel']
transform		a matplotlib.transforms.Transform instance
url		a url string
visible		bool
xdata		1D array
ydata		1D array
zorder		float

References

Bendat & Piersol – Random Data: Analysis and Measurement Procedures, John Wiley & Sons (1986)

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'x', 'y'.
-

matplotlib.axes.Axes.csd

`Axes.csd(x, y, NFFT=None, Fs=None, Fc=None, detrend=None, window=None, noverlap=None, pad_to=None, sides=None, scale_by_freq=None, return_line=None, *, data=None, **kwargs)`
Plot the cross-spectral density.

Call signature:

```
csd(x, y, NFFT=256, Fs=2, Fc=0, detrend=mlab.detrend_none,
    window=mlab.window_hanning, noverlap=0, pad_to=None,
    sides='default', scale_by_freq=None, return_line=None, **kwargs)
```

The cross spectral density P_{xy} by Welch's average periodogram method. The vectors `x` and `y` are divided into `NFFT` length segments. Each segment is detrended by function `detrend` and windowed by function `window`. `noverlap` gives the length of the overlap between segments. The product of the direct FFTs of `x` and `y` are averaged over each segment to compute P_{xy} , with a scaling to correct for power loss due to windowing.

If `len(x) < NFFT` or `len(y) < NFFT`, they will be zero padded to `NFFT`.

Parameters

`x, y` [1-D arrays or sequences] Arrays or sequences containing the data

`Fs` [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.

`window` [callable or ndarray] A function or a vector of length `NFFT`. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.

`sides` [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for

real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.

`pad_to` [integer] The number of points to which the data segment is padded when performing the FFT. This can be different from `NFFT`, which specifies the number of data points used. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` equal to `NFFT`

`NFFT` [integer] The number of data points used in each block for the FFT. A power 2 is most efficient. The default value is 256. This should NOT be used to get zero padding, or the scaling of the result will be incorrect. Use `pad_to` for this instead.

`detrend` [{'default', 'constant', 'mean', 'linear', 'none'} or callable] The function applied to each segment before `fft`-ing, designed to remove the mean or linear trend. Unlike in MATLAB, where the `detrend` parameter is a vector, in matplotlib is it a function. The `pylab` module defines `detrend_none()`, `detrend_mean()`, and `detrend_linear()`, but you can use a custom function as well. You can also use a string to choose one of the functions. 'default', 'constant', and 'mean' call `detrend_mean()`. 'linear' calls `detrend_linear()`. 'none' calls `detrend_none()`.

`scale_by_freq` [boolean, optional] Specifies whether the resulting density values should be scaled by the scaling frequency, which gives density in units of Hz^{-1} . This allows for integration over the returned frequency values. The default is `True` for MATLAB compatibility.

`noverlap` [integer] The number of points of overlap between segments. The default value is 0 (no overlap).

`Fc` [integer] The center frequency of `x` (defaults to 0), which offsets the `x` extents of the plot to reflect the frequency range used when a signal is acquired and then filtered and downsampled to baseband.

`return_line` [bool] Whether to include the line object plotted in the returned values. Default is `False`.

Returns

`Pxy` [1-D array] The values for the cross spectrum $P_{\{xy\}}$ before scaling (complex valued)

`freqs` [1-D array] The frequencies corresponding to the elements in `Pxy`

`line` [a `Line2D` instance] The line created by this function. Only returned if `return_line` is `True`.

Other Parameters

****kwargs** : Keyword arguments control the `Line2D` properties:

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) float array
alpha		float (0.0 transparent through 1.0 opaque)
animated		bool
antialiased or aa		bool
clip_box		a Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
color or c		any matplotlib color
contains		a callable function
dash_capstyle		['butt' 'round' 'projecting']
dash_joinstyle		['miter' 'round' 'bevel']
dashes		sequence of on/off ink in points
drawstyle		['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']
figure		a Figure instance
fillstyle		['full' 'left' 'right' 'bottom' 'top' 'none']
gid		an id string
label		object
linestyle or ls		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']
linewidth or lw		float value in points
marker		A valid marker style
markeredgecolor or mec		any matplotlib color
markeredgewidth or mew		float value in points
markerfacecolor or mfc		any matplotlib color
markerfacecoloralt or mfcalt		any matplotlib color
markersize or ms		float
markevery		[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]
path_effects		AbstractPathEffect
picker		float distance in points or callable pick function fn(artist, event)
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
solid_capstyle		['butt' 'round' 'projecting']
solid_joinstyle		['miter' 'round' 'bevel']
transform		a matplotlib.transforms.Transform instance
url		a url string
visible		bool
xdata		1D array
ydata		1D array
zorder		float

See also:

`psd()` `psd()` is the equivalent to setting `y=x`.

Notes

For plotting, the power is plotted as $10 \log_{10}(P_{xy})$ for decibels, though $P_{\{xy\}}$ itself is returned.

References

Bendat & Piersol – Random Data: Analysis and Measurement Procedures, John Wiley & Sons (1986)

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'x', 'y'.
-

matplotlib.axes.Axes.magnitude_spectrum

`Axes.magnitude_spectrum(x, Fs=None, Fc=None, window=None, pad_to=None, sides=None, scale=None, *, data=None, **kwargs)`

Plot the magnitude spectrum.

Call signature:

```

magnitude_spectrum(x, Fs=2, Fc=0, window=mlab.window_hanning,
                  pad_to=None, sides='default', **kwargs)

```

Compute the magnitude spectrum of x. Data is padded to a length of `pad_to` and the windowing function `window` is applied to the signal.

Parameters

- `x` [1-D array or sequence] Array or sequence containing the data
- `Fs` [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.
- `window` [callable or ndarray] A function or a vector of length `NFFT`. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.
- `sides` [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.

`pad_to` [integer] The number of points to which the data segment is padded when performing the FFT. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` equal to the length of the input signal (i.e. no padding).

`scale` [['default' | 'linear' | 'dB']] The scaling of the values in the spec. 'linear' is no scaling. 'dB' returns the values in dB scale, i.e., the dB amplitude ($20 * \log_{10}$). 'default' is 'linear'.

`Fc` [integer] The center frequency of `x` (defaults to 0), which offsets the `x` extents of the plot to reflect the frequency range used when a signal is acquired and then filtered and downsampled to baseband.

Returns

`spectrum` [1-D array] The values for the magnitude spectrum before scaling (real valued)

`freqs` [1-D array] The frequencies corresponding to the elements in `spectrum`

`line` [a Line2D instance] The line created by this function

Other Parameters

****kwargs** : Keyword arguments control the Line2D properties:

	Property	Description
<code>agg_filter</code>		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)
<code>alpha</code>		float (0.0 transparent through 1.0 opaque)
<code>animated</code>		bool
<code>antialiased</code> or <code>aa</code>		bool
<code>clip_box</code>		a Bbox instance
<code>clip_on</code>		bool
<code>clip_path</code>		[(Path, Transform) Patch None]
<code>color</code> or <code>c</code>		any matplotlib color
<code>contains</code>		a callable function
<code>dash_capstyle</code>		['butt' 'round' 'projecting']
<code>dash_joinstyle</code>		['miter' 'round' 'bevel']
<code>dashes</code>		sequence of on/off ink in points
<code>drawstyle</code>		['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']
<code>figure</code>		a Figure instance
<code>fillstyle</code>		['full' 'left' 'right' 'bottom' 'top' 'none']
<code>gid</code>		an id string
<code>label</code>		object
<code>linestyle</code> or <code>ls</code>		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']
<code>linewidth</code> or <code>lw</code>		float value in points
<code>marker</code>		A valid marker style
<code>markeredgecolor</code> or <code>mec</code>		any matplotlib color

Continued on next page

Table 15 – continued from previous page

	Property	Description
markeredgewidth or mew	float	value in points
markerfacecolor or mfc	any matplotlib color	
markerfacecoloralt or mfcalt	any matplotlib color	
markersize or ms	float	
markevery	[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]	
path_effects	AbstractPathEffect	
picker	float	distance in points or callable pick function <code>fn(artist, event)</code>
pickradius	float	distance in points
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
solid_capstyle	['butt' 'round' 'projecting']	
solid_joinstyle	['miter' 'round' 'bevel']	
transform	a matplotlib.transforms.Transform instance	
url	a url string	
visible	bool	
xdata	1D array	
ydata	1D array	
zorder	float	

See also:

`psd()` `psd()` plots the power spectral density.

`angle_spectrum()` `angle_spectrum()` plots the angles of the corresponding frequencies.

`phase_spectrum()` `phase_spectrum()` plots the phase (unwrapped angle) of the corresponding frequencies.

`specgram()` `specgram()` can plot the magnitude spectrum of segments within the signal in a colormap.

Notes

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'x'.
-

`matplotlib.axes.Axes.phase_spectrum`

```
Axes.phase_spectrum(x, Fs=None, Fc=None, window=None, pad_to=None, sides=None, *,
                    data=None, **kwargs)
```

Plot the phase spectrum.

Call signature:

```
phase_spectrum(x, Fs=2, Fc=0, window=mlab.window_hanning,
               pad_to=None, sides='default', **kwargs)
```

Compute the phase spectrum (unwrapped angle spectrum) of `x`. Data is padded to a length of `pad_to` and the windowing function `window` is applied to the signal.

Parameters

- `x` [1-D array or sequence] Array or sequence containing the data
- `Fs` [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.
- `window` [callable or ndarray] A function or a vector of length `NFFT`. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.
- `sides` [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.
- `pad_to` [integer] The number of points to which the data segment is padded when performing the FFT. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` equal to the length of the input signal (i.e. no padding).
- `Fc` [integer] The center frequency of `x` (defaults to 0), which offsets the `x` extents of the plot to reflect the frequency range used when a signal is acquired and then filtered and downsampled to baseband.

Returns

- `spectrum` [1-D array] The values for the phase spectrum in radians (real valued)
- `freqs` [1-D array] The frequencies corresponding to the elements in `spectrum`
- `line` [a Line2D instance] The line created by this function

Other Parameters

****kwargs** : Keyword arguments control the Line2D properties:

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)
alpha		float (0.0 transparent through 1.0 opaque)
animated		bool
antialiased or aa		bool
clip_box		a Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
color or c		any matplotlib color
contains		a callable function
dash_capstyle		['butt' 'round' 'projecting']
dash_joinstyle		['miter' 'round' 'bevel']
dashes		sequence of on/off ink in points
drawstyle		['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']
figure		a Figure instance
fillstyle		['full' 'left' 'right' 'bottom' 'top' 'none']
gid		an id string
label		object
linestyle or ls		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']
linewidth or lw		float value in points
marker		A valid marker style
markeredgecolor or mec		any matplotlib color
markeredgewidth or mew		float value in points
markerfacecolor or mfc		any matplotlib color
markerfacecoloralt or mfcalt		any matplotlib color
markersize or ms		float
markevery		[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]
path_effects		AbstractPathEffect
picker		float distance in points or callable pick function fn(artist, event)
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
solid_capstyle		['butt' 'round' 'projecting']
solid_joinstyle		['miter' 'round' 'bevel']
transform		a matplotlib.transforms.Transform instance
url		a url string
visible		bool
xdata		1D array
ydata		1D array
zorder		float

See also:

`magnitude_spectrum()` `magnitude_spectrum()` plots the magnitudes of the corresponding frequencies.

`angle_spectrum()` `angle_spectrum()` plots the wrapped version of this function.

`specgram()` `specgram()` can plot the phase spectrum of segments within the signal in a colormap.

Notes

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'x'.
-

`matplotlib.axes.Axes.psd`

`Axes.psd(x, NFFT=None, Fs=None, Fc=None, detrend=None, window=None, noverlap=None, pad_to=None, sides=None, scale_by_freq=None, return_line=None, *, data=None, **kwargs)`
Plot the power spectral density.

Call signature:

```
psd(x, NFFT=256, Fs=2, Fc=0, detrend=mlab.detrend_none,
    window=mlab.window_hanning, noverlap=0, pad_to=None,
    sides='default', scale_by_freq=None, return_line=None, **kwargs)
```

The power spectral density P_{xx} by Welch's average periodogram method. The vector `x` is divided into `NFFT` length segments. Each segment is detrended by function `detrend` and windowed by function `window`. `noverlap` gives the length of the overlap between segments. The $|\text{fft}(i)|^2$ of each segment i are averaged to compute P_{xx} , with a scaling to correct for power loss due to windowing.

If $\text{len}(x) < \text{NFFT}$, it will be zero padded to `NFFT`.

Parameters

`x` [1-D array or sequence] Array or sequence containing the data

`Fs` [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.

window [callable or ndarray] A function or a vector of length NFFT. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.

sides [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.

pad_to [integer] The number of points to which the data segment is padded when performing the FFT. This can be different from NFFT, which specifies the number of data points used. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` equal to NFFT

NFFT [integer] The number of data points used in each block for the FFT. A power 2 is most efficient. The default value is 256. This should NOT be used to get zero padding, or the scaling of the result will be incorrect. Use `pad_to` for this instead.

detrend [{'default', 'constant', 'mean', 'linear', 'none'} or callable] The function applied to each segment before fft-ing, designed to remove the mean or linear trend. Unlike in MATLAB, where the `detrend` parameter is a vector, in matplotlib it is a function. The `pylab` module defines `detrend_none()`, `detrend_mean()`, and `detrend_linear()`, but you can use a custom function as well. You can also use a string to choose one of the functions. 'default', 'constant', and 'mean' call `detrend_mean()`. 'linear' calls `detrend_linear()`. 'none' calls `detrend_none()`.

scale_by_freq [boolean, optional] Specifies whether the resulting density values should be scaled by the scaling frequency, which gives density in units of Hz^{-1} . This allows for integration over the returned frequency values. The default is `True` for MATLAB compatibility.

noverlap [integer] The number of points of overlap between segments. The default value is 0 (no overlap).

Fc [integer] The center frequency of `x` (defaults to 0), which offsets the `x` extents of the plot to reflect the frequency range used when a signal is acquired and then filtered and downsampled to baseband.

return_line [bool] Whether to include the line object plotted in the returned values. Default is `False`.

Returns

Pxx [1-D array] The values for the power spectrum $P_{\{xx\}}$ before scaling (real valued)

freqs [1-D array] The frequencies corresponding to the elements in Pxx

line [a Line2D instance] The line created by this function. Only returned if return_line is True.

Other Parameters

****kwargs** : Keyword arguments control the Line2D properties:

	Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)	
alpha	float (0.0 transparent through 1.0 opaque)	
animated	bool	
antialiased or aa	bool	
clip_box	a Bbox instance	
clip_on	bool	
clip_path	[(Path, Transform) Patch None]	
color or c	any matplotlib color	
contains	a callable function	
dash_capstyle	['butt' 'round' 'projecting']	
dash_joinstyle	['miter' 'round' 'bevel']	
dashes	sequence of on/off ink in points	
drawstyle	['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']	
figure	a Figure instance	
fillstyle	['full' 'left' 'right' 'bottom' 'top' 'none']	
gid	an id string	
label	object	
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']	
linewidth or lw	float value in points	
marker	A valid marker style	
markeredgecolor or mec	any matplotlib color	
markeredgewidth or mew	float value in points	
markerfacecolor or mfc	any matplotlib color	
markerfacecoloralt or mfcalt	any matplotlib color	
markersize or ms	float	
markevery	[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]	
path_effects	AbstractPathEffect	
picker	float distance in points or callable pick function fn(artist, event)	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
solid_capstyle	['butt' 'round' 'projecting']	
solid_joinstyle	['miter' 'round' 'bevel']	
transform	a matplotlib.transforms.Transform instance	
url	a url string	
visible	bool	

Continued on next page

Table 17 – continued from previous page

	Property	Description
xdata	1D array	
ydata	1D array	
zorder	float	

See also:

`specgram()` `specgram()` differs in the default overlap; in not returning the mean of the segment periodograms; in returning the times of the segments; and in plotting a colormap instead of a line.

`magnitude_spectrum()` `magnitude_spectrum()` plots the magnitude spectrum.

`csd()` `csd()` plots the spectral density between two signals.

Notes

For plotting, the power is plotted as $10 \log_{10}(P_{xx})$ for decibels, though P_{xx} itself is returned.

References

Bendat & Piersol – Random Data: Analysis and Measurement Procedures, John Wiley & Sons (1986)

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'x'.
-

matplotlib.axes.Axes.specgram

```
Axes.specgram(x, NFFT=None, Fs=None, Fc=None, detrend=None, window=None,
               noverlap=None, cmap=None, xextent=None, pad_to=None, sides=None,
               scale_by_freq=None, mode=None, scale=None, vmin=None, vmax=None,
               *, data=None, **kwargs)
```

Plot a spectrogram.

Call signature:

```
specgram(x, NFFT=256, Fs=2, Fc=0, detrend=mlab.detrend_none,
         window=mlab.window_hanning, noverlap=128,
         cmap=None, xextent=None, pad_to=None, sides='default',
         scale_by_freq=None, mode='default', scale='default',
         **kwargs)
```

Compute and plot a spectrogram of data in `x`. Data are split into NFFT length segments and the spectrum of each section is computed. The windowing function `window` is applied to each segment, and the amount of overlap of each segment is specified with `noverlap`. The spectrogram is plotted as a colormap (using `imshow`).

Parameters

- `x` [1-D array or sequence] Array or sequence containing the data.
- `Fs` [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.
- `window` [callable or ndarray] A function or a vector of length NFFT. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.
- `sides` [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.
- `pad_to` [integer] The number of points to which the data segment is padded when performing the FFT. This can be different from NFFT, which specifies the number of data points used. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` equal to NFFT
- `NFFT` [integer] The number of data points used in each block for the FFT. A power 2 is most efficient. The default value is 256. This should NOT be used to get zero padding, or the scaling of the result will be incorrect. Use `pad_to` for this instead.
- `detrend` [{'default', 'constant', 'mean', 'linear', 'none'} or callable] The function applied to each segment before `fft`-ing, designed to remove the mean or linear trend. Unlike in MATLAB, where the `detrend` parameter is a vector, in matplotlib is it a function. The `pylab` module defines `detrend_none()`, `detrend_mean()`, and `detrend_linear()`, but you can use a custom function as well. You can also use a string to choose one of the functions. 'default', 'constant', and 'mean' call `detrend_mean()`. 'linear' calls `detrend_linear()`. 'none' calls `detrend_none()`.
- `scale_by_freq` [boolean, optional] Specifies whether the resulting density values should be scaled by the scaling frequency, which gives density in units of Hz^{-1} . This allows for integration over the returned frequency values. The default is `True` for MATLAB compatibility.

`mode` `[['default' | 'psd' | 'magnitude' | 'angle' | 'phase']]` What sort of spectrum to use. Default is 'psd', which takes the power spectral density. 'complex' returns the complex-valued frequency spectrum. 'magnitude' returns the magnitude spectrum. 'angle' returns the phase spectrum without unwrapping. 'phase' returns the phase spectrum with unwrapping.

`noverlap` `[integer]` The number of points of overlap between blocks. The default value is 128.

`scale` `[['default' | 'linear' | 'dB']]` The scaling of the values in the spec. 'linear' is no scaling. 'dB' returns the values in dB scale. When mode is 'psd', this is dB power ($10 * \log_{10}$). Otherwise this is dB amplitude ($20 * \log_{10}$). 'default' is 'dB' if mode is 'psd' or 'magnitude' and 'linear' otherwise. This must be 'linear' if mode is 'angle' or 'phase'.

`Fc` `[integer]` The center frequency of `x` (defaults to 0), which offsets the `x` extents of the plot to reflect the frequency range used when a signal is acquired and then filtered and downsampled to baseband.

`cmap` : A `matplotlib.colors.Colormap` instance; if `None`, use default determined by `rc`

`xextent` `[[None | (xmin, xmax)]]` The image extent along the `x`-axis. The default sets `xmin` to the left border of the first bin (spectrum column) and `xmax` to the right border of the last bin. Note that for `noverlap>0` the width of the bins is smaller than those of the segments.

****kwargs** : Additional kwargs are passed on to `imshow` which makes the spectrogram image

Returns

`spectrum` `[2-D array]` Columns are the periodograms of successive segments.

`freqs` `[1-D array]` The frequencies corresponding to the rows in `spectrum`.

`t` `[1-D array]` The times corresponding to midpoints of segments (i.e., the columns in `spectrum`).

`im` `[instance of class AxesImage]` The image created by `imshow` containing the spectrogram

See also:

`psd()` `psd()` differs in the default overlap; in returning the mean of the segment periodograms; in not returning times; and in generating a line plot instead of colormap.

`magnitude_spectrum()` A single spectrum, similar to having a single segment when mode is 'magnitude'. Plots a line instead of a colormap.

`angle_spectrum()` A single spectrum, similar to having a single segment when mode is 'angle'. Plots a line instead of a colormap.

`phase_spectrum()` A single spectrum, similar to having a single segment when mode is 'phase'. Plots a line instead of a colormap.

Notes

The parameters `detrend` and `scale_by_freq` do only apply when mode is set to `'psd'`.

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: `'x'`.
-

Examples using `matplotlib.axes.Axes.specgram`

- `sphx_glr_gallery_images_contours_and_fields_specgram_demo.py`

`matplotlib.axes.Axes.xcorr`

`Axes.xcorr(x, y, normed=True, detrend=<function detrend_none>, usevlines=True, maxlags=10, *, data=None, **kwargs)`

Plot the cross correlation between `x` and `y`.

The correlation with lag `k` is defined as $\sum_n x[n+k] * \text{conj}(y[n])$.

Parameters

`x` [sequence of scalars of length `n`]

`y` [sequence of scalars of length `n`]

`hold` [bool, optional, deprecated, default: `True`]

`detrend` [callable, optional, default: `mlab.detrend_none`] `x` is detrended by the `detrend` callable. Default is no normalization.

`normed` [bool, optional, default: `True`] If `True`, input vectors are normalised to unit length.

`usevlines` [bool, optional, default: `True`] If `True`, `Axes.vlines` is used to plot the vertical lines from the origin to the `acorr`. Otherwise, `Axes.plot` is used.

`maxlags` [int, optional] Number of lags to show. If `None`, will return all $2 * \text{len}(x) - 1$ lags. Default is 10.

Returns

`lags` [array (length $2 * \text{maxlags} + 1$)] lag vector.

`c` [array (length $2 * \text{maxlags} + 1$)] auto correlation vector.

`line` [LineCollection or Line2D] Artist added to the axes of the correlation
LineCollection if `usevlines` is `True` Line2D if `usevlines` is `False`

- b [Line2D or None] Horizontal line at 0 if usevlines is True None usevlines is False

Other Parameters

- linestyle [Line2D property, optional] Only used if usevlines is False.
- marker [string, optional] Default is 'o'.

Notes

The cross correlation is performed with `numpy.correlate()` with `mode = 2`.

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'x', 'y'.
-

35.1.4 Statistics

<code>Axes.boxplot</code>	Make a box and whisker plot.
<code>Axes.violinplot</code>	Make a violin plot.
<code>Axes.violin</code>	Drawing function for violin plots.
<code>Axes.bxp</code>	Drawing function for box and whisker plots.

`matplotlib.axes.Axes.boxplot`

`Axes.boxplot(x, notch=None, sym=None, vert=None, whis=None, positions=None, widths=None, patch_artist=None, bootstrap=None, usermedians=None, conf_intervals=None, meanline=None, showmeans=None, showcaps=None, showbox=None, showfliers=None, boxprops=None, labels=None, flierprops=None, medianprops=None, meanprops=None, capprops=None, whiskerprops=None, manage_xticks=True, autorange=False, zorder=None, *, data=None)`

Make a box and whisker plot.

Make a box and whisker plot for each column of `x` or each vector in sequence `x`. The box extends from the lower to upper quartile values of the data, with a line at the median. The whiskers extend from the box to show the range of the data. Flier points are those past the end of the whiskers.

Parameters

- x [Array or a sequence of vectors.] The input data.
- notch [bool, optional (False)] If `True`, will produce a notched box plot. Otherwise, a rectangular boxplot is produced. The notches represent the con-

fidence interval (CI) around the median. See the entry for the bootstrap parameter for information regarding how the locations of the notches are computed.

Note: In cases where the values of the CI are less than the lower quartile or greater than the upper quartile, the notches will extend beyond the box, giving it a distinctive "flipped" appearance. This is expected behavior and consistent with other statistical visualization packages.

`sym` [str, optional] The default symbol for flier points. Enter an empty string (") if you don't want to show fliers. If `None`, then the fliers default to 'b+'. If you want more control use the `flierprops` kwarg.

`vert` [bool, optional (True)] If `True` (default), makes the boxes vertical. If `False`, everything is drawn horizontally.

`whis` [float, sequence, or string (default = 1.5)] As a float, determines the reach of the whiskers to the beyond the first and third quartiles. In other words, where IQR is the interquartile range ($Q3 - Q1$), the upper whisker will extend to last datum less than $Q3 + whis * IQR$. Similarly, the lower whisker will extend to the first datum greater than $Q1 - whis * IQR$. Beyond the whiskers, data are considered outliers and are plotted as individual points. Set this to an unreasonably high value to force the whiskers to show the min and max values. Alternatively, set this to an ascending sequence of percentile (e.g., [5, 95]) to set the whiskers at specific percentiles of the data. Finally, `whis` can be the string 'range' to force the whiskers to the min and max of the data.

`bootstrap` [int, optional] Specifies whether to bootstrap the confidence intervals around the median for notched boxplots. If `bootstrap` is `None`, no bootstrapping is performed, and notches are calculated using a Gaussian-based asymptotic approximation (see McGill, R., Tukey, J.W., and Larsen, W.A., 1978, and Kendall and Stuart, 1967). Otherwise, `bootstrap` specifies the number of times to bootstrap the median to determine its 95% confidence intervals. Values between 1000 and 10000 are recommended.

`usermedians` [array-like, optional] An array or sequence whose first dimension (or length) is compatible with `x`. This overrides the medians computed by matplotlib for each element of `usermedians` that is not `None`. When an element of `usermedians` is `None`, the median will be computed by matplotlib as normal.

`conf_intervals` [array-like, optional] Array or sequence whose first dimension (or length) is compatible with `x` and whose second dimension is 2. When the an element of `conf_intervals` is not `None`, the notch locations computed by matplotlib are overridden (provided `notch` is `True`). When an element of `conf_intervals` is `None`, the notches are computed by the method specified by the other kwargs (e.g., `bootstrap`).

`positions` [array-like, optional] Sets the positions of the boxes. The ticks and

limits are automatically set to match the positions. Defaults to `range(1, N+1)` where `N` is the number of boxes to be drawn.

`widths` [scalar or array-like] Sets the width of each box either with a scalar or a sequence. The default is 0.5, or $0.15 \times (\text{distance between extreme positions})$, if that is smaller.

`patch_artist` [bool, optional (False)] If `False` produces boxes with the `Line2D` artist. Otherwise, boxes and drawn with `Patch` artists.

`labels` [sequence, optional] Labels for each dataset. Length must be compatible with dimensions of `x`.

`manage_xticks` [bool, optional (True)] If the function should adjust the `xlim` and `xtick` locations.

`autorange` [bool, optional (False)] When `True` and the data are distributed such that the 25th and 75th percentiles are equal, `whis` is set to 'range' such that the whisker ends are at the minimum and maximum of the data.

`meanline` [bool, optional (False)] If `True` (and `showmeans` is `True`), will try to render the mean as a line spanning the full width of the box according to `meanprops` (see below). Not recommended if `shownotches` is also `True`. Otherwise, means will be shown as points.

`zorder` [scalar, optional (None)] Sets the `zorder` of the boxplot.

Returns

`result` [dict] A dictionary mapping each component of the boxplot to a list of the `matplotlib.lines.Line2D` instances created. That dictionary has the following keys (assuming vertical boxplots):

- `boxes`: the main body of the boxplot showing the quartiles and the median's confidence intervals if enabled.
- `medians`: horizontal lines at the median of each box.
- `whiskers`: the vertical lines extending to the most extreme, non-outlier data points.
- `caps`: the horizontal lines at the ends of the whiskers.
- `fliers`: points representing data that extend beyond the whiskers (fliers).
- `means`: points or lines representing the means.

Other Parameters

`showcaps` [bool, optional (True)] Show the caps on the ends of whiskers.

`showbox` [bool, optional (True)] Show the central box.

`showfliers` [bool, optional (True)] Show the outliers beyond the caps.

`showmeans` [bool, optional (False)] Show the arithmetic means.

`capprops` [dict, optional (None)] Specifies the style of the caps.

`boxprops` [dict, optional (None)] Specifies the style of the box.

`whiskerprops` [dict, optional (None)] Specifies the style of the whiskers.

`flierprops` [dict, optional (None)] Specifies the style of the fliers.

`medianprops` [dict, optional (None)] Specifies the style of the median.

`meanprops` [dict, optional (None)] Specifies the style of the mean.

Notes

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All positional and all keyword arguments.
-

`matplotlib.axes.Axes.violinplot`

`Axes.violinplot(dataset, positions=None, vert=True, widths=0.5, showmeans=False, showextrema=True, showmedians=False, points=100, bw_method=None, *, data=None)`

Make a violin plot.

Make a violin plot for each column of dataset or each vector in sequence dataset. Each filled area extends to represent the entire data range, with optional lines at the mean, the median, the minimum, and the maximum.

Parameters

`dataset` [Array or a sequence of vectors.] The input data.

`positions` [array-like, default = [1, 2, ..., n]] Sets the positions of the violins. The ticks and limits are automatically set to match the positions.

`vert` [bool, default = True.] If true, creates a vertical violin plot. Otherwise, creates a horizontal violin plot.

`widths` [array-like, default = 0.5] Either a scalar or a vector that sets the maximal width of each violin. The default is 0.5, which uses about half of the available horizontal space.

`showmeans` [bool, default = False] If `True`, will toggle rendering of the means.

`showextrema` [bool, default = True] If `True`, will toggle rendering of the extrema.

`showmedians` [bool, default = False] If `True`, will toggle rendering of the medians.

`points` [scalar, default = 100] Defines the number of points to evaluate each of the gaussian kernel density estimations at.

`bw_method` [str, scalar or callable, optional] The method used to calculate the estimator bandwidth. This can be 'scott', 'silverman', a scalar constant or a callable. If a scalar, this will be used directly as `kde.factor`. If a callable, it should take a `GaussianKDE` instance as its only parameter and return a scalar. If `None` (default), 'scott' is used.

Returns

`result` [dict] A dictionary mapping each component of the violinplot to a list of the corresponding collection instances created. The dictionary has the following keys:

- `bodies`: A list of the `matplotlib.collections.PolyCollection` instances containing the filled area of each violin.
- `cmeans`: A `matplotlib.collections.LineCollection` instance created to identify the mean values of each of the violin's distribution.
- `cmns`: A `matplotlib.collections.LineCollection` instance created to identify the bottom of each violin's distribution.
- `cmaxes`: A `matplotlib.collections.LineCollection` instance created to identify the top of each violin's distribution.
- `cbars`: A `matplotlib.collections.LineCollection` instance created to identify the centers of each violin's distribution.
- `cmedians`: A `matplotlib.collections.LineCollection` instance created to identify the median values of each of the violin's distribution.

Notes

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'dataset'.
-

`matplotlib.axes.Axes.violin`

`Axes.violin(vpstats, positions=None, vert=True, widths=0.5, showmeans=False, showextrema=True, showmedians=False)`

Drawing function for violin plots.

Draw a violin plot for each column of `vpstats`. Each filled area extends to represent the entire data range, with optional lines at the mean, the median, the minimum, and the maximum.

Parameters

`vpstats` [list of dicts] A list of dictionaries containing stats for each violin plot. Required keys are:

- `coords`: A list of scalars containing the coordinates that the violin's kernel density estimate were evaluated at.
- `vals`: A list of scalars containing the values of the kernel density estimate at each of the coordinates given in `coords`.
- `mean`: The mean value for this violin's dataset.
- `median`: The median value for this violin's dataset.
- `min`: The minimum value for this violin's dataset.
- `max`: The maximum value for this violin's dataset.

`positions` [array-like, default = [1, 2, ..., n]] Sets the positions of the violins. The ticks and limits are automatically set to match the positions.

`vert` [bool, default = True.] If true, plots the violins veritcally. Otherwise, plots the violins horizontally.

`widths` [array-like, default = 0.5] Either a scalar or a vector that sets the maximal width of each violin. The default is 0.5, which uses about half of the available horizontal space.

`showmeans` [bool, default = False] If true, will toggle rendering of the means.

`showextrema` [bool, default = True] If true, will toggle rendering of the extrema.

`showmedians` [bool, default = False] If true, will toggle rendering of the medians.

Returns

`result` [dict] A dictionary mapping each component of the violinplot to a list of the corresponding collection instances created. The dictionary has the following keys:

- `bodies`: A list of the `matplotlib.collections.PolyCollection` instances containing the filled area of each violin.
- `cmeans`: A `matplotlib.collections.LineCollection` instance created to identify the mean values of each of the violin's distribution.
- `cmins`: A `matplotlib.collections.LineCollection` instance created to identify the bottom of each violin's distribution.
- `cmaxes`: A `matplotlib.collections.LineCollection` instance created to identify the top of each violin's distribution.
- `cbars`: A `matplotlib.collections.LineCollection` instance created to identify the centers of each violin's distribution.

- `cmedians`: A `matplotlib.collections.LineCollection` instance created to identify the median values of each of the violin's distribution.

`matplotlib.axes.Axes.bxp`

`Axes.bxp(bxpstats, positions=None, widths=None, vert=True, patch_artist=False, shownotches=False, showmeans=False, showcaps=True, showbox=True, showfliers=True, boxprops=None, whiskerprops=None, flierprops=None, medianprops=None, capprops=None, meanprops=None, meanline=False, manage_xticks=True, zorder=None)`

Drawing function for box and whisker plots.

Make a box and whisker plot for each column of `x` or each vector in sequence `x`. The box extends from the lower to upper quartile values of the data, with a line at the median. The whiskers extend from the box to show the range of the data. Flier points are those past the end of the whiskers.

Parameters

`bxpstats` [list of dicts] A list of dictionaries containing stats for each boxplot.

Required keys are:

- `med`: The median (scalar float).
- `q1`: The first quartile (25th percentile) (scalar float).
- `q3`: The third quartile (75th percentile) (scalar float).
- `whislo`: Lower bound of the lower whisker (scalar float).
- `whishi`: Upper bound of the upper whisker (scalar float).

Optional keys are:

- `mean`: The mean (scalar float). Needed if `showmeans=True`.
- `fliers`: Data beyond the whiskers (sequence of floats). Needed if `showfliers=True`.
- `cilo` & `cihi`: Lower and upper confidence intervals about the median. Needed if `shownotches=True`.
- `label`: Name of the dataset (string). If available, this will be used a tick label for the boxplot

`positions` [array-like, default = [1, 2, ..., n]] Sets the positions of the boxes. The ticks and limits are automatically set to match the positions.

`widths` [array-like, default = None] Either a scalar or a vector and sets the width of each box. The default is $0.15 \times (\text{distance between extreme positions})$, clipped to no less than 0.15 and no more than 0.5.

`vert` [bool, default = False] If `True` (default), makes the boxes vertical. If `False`, makes horizontal boxes.

`patch_artist` [bool, default = False] If `False` produces boxes with the `Line2D` artist. If `True` produces boxes with the `Patch` artist.

`shownotches` [bool, default = False] If `False` (default), produces a rectangular box plot. If `True`, will produce a notched box plot

`showmeans` [bool, default = False] If `True`, will toggle on the rendering of the means

`showcaps` [bool, default = True] If `True`, will toggle on the rendering of the caps

`showbox` [bool, default = True] If `True`, will toggle on the rendering of the box

`showfliers` [bool, default = True] If `True`, will toggle on the rendering of the fliers

`boxprops` [dict or None (default)] If provided, will set the plotting style of the boxes

`whiskerprops` [dict or None (default)] If provided, will set the plotting style of the whiskers

`capprops` [dict or None (default)] If provided, will set the plotting style of the caps

`flierprops` [dict or None (default)] If provided will set the plotting style of the fliers

`medianprops` [dict or None (default)] If provided, will set the plotting style of the medians

`meanprops` [dict or None (default)] If provided, will set the plotting style of the means

`meanline` [bool, default = False] If `True` (and `showmeans` is `True`), will try to render the mean as a line spanning the full width of the box according to `meanprops`. Not recommended if `shownotches` is also `True`. Otherwise, means will be shown as points.

`manage_xticks` [bool, default = True] If the function should adjust the `xlim` and `xtick` locations.

`zorder` [scalar, default = None] The `zorder` of the resulting boxplot

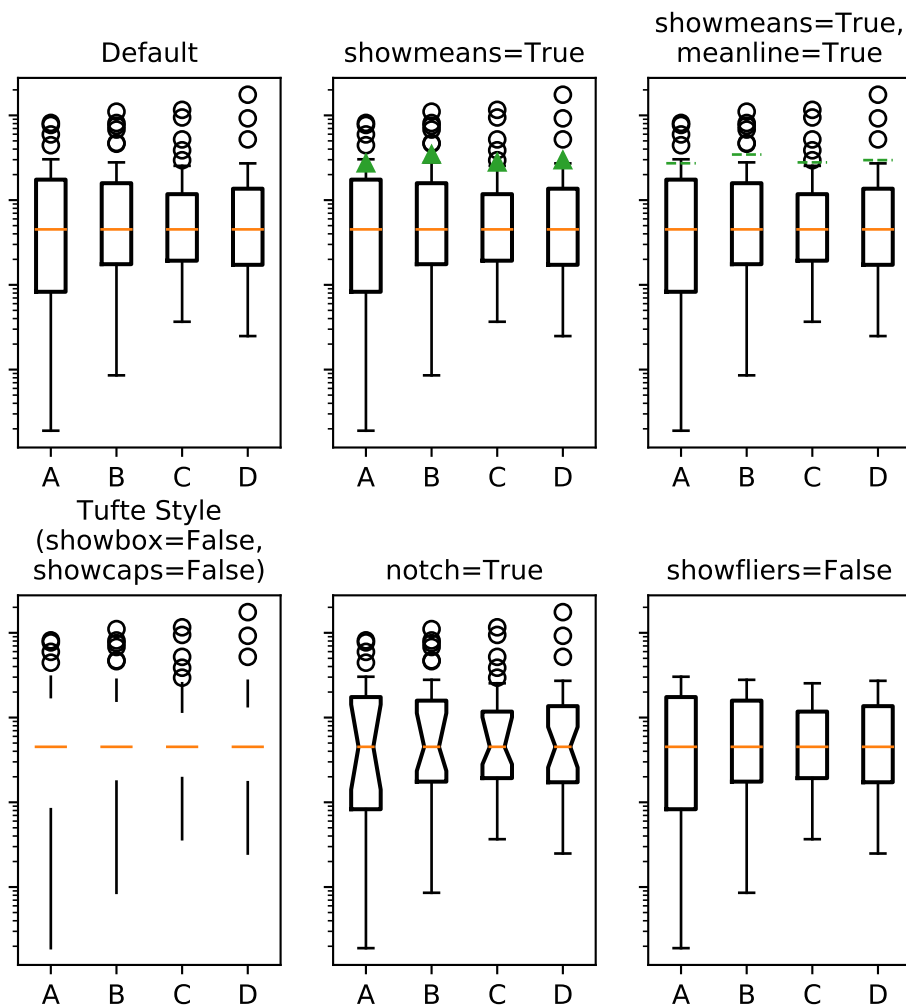
Returns

`result` [dict] A dictionary mapping each component of the boxplot to a list of the `matplotlib.lines.Line2D` instances created. That dictionary has the following keys (assuming vertical boxplots):

- `boxes`: the main body of the boxplot showing the quartiles and the median's confidence intervals if enabled.
- `medians`: horizontal lines at the median of each box.

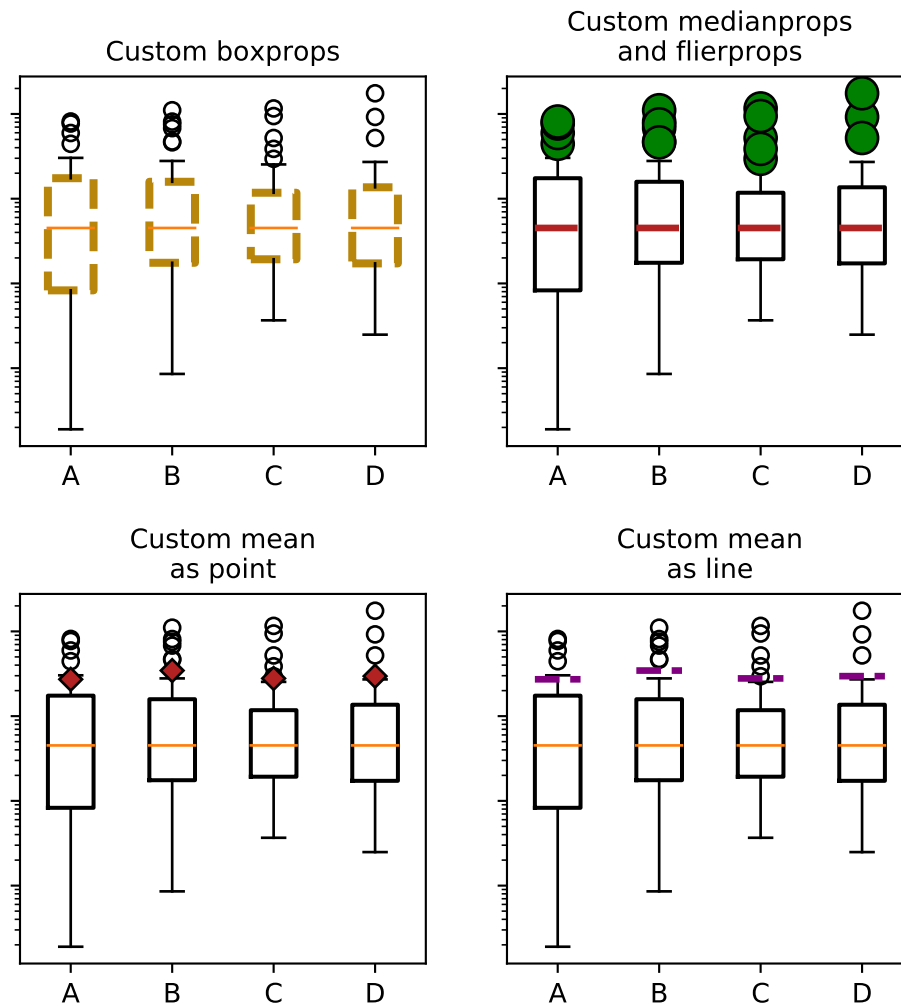
- whiskers: the vertical lines extending to the most extreme, non-outlier data points.
- caps: the horizontal lines at the ends of the whiskers.
- fliers: points representing data that extend beyond the whiskers (liers).
- means: points or lines representing the means.

Examples



35.1.5 Binned

I never said they'd be pretty



<code>Axes.hexbin</code>	Make a hexagonal binning plot.
<code>Axes.hist</code>	Plot a histogram.
<code>Axes.hist2d</code>	Make a 2D histogram plot.

`matplotlib.axes.Axes.hexbin`

`Axes.hexbin(x, y, C=None, gridsize=100, bins=None, xscale='linear', yscale='linear', extent=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, edgecolors='face', reduce_C_function=<function mean>, mincnt=None, marginals=False, *, data=None, **kwargs)`

Make a hexagonal binning plot.

Make a hexagonal binning plot of *x* versus *y*, where *x*, *y* are 1-D sequences of the same length, *N*. If *C* is *None* (the default), this is a histogram of the number of occurrences of the observations at (*x*[*i*],*y*[*i*]).

If *C* is specified, it specifies values at the coordinate (*x*[*i*],*y*[*i*]). These values are accumulated for each hexagonal bin and then reduced according to `reduce_C_function`, which defaults to numpy's mean function (`np.mean`). (If *C* is specified, it must also be a 1-D sequence of the same length as *x* and *y*.)

Parameters

x, *y* [array or masked array]

C [array or masked array, optional, default is *None*]

gridsize [int or (int, int), optional, default is 100] The number of hexagons in the *x*-direction, default is 100. The corresponding number of hexagons in the *y*-direction is chosen such that the hexagons are approximately regular. Alternatively, *gridsize* can be a tuple with two elements specifying the number of hexagons in the *x*-direction and the *y*-direction.

bins [{*'log'*} or int or sequence, optional, default is *None*] If *None*, no binning is applied; the color of each hexagon directly corresponds to its count value.

If *'log'*, use a logarithmic scale for the color map. Internally, $\log_{10}(i + 1)$ is used to determine the hexagon color.

If an integer, divide the counts in the specified number of bins, and color the hexagons accordingly.

If a sequence of values, the values of the lower bound of the bins to be used.

xscale [{*'linear'*, *'log'*}, optional, default is *'linear'*] Use a linear or log10 scale on the horizontal axis.

yscale [{*'linear'*, *'log'*}, optional, default is *'linear'*] Use a linear or log10 scale on the vertical axis.

mincnt [int > 0, optional, default is *None*] If not *None*, only display cells with more than *mincnt* number of points in the cell

marginals [bool, optional, default is False] if **marginals** is True, plot the marginal density as colormapped rectangles along the bottom of the x-axis and left of the y-axis

extent [scalar, optional, default is None] The limits of the bins. The default assigns the limits based on **gridsize**, **x**, **y**, **xscale** and **yscale**.

If **xscale** or **yscale** is set to 'log', the limits are expected to be the exponent for a power of 10. E.g. for x-limits of 1 and 50 in 'linear' scale and y-limits of 10 and 1000 in 'log' scale, enter (1, 50, 1, 3).

Order of scalars is (left, right, bottom, top).

Returns

object a PolyCollection instance; use `get_array()` on this PolyCollection to get the counts in each hexagon.

If **marginals** is True, horizontal bar and vertical bar (both PolyCollections) will be attached to the return collection as attributes **hbar** and **vbar**.

Other Parameters

cmap [object, optional, default is None] a `matplotlib.colors.Colormap` instance. If None, defaults to `rc image.cmap`.

norm [object, optional, default is None] `matplotlib.colors.Normalize` instance is used to scale luminance data to 0,1.

vmin, **vmax** [scalar, optional, default is None] **vmin** and **vmax** are used in conjunction with **norm** to normalize luminance data. If None, the min and max of the color array **C** are used. Note if you pass a **norm** instance your settings for **vmin** and **vmax** will be ignored.

alpha [scalar between 0 and 1, optional, default is None] the alpha value for the patches

linewidths [scalar, optional, default is None] If None, defaults to 1.0.

edgecolors [{ 'face', 'none', None } or color, optional] If 'face' (the default), draws the edges in the same color as the fill color.

If 'none', no edge is drawn; this can sometimes lead to unsightly unpainted pixels between the hexagons.

If None, draws outlines in the default color.

If a matplotlib color arg, draws outlines in the specified color.

Notes

The standard descriptions of all the Collection parameters:

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a
alpha		float or None
animated		bool
antialiased or antialiaseds		Boolean or sequence of booleans
array		ndarray
capstyle		unknown
clim		a length 2 sequence of floats; may be overridden in methods that have vmin and vm
clip_box		a Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
cmap		a colormap or registered colormap name
color		matplotlib color arg or sequence of rgba tuples
contains		a callable function
edgecolor or edgecolors		matplotlib color spec or sequence of specs
facecolor or facecolors		matplotlib color spec or sequence of specs
figure		a Figure instance
gid		an id string
hatch		['/' '\' '.' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle		unknown
label		object
linestyle or dashes or linestyles		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ...]
linewidth or linewidths or lw		float or sequence of floats
norm		Normalize
offset_position		['screen' 'data']
offsets		float or sequence of floats
path_effects		AbstractPathEffect
picker		[None bool float callable]
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
transform		Transform
url		a url string
urls		List[str] or None
visible		bool
zorder		float

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All arguments with the following names: 'x', 'y'.

`matplotlib.axes.Axes.hist`

```
Axes.hist(x, bins=None, range=None, density=None, weights=None, cumulative=False,  
          bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None,  
          log=False, color=None, label=None, stacked=False, normed=None, *, data=None,  
          **kwargs)
```

Plot a histogram.

Compute and draw the histogram of `x`. The return value is a tuple `(n, bins, patches)` or `([n0, n1, ...], bins, [patches0, patches1,...])` if the input contains multiple data.

Multiple data can be provided via `x` as a list of datasets of potentially different length (`[x0, x1, ...]`), or as a 2-D ndarray in which each column is a dataset. Note that the ndarray form is transposed relative to the list form.

Masked arrays are not supported at present.

Parameters

`x` [(`n`,) array or sequence of (`n`,) arrays] Input values, this takes either a single array or a sequence of arrays which are not required to be of the same length

`bins` [integer or sequence or 'auto', optional] If an integer is given, `bins + 1` bin edges are calculated and returned, consistent with `numpy.histogram()`.

If `bins` is a sequence, gives bin edges, including left edge of first bin and right edge of last bin. In this case, `bins` is returned unmodified.

All but the last (righthand-most) bin is half-open. In other words, if `bins` is:

`[1, 2, 3, 4]`

then the first bin is `[1, 2)` (including 1, but excluding 2) and the second `[2, 3)`. The last bin, however, is `[3, 4]`, which includes 4.

Unequally spaced bins are supported if `bins` is a sequence.

If Numpy 1.11 is installed, may also be 'auto'.

Default is taken from the rcParam `hist.bins`.

`range` [tuple or None, optional] The lower and upper range of the bins. Lower and upper outliers are ignored. If not provided, range is `(x.min(), x.max())`. Range has no effect if `bins` is a sequence.

If `bins` is a sequence or range is specified, autoscaling is based on the specified bin range instead of the range of `x`.

Default is None

`density` [boolean, optional] If True, the first element of the return tuple will be the counts normalized to form a probability density, i.e., the area (or integral) under the histogram will sum to 1. This is achieved by dividing the

count by the number of observations times the bin width and not dividing by the total number of observations. If stacked is also True, the sum of the histograms is normalized to 1.

Default is None for both normed and density. If either is set, then that value will be used. If neither are set, then the args will be treated as False.

If both density and normed are set an error is raised.

weights [(n,) array_like or None, optional] An array of weights, of the same shape as x. Each value in x only contributes its associated weight towards the bin count (instead of 1). If normed or density is True, the weights are normalized, so that the integral of the density over the range remains 1.

Default is None

cumulative [boolean, optional] If True, then a histogram is computed where each bin gives the counts in that bin plus all bins for smaller values. The last bin gives the total number of datapoints. If normed or density is also True then the histogram is normalized such that the last bin equals 1. If cumulative evaluates to less than 0 (e.g., -1), the direction of accumulation is reversed. In this case, if normed and/or density is also True, then the histogram is normalized such that the first bin equals 1.

Default is False

bottom [array_like, scalar, or None] Location of the bottom baseline of each bin. If a scalar, the base line for each bin is shifted by the same amount. If an array, each bin is shifted independently and the length of bottom must match the number of bins. If None, defaults to 0.

Default is None

histtype [{ 'bar', 'barstacked', 'step', 'stepfilled' }, optional] The type of histogram to draw.

- 'bar' is a traditional bar-type histogram. If multiple data are given the bars are arranged side by side.
- 'barstacked' is a bar-type histogram where multiple data are stacked on top of each other.
- 'step' generates a lineplot that is by default unfilled.
- 'stepfilled' generates a lineplot that is by default filled.

Default is 'bar'

align [{ 'left', 'mid', 'right' }, optional] Controls how the histogram is plotted.

- 'left': bars are centered on the left bin edges.
- 'mid': bars are centered between the bin edges.
- 'right': bars are centered on the right bin edges.

Default is 'mid'

`orientation` [{`'horizontal'`, `'vertical'`}, optional] If `'horizontal'`, `barh` will be used for bar-type histograms and the bottom kwarg will be the left edges.

`rwidth` [scalar or `None`, optional] The relative width of the bars as a fraction of the bin width. If `None`, automatically compute the width.

Ignored if `histtype` is `'step'` or `'stepfilled'`.

Default is `None`

`log` [boolean, optional] If `True`, the histogram axis will be set to a log scale. If `log` is `True` and `x` is a 1D array, empty bins will be filtered out and only the non-empty (`n`, `bins`, `patches`) will be returned.

Default is `False`

`color` [color or array_like of colors or `None`, optional] Color spec or sequence of color specs, one per dataset. Default (`None`) uses the standard line color sequence.

Default is `None`

`label` [string or `None`, optional] String, or sequence of strings to match multiple datasets. Bar charts yield multiple patches per dataset, but only the first gets the label, so that the legend command will work as expected.

default is `None`

`stacked` [boolean, optional] If `True`, multiple data are stacked on top of each other. If `False` multiple data are arranged side by side if `histtype` is `'bar'` or on top of each other if `histtype` is `'step'`

Default is `False`

`normed` [bool, optional] Deprecated; use the density keyword argument instead.

Returns

`n` [array or list of arrays] The values of the histogram bins. See `normed` or `density` and `weights` for a description of the possible semantics. If input `x` is an array, then this is an array of length `nbins`. If input is a sequence arrays [`data1`, `data2`,...], then this is a list of arrays with the values of the histograms for each of the arrays in the same order.

`bins` [array] The edges of the bins. Length `nbins + 1` (`nbins` left edges and right edge of last bin). Always a single array even when multiple data sets are passed in.

`patches` [list or list of lists] Silent list of individual patches used to create the histogram or list of such list if multiple input datasets.

Other Parameters

`**kwargs` [Patch properties]

See also:

hist2d 2D histograms

Notes

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'weights', 'x'.
-

Examples using `matplotlib.axes.Axes.hist`

- `sphx_glr_gallery_statistics_histogram_features.py`

`matplotlib.axes.Axes.hist2d`

`Axes.hist2d(x, y, bins=10, range=None, normed=False, weights=None, cmin=None, cmax=None, *, data=None, **kwargs)`

Make a 2D histogram plot.

Parameters

`x, y`: `array_like`, `shape (n,)` Input values

`bins`: [`None` | `int` | [`int`, `int`] | `array_like` | [`array`, `array`]] The bin specification:

- If `int`, the number of bins for the two dimensions (`nx=ny=bins`).
- If [`int`, `int`], the number of bins in each dimension (`nx, ny = bins`).
- If `array_like`, the bin edges for the two dimensions (`x_edges=y_edges=bins`).
- If [`array`, `array`], the bin edges in each dimension (`x_edges, y_edges = bins`).

The default value is 10.

`range` [`array_like` `shape(2, 2)`, optional, default: `None`] The leftmost and rightmost edges of the bins along each dimension (if not specified explicitly in the bins parameters): [`xmin, xmax`], [`ymin, ymax`]]. All values outside of this range will be considered outliers and not tallied in the histogram.

`normed` [`boolean`, optional, default: `False`] Normalize histogram.

`weights` [`array_like`, `shape (n,)`, optional, default: `None`] An array of values `w_i` weighing each sample (`x_i, y_i`).

`cmin` [scalar, optional, default: None] All bins that has count less than `cmin` will not be displayed and these count values in the return value `count` histogram will also be set to `nan` upon return

`cmax` [scalar, optional, default: None] All bins that has count more than `cmax` will not be displayed (set to `none` before passing to `imshow`) and these count values in the return value `count` histogram will also be set to `nan` upon return

Returns

`h` [2D array] The bi-dimensional histogram of samples `x` and `y`. Values in `x` are histogrammed along the first dimension and values in `y` are histogrammed along the second dimension.

`xedges` [1D array] The bin edges along the `x` axis.

`yedges` [1D array] The bin edges along the `y` axis.

`image` [AxesImage]

Other Parameters

`cmap` [Colormap or str, optional] A `colors.Colormap` instance. If not set, use `rc` settings.

`norm` [Normalize, optional] A `colors.Normalize` instance is used to scale luminance data to `[0, 1]`. If not set, defaults to `colors.Normalize()`.

`vmin/vmax` [None or scalar, optional] Arguments passed to the `Normalize` instance.

`alpha` [`0 <= scalar <= 1` or None, optional] The alpha blending value.

See also:

`hist` 1D histogram plotting

Notes

- Currently `hist2d` calculates it's own axis limits, and any limits previously set are ignored.
- Rendering the histogram with a logarithmic color scale is accomplished by passing a `colors.LogNorm` instance to the `norm` keyword argument. Likewise, power-law normalization (similar in effect to gamma correction) can be accomplished with `colors.PowerNorm`.

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: `'weights'`, `'x'`, `'y'`.
-

35.1.6 Contours

<code>Axes.clabel</code>	Label a contour plot.
<code>Axes.contour</code>	Plot contours.
<code>Axes.contourf</code>	Plot contours.

`matplotlib.axes.Axes.clabel`

`Axes.clabel(CS, *args, **kwargs)`

Label a contour plot.

Call signature:

```
clabel(cs, **kwargs)
```

Adds labels to line contours in `cs`, where `cs` is a `ContourSet` object returned by `contour`.

```
clabel(cs, v, **kwargs)
```

only labels contours listed in `v`.

Parameters

`fontsize` [string or float, optional] Size in points or relative size e.g., 'smaller', 'x-large'. See `Text.set_size` for accepted string values.

`colors` : Color of each label

- if `None`, the color of each label matches the color of the corresponding contour
- if one string color, e.g., `colors = 'r'` or `colors = 'red'`, all labels will be plotted in this color
- if a tuple of matplotlib color args (string, float, rgb, etc), different labels will be plotted in different colors in the order specified

`inline` [bool, optional] If `True` the underlying contour is removed where the label is placed. Default is `True`.

`inline_spacing` [float, optional] Space in pixels to leave on each side of label when placing inline. Defaults to 5.

This spacing will be exact for labels at locations where the contour is straight, less so for labels on curved contours.

`fmt` [string or dict, optional] A format string for the label. Default is `'%1.3f'`

Alternatively, this can be a dictionary matching contour levels with arbitrary strings to use for each contour level (i.e., `fmt[level]=string`), or it can be any callable, such as a `Formatter` instance, that returns a string when called with a numeric contour level.

`manual` [bool or iterable, optional] If True, contour labels will be placed manually using mouse clicks. Click the first button near a contour to add a label, click the second button (or potentially both mouse buttons at once) to finish adding labels. The third button can be used to remove the last label added, but only if labels are not inline. Alternatively, the keyboard can be used to select label locations (enter to end label placement, delete or backspace act like the third mouse button, and any other key will select a label location).

`manual` can also be an iterable object of x,y tuples. Contour labels will be created as if mouse is clicked at each x,y positions.

`rightside_up` [bool, optional] If True, label rotations will always be plus or minus 90 degrees from level. Default is True.

`use_clabeltext` [bool, optional] If True, `ClabelText` class (instead of `Text`) is used to create labels. `ClabelText` recalculates rotation angles of texts during the drawing time, therefore this can be used if aspect of the axes changes. Default is False.

Examples using `matplotlib.axes.Axes.clabel`

- `sphx_glr_gallery_images_contours_and_fields_contour_label_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_contour_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_contourf_demo.py`

`matplotlib.axes.Axes.contour`

`Axes.contour(*args, data=None, **kwargs)`

Plot contours.

Call signature:

`contour([X, Y], Z, [levels], **kwargs)`

`contour()` and `contourf()` draw contour lines and filled contours, respectively. Except as noted, function signatures and return values are the same for both versions.

Parameters

`X, Y` [array-like, optional] The coordinates of the values in `Z`.

`X` and `Y` must both be 2-D with the same shape as `Z` (e.g. created via `numpy.meshgrid()`), or they must both be 1-D such that `len(X) == M` is the number of columns in `Z` and `len(Y) == N` is the number of rows in `Z`.

If not given, they are assumed to be integer indices, i.e. `X = range(M)`, `Y = range(N)`.

`Z` [array-like(N, M)] The height values over which the contour is drawn.

levels [int or array-like, optional] Determines the number and positions of the contour lines / regions.

If an int *n*, use *n* data intervals; i.e. draw *n*+1 contour lines. The level heights are automatically chosen.

If array-like, draw contour lines at the specified levels. The values must be in increasing order.

Returns

:class:~matplotlib.contour.QuadContourSet

Other Parameters

corner_mask [bool, optional] Enable/disable corner masking, which only has an effect if *Z* is a masked array. If False, any quad touching a masked point is masked out. If True, only the triangular corners of quads nearest those points are always masked out, other triangular corners comprising three unmasked points are contoured as usual.

Defaults to rcParams['contour.corner_mask'], which defaults to True.

colors [color string or sequence of colors, optional] The colors of the levels, i.e. the lines for contour and the areas for contourf.

The sequence is cycled for the levels in ascending order. If the sequence is shorter than the number of levels, it's repeated.

As a shortcut, single color strings may be used in place of one-element lists, i.e. 'red' instead of ['red'] to color all levels with the same color. This shortcut does only work for color strings, not for other ways of specifying colors.

By default (value None), the colormap specified by *cmap* will be used.

alpha [float, optional] The alpha blending value, between 0 (transparent) and 1 (opaque).

cmap [str or Colormap, optional] A Colormap instance or registered colormap name. The colormap maps the level values to colors. Defaults to rcParams["image.cmap"].

If given, colors take precedence over *cmap*.

norm [Normalize, optional] If a colormap is used, the Normalize instance scales the level values to the canonical colormap range [0, 1] for mapping to colors. If not given, the default linear scaling is used.

vmin, vmax [float, optional] If not None, either or both of these values will be supplied to the Normalize instance, overriding the default color scaling based on levels.

origin [{None, 'upper', 'lower', 'image'}, optional] Determines the orientation and exact position of *Z* by specifying the position of *Z*[0, 0]. This is only relevant, if *X*, *Y* are not given.

- None: $Z[0, 0]$ is at $X=0$, $Y=0$ in the lower left corner.
- 'lower': $Z[0, 0]$ is at $X=0.5$, $Y=0.5$ in the lower left corner.
- 'upper': $Z[0, 0]$ is at $X=N+0.5$, $Y=0.5$ in the upper left corner.
- 'image': Use the value from `rcParams["image.origin"]`. Note: The value None in the rcParam is currently handled as 'lower'.

extent [(x0, x1, y0, y1), optional] If origin is not None, then extent is interpreted as in `matplotlib.pyplot.imshow()`: it gives the outer pixel boundaries. In this case, the position of $Z[0,0]$ is the center of the pixel, not a corner. If origin is None, then (x0, y0) is the position of $Z[0,0]$, and (x1, y1) is the position of $Z[-1,-1]$.

This keyword is not active if X and Y are specified in the call to `contour`.

locator [ticker.Locator subclass, optional] The locator is used to determine the contour levels if they are not given explicitly via levels. Defaults to `MaxNLocator`.

extend [{'neither', 'both', 'min', 'max'}, optional] Unless this is 'neither', contour levels are automatically added to one or both ends of the range so that all data are included. These added ranges are then mapped to the special colormap values which default to the ends of the colormap range, but can be set via `matplotlib.colors.Colormap.set_under()` and `matplotlib.colors.Colormap.set_over()` methods.

xunits, yunits [registered units, optional] Override axis units by specifying an instance of a `matplotlib.units.ConversionInterface`.

antialiased [bool, optional] Enable antialiasing, overriding the defaults. For filled contours, the default is True. For line contours, it is taken from `rcParams["lines.antialiased"]`.

Nchunk [int ≥ 0 , optional] If 0, no subdivision of the domain. Specify a positive integer to divide the domain into subdomains of nchunk by nchunk quads. Chunking reduces the maximum length of polygons generated by the contouring algorithm which reduces the rendering workload passed on to the backend and also requires slightly less RAM. It can however introduce rendering artifacts at chunk boundaries depending on the backend, the antialiased flag and value of alpha.

linewidths [float or sequence of float, optional] Only applies to `contour`.

The line width of the contour lines.

If a number, all levels will be plotted with this linewidth.

If a sequence, the levels in ascending order will be plotted with the linewidths in the order specified.

Defaults to `rcParams["lines.linewidth"]`.

linestyles [{None, 'solid', 'dashed', 'dashdot', 'dotted'}, optional] Only applies to `contour`.

If `linestyles` is `None`, the default is `'solid'` unless the lines are monochrome. In that case, negative contours will take their linestyle from `rcParams["contour.negative_linestyle"]` setting.

`linestyles` can also be an iterable of the above strings specifying a set of linestyles to be used. If this iterable is shorter than the number of contour levels it will be repeated as necessary.

`hatches` [List[str], optional] Only applies to `contourf`.

A list of cross hatch patterns to use on the filled areas. If `None`, no hatching will be added to the contour. Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Notes

1. `contourf()` differs from the MATLAB version in that it does not draw the polygon edges. To draw edges, add line contours with calls to `contour()`.
2. `contourf` fills intervals that are closed at the top; that is, for boundaries `z1` and `z2`, the filled region is:

$$z1 < Z \leq z2$$

There is one exception: if the lowest boundary coincides with the minimum value of the `Z` array, then that minimum value will be included in the lowest interval.

Examples using `matplotlib.axes.Axes.contour`

- `sphx_glr_gallery_images_contours_and_fields_contour_corner_mask.py`
- `sphx_glr_gallery_images_contours_and_fields_contourf_hatching.py`
- `sphx_glr_gallery_images_contours_and_fields_contour_label_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_irregulardatagrid.py`
- `sphx_glr_gallery_images_contours_and_fields_contour_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_image_transparency_blend.py`
- `sphx_glr_gallery_images_contours_and_fields_contour_image.py`
- `sphx_glr_gallery_images_contours_and_fields_contourf_demo.py`

`matplotlib.axes.Axes.contourf`

`Axes.contourf(*args, data=None, **kwargs)`

Plot contours.

Call signature:

`contour([X, Y], Z, [levels], **kwargs)`

`contour()` and `contourf()` draw contour lines and filled contours, respectively. Except as noted, function signatures and return values are the same for both versions.

Parameters

X, Y [array-like, optional] The coordinates of the values in Z.

X and Y must both be 2-D with the same shape as Z (e.g. created via `numpy.meshgrid()`), or they must both be 1-D such that `len(X) == M` is the number of columns in Z and `len(Y) == N` is the number of rows in Z.

If not given, they are assumed to be integer indices, i.e. `X = range(M)`, `Y = range(N)`.

Z [array-like(N, M)] The height values over which the contour is drawn.

levels [int or array-like, optional] Determines the number and positions of the contour lines / regions.

If an int `n`, use `n` data intervals; i.e. draw `n+1` contour lines. The level heights are automatically chosen.

If array-like, draw contour lines at the specified levels. The values must be in increasing order.

Returns

:class:~matplotlib.contour.QuadContourSet

Other Parameters

corner_mask [bool, optional] Enable/disable corner masking, which only has an effect if Z is a masked array. If False, any quad touching a masked point is masked out. If True, only the triangular corners of quads nearest those points are always masked out, other triangular corners comprising three unmasked points are contoured as usual.

Defaults to `rcParams['contour.corner_mask']`, which defaults to True.

colors [color string or sequence of colors, optional] The colors of the levels, i.e. the lines for `contour` and the areas for `contourf`.

The sequence is cycled for the levels in ascending order. If the sequence is shorter than the number of levels, it's repeated.

As a shortcut, single color strings may be used in place of one-element lists, i.e. 'red' instead of ['red'] to color all levels with the same color. This shortcut does only work for color strings, not for other ways of specifying colors.

By default (value None), the colormap specified by `cmap` will be used.

alpha [float, optional] The alpha blending value, between 0 (transparent) and 1 (opaque).

cmap [str or Colormap, optional] A Colormap instance or registered colormap name. The colormap maps the level values to colors. Defaults to `rcParams["image.cmap"]`.

If given, colors take precedence over `cmap`.

norm [Normalize, optional] If a colormap is used, the Normalize instance scales the level values to the canonical colormap range `[0, 1]` for mapping to colors. If not given, the default linear scaling is used.

vmin, vmax [float, optional] If not None, either or both of these values will be supplied to the Normalize instance, overriding the default color scaling based on levels.

origin [{None, 'upper', 'lower', 'image'}, optional] Determines the orientation and exact position of Z by specifying the position of `Z[0, 0]`. This is only relevant, if X, Y are not given.

- None: `Z[0, 0]` is at `X=0, Y=0` in the lower left corner.
- 'lower': `Z[0, 0]` is at `X=0.5, Y=0.5` in the lower left corner.
- 'upper': `Z[0, 0]` is at `X=N+0.5, Y=0.5` in the upper left corner.
- 'image': Use the value from `rcParams["image.origin"]`. Note: The value None in the rcParam is currently handled as 'lower'.

extent [(x0, x1, y0, y1), optional] If origin is not None, then extent is interpreted as in `matplotlib.pyplot.imshow()`: it gives the outer pixel boundaries. In this case, the position of `Z[0,0]` is the center of the pixel, not a corner. If origin is None, then `(x0, y0)` is the position of `Z[0,0]`, and `(x1, y1)` is the position of `Z[-1,-1]`.

This keyword is not active if X and Y are specified in the call to `contour`.

locator [ticker.Locator subclass, optional] The locator is used to determine the contour levels if they are not given explicitly via levels. Defaults to `MaxNLocator`.

extend [{'neither', 'both', 'min', 'max'}, optional] Unless this is 'neither', contour levels are automatically added to one or both ends of the range so that all data are included. These added ranges are then mapped to the special colormap values which default to the ends of the colormap range, but can be set via `matplotlib.colors.Colormap.set_under()` and `matplotlib.colors.Colormap.set_over()` methods.

xunits, yunits [registered units, optional] Override axis units by specifying an instance of a `matplotlib.units.ConversionInterface`.

antialiased [bool, optional] Enable antialiasing, overriding the defaults. For filled contours, the default is True. For line contours, it is taken from `rcParams["lines.antialiased"]`.

Nchunk [int ≥ 0 , optional] If 0, no subdivision of the domain. Specify a positive integer to divide the domain into subdomains of `nchunk` by `nchunk`

quads. Chunking reduces the maximum length of polygons generated by the contouring algorithm which reduces the rendering workload passed on to the backend and also requires slightly less RAM. It can however introduce rendering artifacts at chunk boundaries depending on the backend, the antialiased flag and value of alpha.

`linewidths` [float or sequence of float, optional] Only applies to `contour`.

The line width of the contour lines.

If a number, all levels will be plotted with this linewidth.

If a sequence, the levels in ascending order will be plotted with the linewidths in the order specified.

Defaults to `rcParams["lines.linewidth"]`.

`linestyles` [{None, 'solid', 'dashed', 'dashdot', 'dotted'}, optional] Only applies to `contour`.

If `linestyles` is None, the default is 'solid' unless the lines are monochrome. In that case, negative contours will take their linestyle from `rcParams["contour.negative_linestyle"]` setting.

`linestyles` can also be an iterable of the above strings specifying a set of linestyles to be used. If this iterable is shorter than the number of contour levels it will be repeated as necessary.

`hatches` [List[str], optional] Only applies to `contourf`.

A list of cross hatch patterns to use on the filled areas. If None, no hatching will be added to the contour. Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Notes

1. `contourf()` differs from the MATLAB version in that it does not draw the polygon edges. To draw edges, add line contours with calls to `contour()`.
2. `contourf` fills intervals that are closed at the top; that is, for boundaries `z1` and `z2`, the filled region is:

$$z1 < Z \leq z2$$

There is one exception: if the lowest boundary coincides with the minimum value of the `Z` array, then that minimum value will be included in the lowest interval.

Examples using `matplotlib.axes.Axes.contourf`

- `sphx_glr_gallery_images_contours_and_fields_contour_corner_mask.py`
- `sphx_glr_gallery_images_contours_and_fields_contourf_hatching.py`

- sphx_glr_gallery_images_contours_and_fields_contourf_log.py
- sphx_glr_gallery_images_contours_and_fields_pcolormesh_levels.py
- sphx_glr_gallery_images_contours_and_fields_irregulardatagrid.py
- sphx_glr_gallery_images_contours_and_fields_triinterp_demo.py
- sphx_glr_gallery_images_contours_and_fields_contourf_demo.py

35.1.7 Array

<code>Axes.imshow</code>	Display an image on the axes.
<code>Axes.matshow</code>	Plot the values of a 2D matrix or array as color-coded image.
<code>Axes.pcolor</code>	Create a pseudocolor plot with a non-regular rectangular grid.
<code>Axes.pcolorfast</code>	Create a pseudocolor plot with a non-regular rectangular grid.
<code>Axes.pcolormesh</code>	Create a pseudocolor plot with a non-regular rectangular grid.
<code>Axes.spy</code>	Plot the sparsity pattern on a 2-D array.

matplotlib.axes.Axes.imshow

`Axes.imshow(X, cmap=None, norm=None, aspect=None, interpolation=None, alpha=None, vmin=None, vmax=None, origin=None, extent=None, shape=None, filternorm=1, filterrad=4.0, imlim=None, resample=None, url=None, *, data=None, **kwargs)`

Display an image on the axes.

Parameters

X [array_like, shape (n, m) or (n, m, 3) or (n, m, 4)] Display the image in X to current axes. X may be an array or a PIL image. If X is an array, it can have the following shapes and types:

- MxN – values to be mapped (float or int)
- MxNx3 – RGB (float or uint8)
- MxNx4 – RGBA (float or uint8)

MxN arrays are mapped to colors based on the norm (mapping scalar to scalar) and the cmap (mapping the normed scalar to a color).

Elements of RGB and RGBA arrays represent pixels of an MxN image. All values should be in the range [0 .. 1] for floats or [0 .. 255] for integers. Out-of-range values will be clipped to these bounds.

cmap [Colormap, optional, default: None] If None, default to rc image.cmap value. cmap is ignored if X is 3-D, directly specifying RGB(A) values.

aspect `[[‘auto’ | ‘equal’ | scalar], optional, default: None]` If ‘auto’, changes the image aspect ratio to match that of the axes.

If ‘equal’, and extent is None, changes the axes aspect ratio to match that of the image. If extent is not `None`, the axes aspect ratio is changed to match that of the extent.

If None, default to `rc image.aspect` value.

interpolation `[string, optional, default: None]` Acceptable values are ‘none’, ‘nearest’, ‘bilinear’, ‘bicubic’, ‘spline16’, ‘spline36’, ‘hanning’, ‘hamming’, ‘hermite’, ‘kaiser’, ‘quadric’, ‘catrom’, ‘gaussian’, ‘bessel’, ‘mitchell’, ‘sinc’, ‘lanczos’

If interpolation is None, default to `rc image.interpolation`. See also the `filtnorm` and `filtrrad` parameters. If interpolation is ‘none’, then no interpolation is performed on the Agg, ps and pdf backends. Other backends will fall back to ‘nearest’.

norm `[Normalize, optional, default: None]` A Normalize instance is used to scale a 2-D float X input to the (0, 1) range for input to the `cmap`. If norm is None, use the default `func:normalize`. If norm is an instance of `NoNorm`, X must be an array of integers that index directly into the lookup table of the `cmap`.

vmin, vmax `[scalar, optional, default: None]` vmin and vmax are used in conjunction with norm to normalize luminance data. Note if you pass a norm instance, your settings for vmin and vmax will be ignored.

alpha `[scalar, optional, default: None]` The alpha blending value, between 0 (transparent) and 1 (opaque). The alpha argument is ignored for RGBA input data.

origin `[[‘upper’ | ‘lower’], optional, default: None]` Place the [0,0] index of the array in the upper left or lower left corner of the axes. If None, default to `rc image.origin`.

extent `[scalars (left, right, bottom, top), optional, default: None]` The location, in data-coordinates, of the lower-left and upper-right corners. If `None`, the image is positioned such that the pixel centers fall on zero-based (row, column) indices.

shape `[scalars (columns, rows), optional, default: None]` For raw buffer images

filtnorm `[scalar, optional, default: 1]` A parameter for the antigrain image resize filter. From the antigrain documentation, if `filtnorm = 1`, the filter normalizes integer values and corrects the rounding errors. It doesn’t do anything with the source floating point values, it corrects only integers according to the rule of 1.0 which means that any sum of pixel weights must be equal to 1.0. So, the filter function must produce a graph of the proper shape.

filtrrad `[scalar, optional, default: 4.0]` The filter radius for filters that have a radius parameter, i.e. when interpolation is one of: ‘sinc’, ‘lanczos’ or

'blackman'

Returns

image [AxesImage]

Other Parameters

**kwargs [Artist properties.]

See also:

matshow Plot a matrix or an array as an image.

Notes

Unless extent is used, pixel centers will be located at integer coordinates. In other words: the origin will coincide with the center of pixel (0, 0).

Two typical representations are used for RGB images with an alpha channel:

- Straight (unassociated) alpha: R, G, and B channels represent the color of the pixel, disregarding its opacity.
- Premultiplied (associated) alpha: R, G, and B channels represent the color of the pixel, adjusted for its opacity by multiplication.

imshow expects RGB images adopting the straight (unassociated) alpha representation.

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All positional and all keyword arguments.
-

Examples using matplotlib.axes.Axes.imshow

- sphx_glr_gallery_color_colorbar_basics.py
- sphx_glr_gallery_color_colormap_reference.py
- sphx_glr_gallery_images_contours_and_fields_image_clip_path.py
- sphx_glr_gallery_images_contours_and_fields_barcode_demo.py
- sphx_glr_gallery_images_contours_and_fields_interpolation_methods.py
- sphx_glr_gallery_images_contours_and_fields_layer_images.py
- sphx_glr_gallery_images_contours_and_fields_affine_image.py
- sphx_glr_gallery_images_contours_and_fields_shading_example.py

- sphx_glr_gallery_images_contours_and_fields_multi_image.py
- sphx_glr_gallery_images_contours_and_fields_image_demo.py
- sphx_glr_gallery_images_contours_and_fields_image_masked.py
- sphx_glr_gallery_images_contours_and_fields_pcolor_demo.py
- sphx_glr_gallery_images_contours_and_fields_image_transparency_blend.py
- sphx_glr_gallery_images_contours_and_fields_contour_image.py
- sphx_glr_gallery_images_contours_and_fields_custom_cmap.py
- sphx_glr_gallery_images_contours_and_fields_image_annotated_heatmap.py

matplotlib.axes.Axes.matshow

Axes.matshow(Z, **kwargs)

Plot the values of a 2D matrix or array as color-coded image.

The matrix will be shown the way it would be printed, with the first row at the top. Row and column numbering is zero-based.

Parameters

Z [array-like(N, M)] The matrix to be displayed.

Returns

image [AxesImage]

Other Parameters

**kwargs [imshow arguments]

See also:

imshow More general function to plot data on a 2D regular raster.

Notes

This is just a convenience function wrapping imshow to set useful defaults for a displaying a matrix. In particular:

- Set origin='upper'.
- Set interpolation='nearest'.
- Set aspect='equal'.
- Ticks are placed to the left and above.
- Ticks are formatted to show integer indices.

Examples using `matplotlib.axes.Axes.matshow`

- `sphx_glr_gallery_images_contours_and_fields_matshow.py`

`matplotlib.axes.Axes.pcolor`

`Axes.pcolor(*args, data=None, **kwargs)`

Create a pseudocolor plot with a non-regular rectangular grid.

Call signature:

```
pcolor([X, Y,] C, **kwargs)
```

X and Y can be used to specify the corners of the quadrilaterals.

Hint: `pcolor()` can be very slow for large arrays. In most cases you should use the the similar but much faster `pcolormesh` instead. See there for a discussion of the differences.

Parameters

C [array_like] A scalar 2-D array. The values will be color-mapped.

X, Y [array_like, optional] The coordinates of the quadrilateral corners. The quadrilateral for `C[i,j]` has corners at:

$(X[i+1, j], Y[i+1, j])$	$(X[i+1, j+1], Y[i+1, j+1])$
$+-----+$	
$ C[i,j] $	
$+-----+$	
$(X[i, j], Y[i, j])$	$(X[i, j+1], Y[i, j+1]),$

Note that the column index corresponds to the x-coordinate, and the row index corresponds to y. For details, see the Notes section below.

The dimensions of X and Y should be one greater than those of C. Alternatively, X, Y and C may have equal dimensions, in which case the last row and column of C will be ignored.

If X and/or Y are 1-D arrays or column vectors they will be expanded as needed into the appropriate 2-D arrays, making a rectangular grid.

cmap [str or Colormap, optional] A Colormap instance or registered colormap name. The colormap maps the C values to colors. Defaults to `rcParams["image.cmap"]`.

norm [Normalize, optional] The Normalize instance scales the data values to the canonical colormap range [0, 1] for mapping to colors. By default, the data range is mapped to the colorbar range using linear scaling.

`vmin, vmax` [scalar, optional, default: `None`] The colorbar range. If `None`, suitable min/max values are automatically chosen by the `Normalize` instance (defaults to the respective min/max values of `C` in case of the default linear scaling).

`edgecolors` [{`'none'`, `None`, `'face'`, color, color sequence}, optional] The color of the edges. Defaults to `'none'`. Possible values:

- `'none'` or `''`: No edge.
- `None`: `rcParams["patch.edgecolor"]` will be used. Note that currently `rcParams["patch.force_edgecolor"]` has to be `True` for this to work.
- `'face'`: Use the adjacent face color.
- An `mpl` color or sequence of colors will set the edge color.

The singular form `edgecolor` works as an alias.

`alpha` [scalar, optional, default: `None`] The alpha blending value of the face color, between 0 (transparent) and 1 (opaque). Note: The `edgecolor` is currently not affected by this.

`snap` [bool, optional, default: `False`] Whether to snap the mesh to pixel boundaries.

Returns

`collection` [`matplotlib.collections.Collection`]

Other Parameters

`antialiaseds` [bool, optional, default: `False`] The default `antialiaseds` is `False` if the default `edgecolors="none"` is used. This eliminates artificial lines at patch boundaries, and works regardless of the value of `alpha`. If `edgecolors` is not `"none"`, then the default `antialiaseds` is taken from `rcParams["patch.antialiased"]`, which defaults to `True`. Stroking the edges may be preferred if `alpha` is 1, but will cause artifacts otherwise.

****kwargs** : Additionally, the following arguments are allowed. They are passed along to the `PolyCollection` constructor:

	<table border="1"><tr><th>Property</th><th>Description</th></tr></table>	Property	Description
Property	Description		
<code>agg_filter</code>	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a		
<code>alpha</code>	float or <code>None</code>		
<code>animated</code>	bool		
<code>antialiased</code> or <code>antialiaseds</code>	Boolean or sequence of booleans		
<code>array</code>	ndarray		
<code>capstyle</code>	unknown		
<code>clim</code>	a length 2 sequence of floats; may be overridden in methods that have <code>vmin</code> and <code>vm</code>		
<code>clip_box</code>	a <code>Bbox</code> instance		
<code>clip_on</code>	bool		

Continued on next page

Table 23 – continued from previous page

	Property	Description
clip_path	[(Path, Transform) Patch None]	
cmap	a colormap or registered colormap name	
color	matplotlib color arg or sequence of rgba tuples	
contains	a callable function	
edgecolor or edgecolors	matplotlib color spec or sequence of specs	
facecolor or facecolors	matplotlib color spec or sequence of specs	
figure	a Figure instance	
gid	an id string	
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']	
joinstyle	unknown	
label	object	
linestyle or dashes or linestyles	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' '']	
linewidth or linewidths or lw	float or sequence of floats	
norm	Normalize	
offset_position	['screen' 'data']	
offsets	float or sequence of floats	
path_effects	AbstractPathEffect	
picker	[None bool float callable]	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
transform	Transform	
url	a url string	
urls	List[str] or None	
visible	bool	
zorder	float	

See also:

`pcolormesh` for an explanation of the differences between `pcolor` and `pcolormesh`.

`imshow` If X and Y are each equidistant, `imshow` can be a faster alternative.

Notes

Masked arrays

X, Y and C may be masked arrays. If either C[i, j], or one of the vertices surrounding C[i,j] (X or Y at [i, j], [i+1, j], [i, j+1], [i+1, j+1]) is masked, nothing is plotted.

Grid orientation

The grid orientation follows the standard matrix convention: An array C with shape (nrows, ncolumns) is plotted with the column number as X and the row number as Y.

Handling of `pcolor()` end-cases

`pcolor()` displays all columns of `C` if `X` and `Y` are not specified, or if `X` and `Y` have one more column than `C`. If `X` and `Y` have the same number of columns as `C` then the last column of `C` is dropped. Similarly for the rows.

Note: This behavior is different from MATLAB's `pcolor()`, which always discards the last row and column of `C`.

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All positional and all keyword arguments.
-

Examples using `matplotlib.axes.Axes.pcolor`

- `sphx_glr_gallery_images_contours_and_fields_pcolor_demo.py`

`matplotlib.axes.Axes.pcolorfast`

`Axes.pcolorfast(*args, data=None, **kwargs)`

Create a pseudocolor plot with a non-regular rectangular grid.

Call signatures:

```
ax.pcolorfast(C, **kwargs)
ax.pcolorfast(xr, yr, C, **kwargs)
ax.pcolorfast(x, y, C, **kwargs)
ax.pcolorfast(X, Y, C, **kwargs)
```

This method is similar to `~.Axes.pcolor` and `pcolormesh`. It's designed to provide the fastest `pcolor`-type plotting with the Agg backend. To achieve this, it uses different algorithms internally depending on the complexity of the input grid (regular rectangular, non-regular rectangular or arbitrary quadrilateral).

Warning: This method is experimental. Compared to `pcolor` or `pcolormesh` it has some limitations:

- It supports only flat shading (no outlines)
- It lacks support for log scaling of the axes.
- It does not have a `pyplot` wrapper.

Parameters

C [array-like(M, N)] A scalar 2D array. The values will be color-mapped. C may be a masked array.

x, y [tuple or array-like] X and Y are used to specify the coordinates of the quadrilaterals. There are different ways to do this:

- Use tuples `xr=(xmin, xmax)` and `yr=(ymin, ymax)` to define a uniform rectangular grid.

The tuples define the outer edges of the grid. All individual quadrilaterals will be of the same size. This is the fastest version.

- Use 1D arrays `x, y` to specify a non-uniform rectangular grid.

In this case `x` and `y` have to be monotonic 1D arrays of length `N+1` and `M+1`, specifying the `x` and `y` boundaries of the cells.

The speed is intermediate. Note: The grid is checked, and if found to be uniform the fast version is used.

- Use 2D arrays `X, Y` if you need an arbitrary quadrilateral grid (i.e. if the quadrilaterals are not rectangular).

In this case `X` and `Y` are 2D arrays with shape `(M, N)`, specifying the `x` and `y` coordinates of the corners of the colored quadrilaterals. See `pcolormesh` for details.

This is the most general, but the slowest to render. It may produce faster and more compact output using `ps`, `pdf`, and `svg` backends, however.

Leaving out `x` and `y` defaults to `xr=(0, N)`, `yr=(0, M)`.

cmap [str or Colormap, optional] A Colormap instance or registered colormap name. The colormap maps the `C` values to colors. Defaults to `rcParams["image.cmap"]`.

norm [Normalize, optional] The Normalize instance scales the data values to the canonical colormap range `[0, 1]` for mapping to colors. By default, the data range is mapped to the colorbar range using linear scaling.

vmin, vmax [scalar, optional, default: None] The colorbar range. If None, suitable min/max values are automatically chosen by the Normalize instance (defaults to the respective min/max values of `C` in case of the default linear scaling).

alpha [scalar, optional, default: None] The alpha blending value, between 0 (transparent) and 1 (opaque).

snap [bool, optional, default: False] Whether to snap the mesh to pixel boundaries.

Returns

image [AxesImage or PcolorImage or QuadMesh] The return type depends on the type of grid:

- AxesImage for a regular rectangular grid.

- PcolorImage for a non-regular rectangular grid.
- QuadMesh for a non-rectangular grid.

Notes

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All positional and all keyword arguments.
-

Examples using `matplotlib.axes.Axes.pcolorfast`

- `sphx_glr_gallery_images_contours_and_fields_pcolor_demo.py`

`matplotlib.axes.Axes.pcolormesh`

`Axes.pcolormesh(*args, data=None, **kwargs)`

Create a pseudocolor plot with a non-regular rectangular grid.

Call signature:

`pcolor([X, Y,] C, **kwargs)`

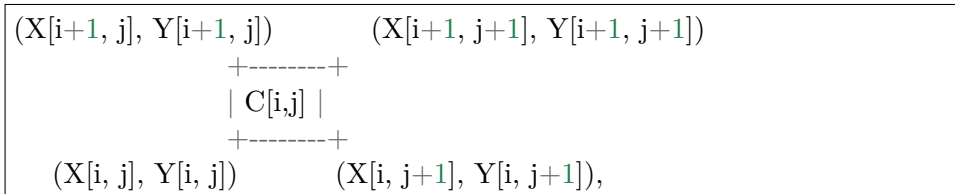
X and Y can be used to specify the corners of the quadrilaterals.

Note: `pcolormesh()` is similar to `pcolor()`. It's much faster and preferred in most cases. For a detailed discussion on the differences see [Differences between `pcolor\(\)` and `pcolormesh\(\)`](#).

Parameters

C [array_like] A scalar 2-D array. The values will be color-mapped.

X, Y [array_like, optional] The coordinates of the quadrilateral corners. The quadrilateral for `C[i,j]` has corners at:



Note that the column index corresponds to the x-coordinate, and the row index corresponds to y. For details, see the Notes section below.

The dimensions of X and Y should be one greater than those of C. Alternatively, X, Y and C may have equal dimensions, in which case the last row and column of C will be ignored.

If X and/or Y are 1-D arrays or column vectors they will be expanded as needed into the appropriate 2-D arrays, making a rectangular grid.

`cmap` [str or Colormap, optional] A Colormap instance or registered colormap name. The colormap maps the C values to colors. Defaults to `rcParams["image.cmap"]`.

`norm` [Normalize, optional] The Normalize instance scales the data values to the canonical colormap range [0, 1] for mapping to colors. By default, the data range is mapped to the colorbar range using linear scaling.

`vmin, vmax` [scalar, optional, default: None] The colorbar range. If None, suitable min/max values are automatically chosen by the Normalize instance (defaults to the respective min/max values of C in case of the default linear scaling).

`edgecolors` [{ 'none', None, 'face', color, color sequence }, optional] The color of the edges. Defaults to 'none'. Possible values:

- 'none' or '': No edge.
- None: `rcParams["patch.edgecolor"]` will be used. Note that currently `rcParams["patch.force_edgecolor"]` has to be True for this to work.
- 'face': Use the adjacent face color.
- An mpl color or sequence of colors will set the edge color.

The singular form `edgecolor` works as an alias.

`alpha` [scalar, optional, default: None] The alpha blending value, between 0 (transparent) and 1 (opaque).

`shading` [{ 'flat', 'gouraud' }, optional] The fill style, Possible values:

- 'flat': A solid color is used for each quad. The color of the quad (i, j), (i+1, j), (i, j+1), (i+1, j+1) is given by `C[i,j]`.
- 'gouraud': Each quad will be Gouraud shaded: The color of the corners (i', j') are given by `C[i',j']`. The color values of the area in between is interpolated from the corner values. When Gouraud shading is used, `edgecolors` is ignored.

`snap` [bool, optional, default: False] Whether to snap the mesh to pixel boundaries.

Returns

`mesh` [matplotlib.collections.QuadMesh]

Other Parameters

****kwargs** Additionally, the following arguments are allowed. They are passed along to the QuadMesh constructor:

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a
alpha		float or None
animated		bool
antialiased or antialiaseds		Boolean or sequence of booleans
array		ndarray
capstyle		unknown
clim		a length 2 sequence of floats; may be overridden in methods that have vmin and vm
clip_box		a Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
cmap		a colormap or registered colormap name
color		matplotlib color arg or sequence of rgba tuples
contains		a callable function
edgecolor or edgecolors		matplotlib color spec or sequence of specs
facecolor or facecolors		matplotlib color spec or sequence of specs
figure		a Figure instance
gid		an id string
hatch		['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle		unknown
label		object
linestyle or dashes or linestyles		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' '']
linewidth or linewidths or lw		float or sequence of floats
norm		Normalize
offset_position		['screen' 'data']
offsets		float or sequence of floats
path_effects		AbstractPathEffect
picker		[None bool float callable]
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
transform		Transform
url		a url string
urls		List[str] or None
visible		bool
zorder		float

See also:

pcolor An alternative implementation with slightly different features. For a detailed discus-

sion on the differences see [Differences between pcolor\(\) and pcolormesh\(\)](#).

`imshow` If `X` and `Y` are each equidistant, `imshow` can be a faster alternative.

Notes

Masked arrays

`C` may be a masked array. If `C[i, j]` is masked, the corresponding quadrilateral will be transparent. Masking of `X` and `Y` is not supported. Use `pcolor` if you need this functionality.

Grid orientation

The grid orientation follows the standard matrix convention: An array `C` with shape `(nrows, ncolumns)` is plotted with the column number as `X` and the row number as `Y`.

Differences between pcolor() and pcolormesh()

Both methods are used to create a pseudocolor plot of a 2-D array using quadrilaterals.

The main difference lies in the created object and internal data handling: While `pcolor` returns a `PolyCollection`, `pcolormesh` returns a `QuadMesh`. The latter is more specialized for the given purpose and thus is faster. It should almost always be preferred.

There is also a slight difference in the handling of masked arrays. Both `pcolor` and `pcolormesh` support masked arrays for `C`. However, only `pcolor` supports masked arrays for `X` and `Y`. The reason lies in the internal handling of the masked values. `pcolor` leaves out the respective polygons from the `PolyCollection`. `pcolormesh` sets the `facecolor` of the masked elements to transparent. You can see the difference when using `edgecolors`. While all edges are drawn irrespective of masking in a `QuadMesh`, the edge between two adjacent masked quadrilaterals in `pcolor` is not drawn as the corresponding polygons do not exist in the `PolyCollection`.

Another difference is the support of Gouraud shading in `pcolormesh`, which is not available with `pcolor`.

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All positional and all keyword arguments.
-

Examples using `matplotlib.axes.Axes.pcolormesh`

- `sphx_glr_gallery_images_contours_and_fields_quadmesh_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_pcolormesh_levels.py`
- `sphx_glr_gallery_images_contours_and_fields_pcolor_demo.py`

matplotlib.axes.Axes.spy

Axes.spy(Z, precision=0, marker=None, markersize=None, aspect='equal', origin='upper',
**kwargs)

Plot the sparsity pattern on a 2-D array.

spy(Z) plots the sparsity pattern of the 2-D array Z.

Parameters

Z [sparse array (n, m)] The array to be plotted.

precision [float, optional, default: 0] If precision is 0, any non-zero value will be plotted; else, values of $|Z| > \textit{precision}$ will be plotted.

For `scipy.sparse.spmatrix` instances, there is a special case: if precision is 'present', any value present in the array will be plotted, even if it is identically zero.

origin [['upper', 'lower'], optional, default: "upper"] Place the [0,0] index of the array in the upper left or lower left corner of the axes.

aspect [['auto' | 'equal' | scalar], optional, default: "equal"] If 'equal', and extent is None, changes the axes aspect ratio to match that of the image. If extent is not `None`, the axes aspect ratio is changed to match that of the extent.

If 'auto', changes the image aspect ratio to match that of the axes.

If None, default to rc image.aspect value.

Two plotting styles are available: image or marker. Both

are available for full arrays, but only the marker style

works for `:class:'scipy.sparse.spmatrix'` instances.

If `*marker*` and `*markersize*` are `*None*`, an image will be

returned and any remaining kwargs are passed to

`:func:'~matplotlib.pyplot.imshow'`; else, a

`:class:'~matplotlib.lines.Line2D'` object will be returned with

the value of marker determining the marker type, and any

remaining kwargs passed to the

`:meth:'~matplotlib.axes.Axes.plot'` method.

If `*marker*` and `*markersize*` are `*None*`, useful kwargs include:

`* *cmap*`

`* *alpha*`

See also:

`imshow` for image options.
`plot` for plotting options

Examples using `matplotlib.axes.Axes.spy`

- `sphx_glr_gallery_images_contours_and_fields_spy_demos.py`

35.1.8 Unstructured Triangles

<code>Axes.tripcolor</code>	Create a pseudocolor plot of an unstructured triangular grid.
<code>Axes.triplot</code>	Draw a unstructured triangular grid as lines and/or markers.
<code>Axes.tricontour</code>	Draw contours on an unstructured triangular grid.
<code>Axes.tricontourf</code>	Draw contours on an unstructured triangular grid.

`matplotlib.axes.Axes.tripcolor`

`Axes.tripcolor(*args, **kwargs)`

Create a pseudocolor plot of an unstructured triangular grid.

The triangulation can be specified in one of two ways; either:

```
tripcolor(triangulation, ...)
```

where `triangulation` is a `matplotlib.tri.Triangulation` object, or

```
tripcolor(x, y, ...)
tripcolor(x, y, triangles, ...)
tripcolor(x, y, triangles=triangles, ...)
tripcolor(x, y, mask=mask, ...)
tripcolor(x, y, triangles, mask=mask, ...)
```

in which case a `Triangulation` object will be created. See `Triangulation` for a explanation of these possibilities.

The next argument must be `C`, the array of color values, either one per point in the triangulation if color values are defined at points, or one per triangle in the triangulation if color values are defined at triangles. If there are the same number of points and triangles in the triangulation it is assumed that color values are defined at points; to force the use of color values at triangles use the kwarg `facecolors=C` instead of just `C`.

shading may be `'flat'` (the default) or `'gouraud'`. If shading is `'flat'` and `C` values are defined at points, the color values used for each triangle are from the mean `C` of the triangle's three

points. If shading is 'gouraud' then color values must be defined at points.

The remaining kwargs are the same as for `pcolor()`.

Examples using `matplotlib.axes.Axes.tripcolor`

- `sphx_glr_gallery_images_contours_and_fields_tripcolor_demo.py`

`matplotlib.axes.Axes.triplot`

`Axes.triplot(*args, **kwargs)`

Draw a unstructured triangular grid as lines and/or markers.

The triangulation to plot can be specified in one of two ways; either:

```
triplot(triangulation, ...)
```

where `triangulation` is a `matplotlib.tri.Triangulation` object, or

```
triplot(x, y, ...)
triplot(x, y, triangles, ...)
triplot(x, y, triangles=triangles, ...)
triplot(x, y, mask=mask, ...)
triplot(x, y, triangles, mask=mask, ...)
```

in which case a `Triangulation` object will be created. See `Triangulation` for a explanation of these possibilities.

The remaining args and kwargs are the same as for `plot()`.

Return a list of 2 `Line2D` containing respectively:

- the lines plotted for triangles edges
- the markers plotted for triangles nodes

Examples using `matplotlib.axes.Axes.triplot`

- `sphx_glr_gallery_images_contours_and_fields_triinterp_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_trigradient_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_triplot_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_tricontour_smooth_delaunay.py`

matplotlib.axes.Axes.tricontour

Axes.tricontour(*args, **kwargs)

Draw contours on an unstructured triangular grid. `tricontour()` and `tricontourf()` draw contour lines and filled contours, respectively. Except as noted, function signatures and return values are the same for both versions.

The triangulation can be specified in one of two ways; either:

```
tricontour(triangulation, ...)
```

where `triangulation` is a `matplotlib.tri.Triangulation` object, or

```
tricontour(x, y, ...)
tricontour(x, y, triangles, ...)
tricontour(x, y, triangles=triangles, ...)
tricontour(x, y, mask=mask, ...)
tricontour(x, y, triangles, mask=mask, ...)
```

in which case a `Triangulation` object will be created. See `Triangulation` for a explanation of these possibilities.

The remaining arguments may be:

```
tricontour(..., Z)
```

where `Z` is the array of values to contour, one per point in the triangulation. The level values are chosen automatically.

```
tricontour(..., Z, N)
```

contour up to `N+1` automatically chosen contour levels (`N` intervals).

```
tricontour(..., Z, V)
```

draw contour lines at the values specified in sequence `V`, which must be in increasing order.

```
tricontourf(..., Z, V)
```

fill the $(\text{len}(V)-1)$ regions between the values in `V`, which must be in increasing order.

```
tricontour(Z, **kwargs)
```

Use keyword args to control colors, linewidth, origin, `cmap` ... see below for more details.

`C = tricontour(...)` returns a `TriContourSet` object.

Optional keyword arguments:

colors: [None | string | (mpl_colors)] If None, the colormap specified by `cmap` will be used.

If a string, like 'r' or 'red', all levels will be plotted in this color.

If a tuple of matplotlib color args (string, float, rgb, etc), different levels will be plotted in different colors in the order specified.

alpha: float The alpha blending value

cmap: [None | Colormap] A cm Colormap instance or None. If cmap is None and colors is None, a default Colormap is used.

norm: [None | Normalize] A matplotlib.colors.Normalize instance for scaling data values to colors. If norm is None and colors is None, the default linear scaling is used.

levels [level0, level1, ..., leveln] A list of floating point numbers indicating the level curves to draw, in increasing order; e.g., to draw just the zero contour pass levels=[0]

origin: [None | 'upper' | 'lower' | 'image'] If None, the first value of Z will correspond to the lower left corner, location (0,0). If 'image', the rc value for image.origin will be used.

This keyword is not active if X and Y are specified in the call to contour.

extent: [None | (x0,x1,y0,y1)]

If origin is not None, then extent is interpreted as in matplotlib.pyplot.imshow(): it gives the outer pixel boundaries. In this case, the position of Z[0,0] is the center of the pixel, not a corner. If origin is None, then (x0, y0) is the position of Z[0,0], and (x1, y1) is the position of Z[-1,-1].

This keyword is not active if X and Y are specified in the call to contour.

locator: [None | ticker.Locator subclass] If locator is None, the default MaxNLocator is used. The locator is used to determine the contour levels if they are not given explicitly via the V argument.

extend: ['neither' | 'both' | 'min' | 'max'] Unless this is 'neither', contour levels are automatically added to one or both ends of the range so that all data are included. These added ranges are then mapped to the special colormap values which default to the ends of the colormap range, but can be set via matplotlib.colors.Colormap.set_under() and matplotlib.colors.Colormap.set_over() methods.

xunits, yunits: [None | registered units] Override axis units by specifying an instance of a matplotlib.units.ConversionInterface.

tricontour-only keyword arguments:

linewidths: [None | number | tuple of numbers] If linewidths is None, the default width in lines.linewidth in matplotlibrc is used.

If a number, all levels will be plotted with this linewidth.

If a tuple, different levels will be plotted with different linewidths in the order specified

linestyles: [None | 'solid' | 'dashed' | 'dashdot' | 'dotted'] If linestyles is None, the 'solid' is used.

linestyles can also be an iterable of the above strings specifying a set of linestyles to be used. If this iterable is shorter than the number of contour levels it will be repeated as necessary.

If contour is using a monochrome colormap and the contour level is less than 0, then the linestyle specified in `contour.negative_linestyle` in `matplotlibrc` will be used.

tricontourf-only keyword arguments:

`antialiased`: bool enable antialiasing

Note: `tricontourf` fills intervals that are closed at the top; that is, for boundaries `z1` and `z2`, the filled region is:

$$z1 < z \leq z2$$

There is one exception: if the lowest boundary coincides with the minimum value of the `z` array, then that minimum value will be included in the lowest interval.

Examples using `matplotlib.axes.Axes.tricontour`

- `sphx_glr_gallery_images_contours_and_fields_irregulardatagrid.py`
- `sphx_glr_gallery_images_contours_and_fields_tricontour_smooth_user.py`
- `sphx_glr_gallery_images_contours_and_fields_trigradient_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_tricontour_smooth_delaunay.py`

`matplotlib.axes.Axes.tricontourf`

`Axes.tricontourf(*args, **kwargs)`

Draw contours on an unstructured triangular grid. `tricontour()` and `tricontourf()` draw contour lines and filled contours, respectively. Except as noted, function signatures and return values are the same for both versions.

The triangulation can be specified in one of two ways; either:

```
tricontour(triangulation, ...)
```

where `triangulation` is a `matplotlib.tri.Triangulation` object, or

```
tricontour(x, y, ...)
tricontour(x, y, triangles, ...)
tricontour(x, y, triangles=triangles, ...)
tricontour(x, y, mask=mask, ...)
tricontour(x, y, triangles, mask=mask, ...)
```

in which case a `Triangulation` object will be created. See `Triangulation` for a explanation of these possibilities.

The remaining arguments may be:

```
tricontour(..., Z)
```

where Z is the array of values to contour, one per point in the triangulation. The level values are chosen automatically.

```
tricontour(..., Z, N)
```

contour up to N+1 automatically chosen contour levels (N intervals).

```
tricontour(..., Z, V)
```

draw contour lines at the values specified in sequence V, which must be in increasing order.

```
tricontourf(..., Z, V)
```

fill the (len(V)-1) regions between the values in V, which must be in increasing order.

```
tricontour(Z, **kwargs)
```

Use keyword args to control colors, linewidth, origin, cmap ... see below for more details.

C = tricontour(...) returns a TriContourSet object.

Optional keyword arguments:

colors: [None | string | (mpl_colors)] If None, the colormap specified by cmap will be used.

If a string, like 'r' or 'red', all levels will be plotted in this color.

If a tuple of matplotlib color args (string, float, rgb, etc), different levels will be plotted in different colors in the order specified.

alpha: float The alpha blending value

cmap: [None | Colormap] A cm Colormap instance or None. If cmap is None and colors is None, a default Colormap is used.

norm: [None | Normalize] A matplotlib.colors.Normalize instance for scaling data values to colors. If norm is None and colors is None, the default linear scaling is used.

levels [level0, level1, ..., leveln] A list of floating point numbers indicating the level curves to draw, in increasing order; e.g., to draw just the zero contour pass levels=[0]

origin: [None | 'upper' | 'lower' | 'image'] If None, the first value of Z will correspond to the lower left corner, location (0,0). If 'image', the rc value for image.origin will be used.

This keyword is not active if X and Y are specified in the call to contour.

extent: [None | (x0,x1,y0,y1)]

If origin is not None, then extent is interpreted as in `matplotlib.pyplot.imshow()`: it gives the outer pixel boundaries. In this case, the position of `Z[0,0]` is the center of the pixel, not a corner. If origin is None, then `(x0, y0)` is the position of `Z[0,0]`, and `(x1, y1)` is the position of `Z[-1,-1]`.

This keyword is not active if X and Y are specified in the call to `contour`.

`locator`: [None | `ticker.Locator` subclass] If `locator` is None, the default `MaxNLocator` is used. The locator is used to determine the contour levels if they are not given explicitly via the `V` argument.

`extend`: ['neither' | 'both' | 'min' | 'max'] Unless this is 'neither', contour levels are automatically added to one or both ends of the range so that all data are included. These added ranges are then mapped to the special colormap values which default to the ends of the colormap range, but can be set via `matplotlib.colors.Colormap.set_under()` and `matplotlib.colors.Colormap.set_over()` methods.

`xunits, yunits`: [None | registered units] Override axis units by specifying an instance of a `matplotlib.units.ConversionInterface`.

tricontour-only keyword arguments:

`linewidths`: [None | number | tuple of numbers] If `linewidths` is None, the default width in `lines.linewidth` in `matplotlibrc` is used.

If a number, all levels will be plotted with this linewidth.

If a tuple, different levels will be plotted with different linewidths in the order specified

`linestyles`: [None | 'solid' | 'dashed' | 'dashdot' | 'dotted'] If `linestyles` is None, the 'solid' is used.

`linestyles` can also be an iterable of the above strings specifying a set of linestyles to be used. If this iterable is shorter than the number of contour levels it will be repeated as necessary.

If `contour` is using a monochrome colormap and the contour level is less than 0, then the linestyle specified in `contour.negative_linestyle` in `matplotlibrc` will be used.

tricontourf-only keyword arguments:

`antialiased`: bool enable antialiasing

Note: `tricontourf` fills intervals that are closed at the top; that is, for boundaries `z1` and `z2`, the filled region is:

$$z1 < z \leq z2$$

There is one exception: if the lowest boundary coincides with the minimum value of the `z` array, then that minimum value will be included in the lowest interval.

Examples using `matplotlib.axes.Axes.tricontourf`

- `sphx_glr_gallery_images_contours_and_fields_irregulardatagrid.py`
- `sphx_glr_gallery_images_contours_and_fields_triinterp_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_tricontour_smooth_user.py`
- `sphx_glr_gallery_images_contours_and_fields_tricontour_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_tricontour_smooth_delaunay.py`

35.1.9 Text and Annotations

<code>Axes.annotate</code>	Annotate the point <code>xy</code> with text <code>s</code> .
<code>Axes.text</code>	Add text to the axes.
<code>Axes.table</code>	Add a table to the current axes.
<code>Axes.arrow</code>	Add an arrow to the axes.

`matplotlib.axes.Axes.annotate``Axes.annotate(*args, **kwargs)`Annotate the point `xy` with text `s`.Additional `kwargs` are passed to `Text`.

Parameters

`s` [`str`] The text of the annotation`xy` [`iterable`] Length 2 sequence specifying the (x,y) point to annotate`xytext` [`iterable`, optional] Length 2 sequence specifying the (x,y) to place the text at. If `None`, defaults to `xy`.`xycoords` [`str`, `Artist`, `Transform`, callable or tuple, optional] The coordinate system that `xy` is given in.For a `str` the allowed values are:

Property	Description
'figure points'	points from the lower left of the figure
'figure pixels'	pixels from the lower left of the figure
'figure fraction'	fraction of figure from lower left
'axes points'	points from lower left corner of axes
'axes pixels'	pixels from lower left corner of axes
'axes fraction'	fraction of axes from lower left
'data'	use the coordinate system of the object being annotated (default)
'polar'	(theta,r) if not native 'data' coordinates

If a Artist object is passed in the units are fraction if it's bounding box.

If a Transform object is passed in use that to transform xy to screen coordinates

If a callable it must take a RendererBase object as input and return a Transform or Bbox object

If a [tuple](#) must be length 2 tuple of str, Artist, Transform or callable objects. The first transform is used for the x coordinate and the second for y.

See [Advanced Annotation](#) for more details.

Defaults to 'data'

`textcoords` [str, Artist, Transform, callable or tuple, optional] The coordinate system that xytext is given, which may be different than the coordinate system used for xy.

All xycoords values are valid as well as the following strings:

Property	Description
'offset points'	offset (in points) from the xy value
'offset pixels'	offset (in pixels) from the xy value

defaults to the input of xycoords

`arrowprops` [dict, optional] If not None, properties used to draw a FancyArrowPatch arrow between xy and xytext.

If arrowprops does not contain the key 'arrowstyle' the allowed keys are:

Key	Description
width	the width of the arrow in points
headwidth	the width of the base of the arrow head in points
headlength	the length of the arrow head in points
shrink	fraction of total length to 'shrink' from both ends
?	any key to matplotlib.patches.FancyArrowPatch

If the arrowprops contains the key 'arrowstyle' the above keys are forbidden.
The allowed values of 'arrowstyle' are:

Name	Attrs
'-'	None
'->'	head_length=0.4,head_width=0.2
'-['	widthB=1.0,lengthB=0.2,angleB=None
' -'	widthA=1.0,widthB=1.0
'-> '	head_length=0.4,head_width=0.2
'<-'	head_length=0.4,head_width=0.2
'<->'	head_length=0.4,head_width=0.2
'< -'	head_length=0.4,head_width=0.2
'< ->'	head_length=0.4,head_width=0.2
'fancy'	head_length=0.4,head_width=0.4,tail_width=0.4
'simple'	head_length=0.5,head_width=0.5,tail_width=0.2
'wedge'	tail_width=0.3,shrink_factor=0.5

Valid keys for FancyArrowPatch are:

Key	Description
arrowstyle	the arrow style
connectionstyle	the connection style
relpos	default is (0.5, 0.5)
patchA	default is bounding box of the text
patchB	default is None
shrinkA	default is 2 points
shrinkB	default is 2 points
mutation_scale	default is text size (in points)
mutation_aspect	default is 1.
?	any key for matplotlib.patches.PathPatch

Defaults to None

annotation_clip [bool, optional] Controls the visibility of the annotation when it goes outside the axes area.

If **True**, the annotation will only be drawn when the xy is inside the axes.
If **False**, the annotation will always be drawn regardless of its position.

The default is **None**, which behave as **True** only if xycoords is "data".

Returns

Annotation

Examples using `matplotlib.axes.Axes.annotate`

- `sphx_glr_gallery_shapes_and_collections_donut.py`

`matplotlib.axes.Axes.text`

`Axes.text(x, y, s, fontdict=None, withdash=False, **kwargs)`

Add text to the axes.

Add the text `s` to the axes at location `x, y` in data coordinates.

Parameters

`x, y` [scalars] The position to place the text. By default, this is in data coordinates. The coordinate system can be changed using the `transform` parameter.

`s` [str] The text.

`fontdict` [dictionary, optional, default: None] A dictionary to override the default text properties. If `fontdict` is None, the defaults are determined by your rc parameters.

`withdash` [boolean, optional, default: False] Creates a `TextWithDash` instance instead of a `Text` instance.

Returns

`text` [Text] The created `Text` instance.

Other Parameters

`**kwargs` [Text properties.] Other miscellaneous text parameters.

Examples

Individual keyword arguments can be used to override any given parameter:

```
>>> text(x, y, s, fontsize=12)
```

The default transform specifies that text is in data coords, alternatively, you can specify text in axis coords (0,0 is lower-left and 1,1 is upper-right). The example below places text in the center of the axes:

```
>>> text(0.5, 0.5, 'matplotlib', horizontalalignment='center',
...      verticalalignment='center', transform=ax.transAxes)
```

You can put a rectangular box around the text instance (e.g., to set a background color) by using the keyword `bbox`. `bbox` is a dictionary of `Rectangle` properties. For example:

```
>>> text(x, y, s, bbox=dict(facecolor='red', alpha=0.5))
```

Examples using `matplotlib.axes.Axes.text`

- `sphinx_glr_gallery_color_named_colors.py`

`matplotlib.axes.Axes.table`

`Axes.table(**kwargs)`

Add a table to the current axes.

Call signature:

```
table(cellText=None, cellColours=None,  
      cellLoc='right', colWidths=None,  
      rowLabels=None, rowColours=None, rowLoc='left',  
      colLabels=None, colColours=None, colLoc='center',  
      loc='bottom', bbox=None)
```

Returns a `matplotlib.table.Table` instance. Either `cellText` or `cellColours` must be provided. For finer grained control over tables, use the `Table` class and add it to the axes with `add_table()`.

Thanks to John Gill for providing the class and table.

`kwargs` control the `Table` properties:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float (0.0 transparent through 1.0 opaque)
animated	bool
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
contains	a callable function
figure	a Figure instance
fontsize	a float in points
gid	an id string
label	object
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

matplotlib.axes.Axes.arrow

`Axes.arrow(x, y, dx, dy, **kwargs)`

Add an arrow to the axes.

This draws an arrow from (x, y) to (x+dx, y+dy).

Parameters

x, y [float] The x/y-coordinate of the arrow base.

dx, dy [float] The length of the arrow along x/y-direction.

Returns

arrow [FancyArrow] The created FancyArrow object.

Other Parameters

****kwargs** Optional kwargs (inherited from FancyArrow patch) control the arrow construction and properties:

Constructor arguments

width: float (default: 0.001) width of full arrow tail

length_includes_head: bool (default: False) True if head is to be counted in calculating the length.

`head_width`: float or None (default: $3 * \text{width}$) total width of the full arrow head

`head_length`: float or None (default: $1.5 * \text{head_width}$) length of arrow head

`shape`: ['full', 'left', 'right'] (default: 'full') draw the left-half, right-half, or full arrow

`overhang`: float (default: 0) fraction that the arrow is swept back (0 overhang means triangular shape). Can be negative or greater than one.

`head_starts_at_zero`: bool (default: False) if True, the head starts being drawn at coordinate 0 instead of ending at coordinate 0.

Other valid kwargs (inherited from :class:'Patch') are:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

Notes

The resulting arrow is affected by the axes aspect ratio and limits. This may produce an arrow whose head is not square with its stem. To create an arrow whose head is square with its stem, use `annotate()` for example:


```
>>> ax.annotate("", xy=(0.5, 0.5), xytext=(0, 0),
...               arrowprops=dict(arrowstyle="->"))
```

35.1.10 Fields

<code>Axes.barbs</code>	Plot a 2-D field of barbs.
<code>Axes.quiver</code>	Plot a 2-D field of arrows.
<code>Axes.quiverkey</code>	Add a key to a quiver plot.
<code>Axes.streamplot</code>	Draws streamlines of a vector flow.

`matplotlib.axes.Axes.barbs`

`Axes.barbs(*args, data=None, **kw)`

Plot a 2-D field of barbs.

Call signatures:

```
barb(U, V, **kw)
barb(U, V, C, **kw)
barb(X, Y, U, V, **kw)
barb(X, Y, U, V, C, **kw)
```

Arguments:

X, Y: The x and y coordinates of the barb locations (default is head of barb; see pivot kwarg)

U, V: Give the x and y components of the barb shaft

C: An optional array used to map colors to the barbs

All arguments may be 1-D or 2-D arrays or sequences. If X and Y are absent, they will be generated as a uniform grid. If U and V are 2-D arrays but X and Y are 1-D, and if `len(X)` and `len(Y)` match the column and row dimensions of U, then X and Y will be expanded with `numpy.meshgrid()`.

U, V, C may be masked arrays, but masked X, Y are not supported at present.

Keyword arguments:

length: Length of the barb in points; the other parts of the barb are scaled against this. Default is 7.

pivot: ['tip' | 'middle' | float] The part of the arrow that is at the grid point; the arrow rotates about this point, hence the name pivot. Default is 'tip'. Can also be a number, which shifts the start of the barb that many points from the origin.

barbcolor: [color | color sequence] Specifies the color all parts of the barb except any flags. This parameter is analogous to the `edgecolor` parameter for polygons,

which can be used instead. However this parameter will override facecolor.

flagcolor: [color | color sequence] Specifies the color of any flags on the barb. This parameter is analogous to the facecolor parameter for polygons, which can be used instead. However this parameter will override facecolor. If this is not set (and C has not either) then flagcolor will be set to match barbcolor so that the barb has a uniform color. If C has been set, flagcolor has no effect.

sizes: A dictionary of coefficients specifying the ratio of a given feature to the length of the barb. Only those values one wishes to override need to be included. These features include:

- 'spacing' - space between features (flags, full/half barbs)
- 'height' - height (distance from shaft to top) of a flag or full barb
- 'width' - width of a flag, twice the width of a full barb
- 'emptybarb' - radius of the circle used for low magnitudes

fill_empty: A flag on whether the empty barbs (circles) that are drawn should be filled with the flag color. If they are not filled, they will be drawn such that no color is applied to the center. Default is False

rounding: A flag to indicate whether the vector magnitude should be rounded when allocating barb components. If True, the magnitude is rounded to the nearest multiple of the half-barb increment. If False, the magnitude is simply truncated to the next lowest multiple. Default is True

barb_increments: A dictionary of increments specifying values to associate with different parts of the barb. Only those values one wishes to override need to be included.

- 'half' - half barbs (Default is 5)
- 'full' - full barbs (Default is 10)
- 'flag' - flags (default is 50)

flip_barb: Either a single boolean flag or an array of booleans. Single boolean indicates whether the lines and flags should point opposite to normal for all barbs. An array (which should be the same size as the other data arrays) indicates whether to flip for each individual barb. Normal behavior is for the barbs and lines to point right (comes from wind barbs having these features point towards low pressure in the Northern Hemisphere.) Default is False

Barbs are traditionally used in meteorology as a way to plot the speed and direction of wind observations, but can technically be used to plot any two dimensional vector quantity. As opposed to arrows, which give vector magnitude by the length of the arrow, the barbs give more quantitative information about the vector magnitude by putting slanted lines or a triangle for various increments in magnitude, as show schematically below:



(continues on next page)

(continued from previous page)



The largest increment is given by a triangle (or "flag"). After those come full lines (barbs). The smallest increment is a half line. There is only, of course, ever at most 1 half line. If the magnitude is small and only needs a single half-line and no full lines or triangles, the half-line is offset from the end of the barb so that it can be easily distinguished from barbs with a single full line. The magnitude for the barb shown above would nominally be 65, using the standard increments of 50, 10, and 5.

linewidths and edgecolors can be used to customize the barb. Additional PolyCollection keyword arguments:

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (
alpha		float or None
animated		bool
antialiased or antialiaseds		Boolean or sequence of booleans
array		ndarray
capstyle		unknown
clim		a length 2 sequence of floats; may be overridden in methods that have vmin and vm
clip_box		a Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
cmap		a colormap or registered colormap name
color		matplotlib color arg or sequence of rgba tuples
contains		a callable function
edgecolor or edgecolors		matplotlib color spec or sequence of specs
facecolor or facecolors		matplotlib color spec or sequence of specs
figure		a Figure instance
gid		an id string
hatch		['/' '\ ' ' ' '- ' '+ ' 'x' 'o' 'O' '?' '*']
joinstyle		unknown
label		object
linestyle or dashes or linestyles		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ']
linewidth or linewidths or lw		float or sequence of floats
norm		Normalize
offset_position		['screen' 'data']
offsets		float or sequence of floats
path_effects		AbstractPathEffect
picker		[None bool float callable]
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)

Continued on next page

Table 28 – continued from previous page

	Property	Description
snap		bool or None
transform		Transform
url		a url string
urls		List[str] or None
visible		bool
zorder		float

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All positional and all keyword arguments.

Examples using `matplotlib.axes.Axes.barbs`

- `sphx_glr_gallery_images_contours_and_fields_barb_demo.py`

`matplotlib.axes.Axes.quiver`

`Axes.quiver(*args, data=None, **kw)`

Plot a 2-D field of arrows.

Call signatures:

```
quiver(U, V, **kw)
quiver(U, V, C, **kw)
quiver(X, Y, U, V, **kw)
quiver(X, Y, U, V, C, **kw)
```

`U` and `V` are the arrow data, `X` and `Y` set the location of the arrows, and `C` sets the color of the arrows. These arguments may be 1-D or 2-D arrays or sequences.

If `X` and `Y` are absent, they will be generated as a uniform grid. If `U` and `V` are 2-D arrays and `X` and `Y` are 1-D, and if `len(X)` and `len(Y)` match the column and row dimensions of `U`, then `X` and `Y` will be expanded with `numpy.meshgrid()`.

The default settings auto-scales the length of the arrows to a reasonable size. To change this behavior see the `scale` and `scale_units` kwargs.

The defaults give a slightly swept-back arrow; to make the head a triangle, make `headaxlength` the same as `headlength`. To make the arrow more pointed, reduce `headwidth` or increase `headlength` and `headaxislength`. To make the head smaller relative to the shaft, scale down all the head parameters. You will probably do best to leave `minshaft` alone.

linewidths and edgecolors can be used to customize the arrow outlines.

Parameters

- X [1D or 2D array, sequence, optional] The x coordinates of the arrow locations
- Y [1D or 2D array, sequence, optional] The y coordinates of the arrow locations
- U [1D or 2D array or masked array, sequence] The x components of the arrow vectors
- V [1D or 2D array or masked array, sequence] The y components of the arrow vectors

C [1D or 2D array, sequence, optional] The arrow colors

units [['width' | 'height' | 'dots' | 'inches' | 'x' | 'y' | 'xy']] The arrow dimensions (except for length) are measured in multiples of this unit.

'width' or 'height': the width or height of the axis

'dots' or 'inches': pixels or inches, based on the figure dpi

'x', 'y', or 'xy': respectively X, Y, or $\sqrt{X^2 + Y^2}$ in data units

The arrows scale differently depending on the units. For 'x' or 'y', the arrows get larger as one zooms in; for other units, the arrow size is independent of the zoom state. For 'width' or 'height', the arrow size increases with the width and height of the axes, respectively, when the window is resized; for 'dots' or 'inches', resizing does not change the arrows.

angles [['uv' | 'xy'], array, optional] Method for determining the angle of the arrows. Default is 'uv'.

'uv': the arrow axis aspect ratio is 1 so that if $U==V$ the orientation of the arrow on the plot is 45 degrees counter-clockwise from the horizontal axis (positive to the right).

'xy': arrows point from (x,y) to (x+u, y+v). Use this for plotting a gradient field, for example.

Alternatively, arbitrary angles may be specified as an array of values in degrees, counter-clockwise from the horizontal axis.

Note: inverting a data axis will correspondingly invert the arrows only with `angles='xy'`.

scale [None, float, optional] Number of data units per arrow length unit, e.g., m/s per plot width; a smaller scale parameter makes the arrow longer. Default is None.

If None, a simple autoscaling algorithm is used, based on the average vector length and the number of vectors. The arrow length unit is given by the `scale_units` parameter

`scale_units` [['width' | 'height' | 'dots' | 'inches' | 'x' | 'y' | 'xy'], None, optional] If the `scale` kwarg is None, the arrow length unit. Default is None.

e.g. `scale_units` is 'inches', `scale` is 2.0, and $(u,v) = (1,0)$, then the vector will be 0.5 inches long.

If `scale_units` is 'width'/'height', then the vector will be half the width/height of the axes.

If `scale_units` is 'x' then the vector will be 0.5 x-axis units. To plot vectors in the x-y plane, with `u` and `v` having the same units as `x` and `y`, use `angles='xy'`, `scale_units='xy'`, `scale=1`.

`width` [scalar, optional] Shaft width in arrow units; default depends on choice of units, above, and number of vectors; a typical starting value is about 0.005 times the width of the plot.

`headwidth` [scalar, optional] Head width as multiple of shaft width, default is 3

`headlength` [scalar, optional] Head length as multiple of shaft width, default is 5

`headaxislength` [scalar, optional] Head length at shaft intersection, default is 4.5

`minshaft` [scalar, optional] Length below which arrow scales, in units of head length. Do not set this to less than 1, or small arrows will look terrible! Default is 1

`minlength` [scalar, optional] Minimum length as a multiple of shaft width; if an arrow length is less than this, plot a dot (hexagon) of this diameter instead. Default is 1.

`pivot` [['tail' | 'mid' | 'middle' | 'tip'], optional] The part of the arrow that is at the grid point; the arrow rotates about this point, hence the name `pivot`.

`color` [[color | color sequence], optional] This is a synonym for the `PolyCollection` `facecolor` kwarg. If `C` has been set, `color` has no effect.

See also:

`quiverkey` Add a key to a quiver plot

Notes

Additional `PolyCollection` keyword arguments:

	Property	Description
<code>agg_filter</code>		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (
<code>alpha</code>		float or None

Continued on next page

Table 29 – continued from previous page

	Property	Description
animated	bool	
antialiased or antialiaseds	Boolean or sequence of booleans	
array	ndarray	
capstyle	unknown	
clim	a length 2 sequence of floats; may be overridden in methods that have vmin and vmax	
clip_box	a Bbox instance	
clip_on	bool	
clip_path	[(Path, Transform) Patch None]	
cmap	a colormap or registered colormap name	
color	matplotlib color arg or sequence of rgba tuples	
contains	a callable function	
edgecolor or edgecolors	matplotlib color spec or sequence of specs	
facecolor or facecolors	matplotlib color spec or sequence of specs	
figure	a Figure instance	
gid	an id string	
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']	
joinstyle	unknown	
label	object	
linestyle or dashes or linestyles	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' '']	
linewidth or linewidths or lw	float or sequence of floats	
norm	Normalize	
offset_position	['screen' 'data']	
offsets	float or sequence of floats	
path_effects	AbstractPathEffect	
picker	[None bool float callable]	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
transform	Transform	
url	a url string	
urls	List[str] or None	
visible	bool	
zorder	float	

Examples using matplotlib.axes.Axes.quiver

- sphx_glr_gallery_images_contours_and_fields_quiver_simple_demo.py
- sphx_glr_gallery_images_contours_and_fields_quiver_demo.py
- sphx_glr_gallery_images_contours_and_fields_trigradient_demo.py

matplotlib.axes.Axes.quiverkey

Axes.quiverkey(*args, **kw)

Add a key to a quiver plot.

Call signature:

quiverkey(Q, X, Y, U, label, **kw)

Arguments:

Q: The Quiver instance returned by a call to quiver.

X, Y: The location of the key; additional explanation follows.

U: The length of the key

label: A string with the length and units of the key

Keyword arguments:

angle = 0 The angle of the key arrow. Measured in degrees anti-clockwise from the x-axis.

coordinates = ['axes' | 'figure' | 'data' | 'inches'] Coordinate system and units for X, Y: 'axes' and 'figure' are normalized coordinate systems with 0,0 in the lower left and 1,1 in the upper right; 'data' are the axes data coordinates (used for the locations of the vectors in the quiver plot itself); 'inches' is position in the figure in inches, with 0,0 at the lower left corner.

color: overrides face and edge colors from Q.

labelpos = ['N' | 'S' | 'E' | 'W'] Position the label above, below, to the right, to the left of the arrow, respectively.

labelsep: Distance in inches between the arrow and the label. Default is 0.1

labelcolor: defaults to default Text color.

fontproperties: A dictionary with keyword arguments accepted by the FontProperties initializer: family, style, variant, size, weight

Any additional keyword arguments are used to override vector properties taken from Q.

The positioning of the key depends on X, Y, coordinates, and labelpos. If labelpos is 'N' or 'S', X, Y give the position of the middle of the key arrow. If labelpos is 'E', X, Y positions the head, and if labelpos is 'W', X, Y positions the tail; in either of these two cases, X, Y is somewhere in the middle of the arrow+label key object.

Examples using matplotlib.axes.Axes.quiverkey

- sphx_glr_gallery_images_contours_and_fields_quiver_simple_demo.py
- sphx_glr_gallery_images_contours_and_fields_quiver_demo.py

`matplotlib.axes.Axes.streamplot`

`Axes.streamplot(x, y, u, v, density=1, linewidth=None, color=None, cmap=None, norm=None, arrowsize=1, arrowstyle='->', minlength=0.1, transform=None, zorder=None, start_points=None, maxlength=4.0, integration_direction='both', *, data=None)`

Draws streamlines of a vector flow.

`x, y` [1d arrays] an evenly spaced grid.

`u, v` [2d arrays] x and y-velocities. Number of rows should match length of y, and the number of columns should match x.

`density` [float or 2-tuple] Controls the closeness of streamlines. When `density = 1`, the domain is divided into a 30x30 grid—density linearly scales this grid. Each cell in the grid can have, at most, one traversing streamline. For different densities in each direction, use `[density_x, density_y]`.

`linewidth` [numeric or 2d array] vary linewidth when given a 2d array with the same shape as velocities.

`color` [matplotlib color code, or 2d array] Streamline color. When given an array with the same shape as velocities, color values are converted to colors using `cmap`.

`cmap` [Colormap] Colormap used to plot streamlines and arrows. Only necessary when using an array input for color.

`norm` [Normalize] Normalize object used to scale luminance data to 0, 1. If None, stretch (min, max) to (0, 1). Only necessary when color is an array.

`arrowsize` [float] Factor scale arrow size.

`arrowstyle` [str] Arrow style specification. See `FancyArrowPatch`.

`minlength` [float] Minimum length of streamline in axes coordinates.

`start_points`: Nx2 array Coordinates of starting points for the streamlines. In data coordinates, the same as the x and y arrays.

`zorder` [int] any number

`maxlength` [float] Maximum length of streamline in axes coordinates.

`integration_direction` [['forward', 'backward', 'both']] Integrate the streamline in forward, backward or both directions.

Returns:

`stream_container` [StreamplotSet] Container object with attributes

- `lines`: `matplotlib.collections.LineCollection` of streamlines
- `arrows`: collection of `matplotlib.patches.FancyArrowPatch` objects representing arrows half-way along stream lines.

This container will probably change in the future to allow changes to the colormap, alpha, etc. for both lines and arrows, but these changes should be backward compatible.

Examples using `matplotlib.axes.Axes.streamplot`

- `sphx_glr_gallery_images_contours_and_fields_plot_streamplot.py`

35.2 Clearing

<code>Axes.cla</code>	Clear the current axes.
<code>Axes.clear</code>	Clear the axes.

35.2.1 `matplotlib.axes.Axes.cla`

`Axes.cla()`
Clear the current axes.

Examples using `matplotlib.axes.Axes.cla`

- `sphx_glr_gallery_misc_custom_projection.py`

35.2.2 `matplotlib.axes.Axes.clear`

`Axes.clear()`
Clear the axes.

35.3 Appearance

<code>Axes.axis</code>	Set axis properties.
<code>Axes.set_axis_off</code>	Turn off the axis.
<code>Axes.set_axis_on</code>	Turn on the axis.
<code>Axes.set_frame_on</code>	Set whether the axes rectangle patch is drawn.
<code>Axes.get_frame_on</code>	Get whether the axes rectangle patch is drawn.
<code>Axes.set_axisbelow</code>	Set whether axis ticks and gridlines are above or below most artists.
<code>Axes.get_axisbelow</code>	Get whether axis ticks and gridlines are above or below most artists.
<code>Axes.grid</code>	Turn the axes grids on or off.
<code>Axes.get_facecolor</code>	Get the Axes facecolor.

Continued on next page

Table 31 – continued from previous page

<code>Axes.get_fc</code>	Get the Axes facecolor.
<code>Axes.set_facecolor</code>	Set the Axes facecolor.
<code>Axes.set_fc</code>	Set the Axes facecolor.

35.3.1 matplotlib.axes.Axes.axis

`Axes.axis(*v, **kwargs)`

Set axis properties.

Valid signatures:

```
xmin, xmax, ymin, ymax = axis()
xmin, xmax, ymin, ymax = axis(list_arg)
xmin, xmax, ymin, ymax = axis(string_arg)
xmin, xmax, ymin, ymax = axis(**kwargs)
```

Parameters

`v` [list of float or {'on', 'off', 'equal', 'tight', 'scaled', 'normal', 'auto', 'image', 'square'}] Optional positional argument

Axis data limits set from a list; or a command relating to axes:

Value	Description
'on'	Toggle axis lines and labels on
'off'	Toggle axis lines and labels off
'equal'	Equal scaling by changing limits
'scaled'	Equal scaling by changing box dimensions
'tight'	Limits set such that all data is shown
'auto'	Automatic scaling, fill rectangle with data
'normal'	Same as 'auto'; deprecated
'image'	'scaled' with axis limits equal to data limits
'square'	Square plot; similar to 'scaled', but initially forcing <code>xmax-xmin = ymax-ymin</code>

`emit` [bool, optional] Passed to `set_{x,y}lim` functions, if observers are notified of axis limit change

`xmin, ymin, xmax, ymax` [float, optional] The axis limits to be set

Returns

`xmin, xmax, ymin, ymax` [float] The axis limits

35.3.2 matplotlib.axes.Axes.set_axis_off

`Axes.set_axis_off()`

Turn off the axis.

Examples using `matplotlib.axes.Axes.set_axis_off`

- `sphx_glr_gallery_color_colormap_reference.py`
- `sphx_glr_gallery_images_contours_and_fields_image_transparency_blend.py`
- `sphx_glr_gallery_pie_and_polar_charts_nested_pie.py`

35.3.3 matplotlib.axes.Axes.set_axis_on

`Axes.set_axis_on()`

Turn on the axis.

35.3.4 matplotlib.axes.Axes.set_frame_on

`Axes.set_frame_on(b)`

Set whether the axes rectangle patch is drawn.

Parameters

b [bool]

35.3.5 matplotlib.axes.Axes.get_frame_on

`Axes.get_frame_on()`

Get whether the axes rectangle patch is drawn.

35.3.6 matplotlib.axes.Axes.set_axisbelow

`Axes.set_axisbelow(b)`

Set whether axis ticks and gridlines are above or below most artists.

Parameters

b [bool or 'line']

35.3.7 matplotlib.axes.Axes.get_axisbelow

`Axes.get_axisbelow()`

Get whether axis ticks and gridlines are above or below most artists.

35.3.8 matplotlib.axes.Axes.grid

`Axes.grid(b=None, which='major', axis='both', **kwargs)`

Turn the axes grids on or off.

Set the axes grids on or off; `b` is a boolean.

If `b` is `None` and `len(kwargs)==0`, toggle the grid state. If `kwargs` are supplied, it is assumed that you want a grid and `b` is thus set to `True`.

which can be 'major' (default), 'minor', or 'both' to control whether major tick grids, minor tick grids, or both are affected.

axis can be 'both' (default), 'x', or 'y' to control which set of gridlines are drawn.

kwargs are used to set the grid line properties, e.g.,:

```
ax.grid(color='r', linestyle='-', linewidth=2)
```

Valid Line2D kwargs are

	Property	Description
<code>agg_filter</code>		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)
<code>alpha</code>		float (0.0 transparent through 1.0 opaque)
<code>animated</code>		bool
<code>antialiased</code> or <code>aa</code>		bool
<code>clip_box</code>		a Bbox instance
<code>clip_on</code>		bool
<code>clip_path</code>		[(Path, Transform) Patch None]
<code>color</code> or <code>c</code>		any matplotlib color
<code>contains</code>		a callable function
<code>dash_capstyle</code>		['butt' 'round' 'projecting']
<code>dash_joinstyle</code>		['miter' 'round' 'bevel']
<code>dashes</code>		sequence of on/off ink in points
<code>drawstyle</code>		['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']
<code>figure</code>		a Figure instance
<code>fillstyle</code>		['full' 'left' 'right' 'bottom' 'top' 'none']
<code>gid</code>		an id string
<code>label</code>		object
<code>linestyle</code> or <code>ls</code>		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']
<code>linewidth</code> or <code>lw</code>		float value in points
<code>marker</code>		A valid marker style
<code>markeredgecolor</code> or <code>mec</code>		any matplotlib color
<code>markeredgewidth</code> or <code>mew</code>		float value in points
<code>markerfacecolor</code> or <code>mfc</code>		any matplotlib color
<code>markerfacecoloralt</code> or <code>mfcalt</code>		any matplotlib color
<code>markersize</code> or <code>ms</code>		float
<code>markevery</code>		[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]

Continued on next page

Table 32 – continued from previous page

	Property	Description
path_effects		AbstractPathEffect
picker		float distance in points or callable pick function fn(artist, event)
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
solid_capstyle		['butt' 'round' 'projecting']
solid_joinstyle		['miter' 'round' 'bevel']
transform		a matplotlib.transforms.Transform instance
url		a url string
visible		bool
xdata		1D array
ydata		1D array
zorder		float

35.3.9 matplotlib.axes.Axes.get_facecolor

Axes.get_facecolor()

Get the Axes facecolor.

35.3.10 matplotlib.axes.Axes.get_fc

Axes.get_fc()

Get the Axes facecolor.

35.3.11 matplotlib.axes.Axes.set_facecolor

Axes.set_facecolor(color)

Set the Axes facecolor.

Parameters

color [color]

Examples using matplotlib.axes.Axes.set_facecolor

- sphx_glr_gallery_color_color_demo.py
- sphx_glr_gallery_color_color_cycle_default.py

35.3.12 matplotlib.axes.Axes.set_fc

Axes.set_fc(color)

Set the Axes facecolor.

Parameters

color [color]

35.4 Property cycle

Axes.set_prop_cycle

Set the property cycle of the Axes.

Axes.set_color_cycle

Deprecated since version 1.5.

35.4.1 matplotlib.axes.Axes.set_prop_cycle

Axes.set_prop_cycle(*args, **kwargs)

Set the property cycle of the Axes.

The property cycle controls the style properties such as color, marker and linestyle of future plot commands. The style properties of data already added to the Axes are not modified.

Call signatures:

```
set_prop_cycle(cycler)
set_prop_cycle(label=values[, label2=values2[, ...]])
set_prop_cycle(label, values)
```

Form 1 sets given [Cycler](#) object.

Form 2 creates a [Cycler](#) which cycles over one or more properties simultaneously and set it as the property cycle of the axes. If multiple properties are given, their value lists must have the same length. This is just a shortcut for explicitly creating a cycler and passing it to the function, i.e. it's short for `set_prop_cycle(cycler(label=values label2=values2, ...))`.

Form 3 creates a [Cycler](#) for a single property and set it as the property cycle of the axes. This form exists for compatibility with the original [cycler.cycler](#) interface. Its use is discouraged in favor of the kwarg form, i.e. `set_prop_cycle(label=values)`.

Parameters

cycler [Cycler] Set the given Cycler. None resets to the cycle defined by the current style.

label [str] The property key. Must be a valid Artist property. For example, 'color' or 'linestyle'. Aliases are allowed, such as 'c' for 'color' and 'lw' for 'linewidth'.

values [iterable] Finite-length iterable of the property values. These values are validated and will raise a `ValueError` if invalid.

See also:

`matplotlib.rcsetup.cycler` Convenience function for creating validated cyclers for properties.

`cycler.cycler` The original function for creating unvalidated cyclers.

Examples

Setting the property cycle for a single property:

```
>>> ax.set_prop_cycle(color=['red', 'green', 'blue'])
```

Setting the property cycle for simultaneously cycling over multiple properties (e.g. red circle, green plus, blue cross):

```
>>> ax.set_prop_cycle(color=['red', 'green', 'blue'],
...                    marker=['o', '+', 'x'])
```

Examples using `matplotlib.axes.Axes.set_prop_cycle`

- `sphx_glr_gallery_color_color_cycler.py`

35.4.2 `matplotlib.axes.Axes.set_color_cycle`

`Axes.set_color_cycle(clist)`

Deprecated since version 1.5: The `set_color_cycle` function was deprecated in version 1.5. Use `set_prop_cycle` instead.

Set the color cycle for any future plot commands on this Axes.

Parameters

`clist` A list of mpl color specifiers.

35.5 Axis / limits

`Axes.get_xaxis`

Return the XAxis instance.

`Axes.get_yaxis`

Return the YAxis instance.

35.5.1 `matplotlib.axes.Axes.get_xaxis`

`Axes.get_xaxis()`

Return the XAxis instance.

35.5.2 matplotlib.axes.Axes.get__yaxis

`Axes.get__yaxis()`

Return the YAxis instance.

35.5.3 Axis Limits and direction

<code>Axes.invert__xaxis</code>	Invert the x-axis.
<code>Axes.xaxis__inverted</code>	Return whether the x-axis is inverted.
<code>Axes.invert__yaxis</code>	Invert the y-axis.
<code>Axes.yaxis__inverted</code>	Return whether the y-axis is inverted.
<code>Axes.set__xlim</code>	Set the data limits for the x-axis
<code>Axes.get__xlim</code>	Get the x-axis range
<code>Axes.set__ylim</code>	Set the data limits for the y-axis
<code>Axes.get__ylim</code>	Get the y-axis range
<code>Axes.update__datalim</code>	Update the data lim bbox with seq of xy tups or equiv.
<code>Axes.update__datalim__bounds</code>	Update the datalim to include the given Bbox bounds
<code>Axes.set__xbound</code>	Set the lower and upper numerical bounds of the x-axis.
<code>Axes.get__xbound</code>	Return the lower and upper x-axis bounds, in increasing order.
<code>Axes.set__ybound</code>	Set the lower and upper numerical bounds of the y-axis.
<code>Axes.get__ybound</code>	Return the lower and upper y-axis bounds, in increasing order.

matplotlib.axes.Axes.invert__xaxis

`Axes.invert__xaxis()`

Invert the x-axis.

matplotlib.axes.Axes.xaxis__inverted

`Axes.xaxis__inverted()`

Return whether the x-axis is inverted.

matplotlib.axes.Axes.invert__yaxis

`Axes.invert__yaxis()`

Invert the y-axis.

`matplotlib.axes.Axes.yaxis_inverted`

`Axes.yaxis_inverted()`

Return whether the y-axis is inverted.

`matplotlib.axes.Axes.set_xlim`

`Axes.set_xlim(left=None, right=None, emit=True, auto=False, **kw)`

Set the data limits for the x-axis

Parameters

`left` [scalar, optional] The left xlim (default: None, which leaves the left limit unchanged).

`right` [scalar, optional] The right xlim (default: None, which leaves the right limit unchanged).

`emit` [bool, optional] Whether to notify observers of limit change (default: True).

`auto` [bool or None, optional] Whether to turn on autoscaling of the x-axis. True turns on, False turns off (default action), None leaves unchanged.

`xlims` [tuple, optional] The left and right xlims may be passed as the tuple (left, right) as the first positional argument (or as the left keyword argument).

Returns

`xlims` [tuple] Returns the new x-axis limits as (left, right).

Notes

The left value may be greater than the right value, in which case the x-axis values will decrease from left to right.

Examples

```
>>> set_xlim(left, right)
>>> set_xlim((left, right))
>>> left, right = set_xlim(left, right)
```

One limit may be left unchanged.

```
>>> set_xlim(right=right_lim)
```

Limits may be passed in reverse order to flip the direction of the x-axis. For example, suppose `x` represents the number of years before present. The x-axis limits might be set like the following so 5000 years ago is on the left of the plot and the present is on the right.

```
>>> set_xlim(5000, 0)
```

Examples using `matplotlib.axes.Axes.set_xlim`

- `sphx_glr_gallery_shapes_and_collections_donut.py`
- `sphx_glr_gallery_misc_custom_projection.py`

`matplotlib.axes.Axes.get_xlim`

`Axes.get_xlim()`

Get the x-axis range

Returns

`xlimits` [tuple] Returns the current x-axis limits as the tuple (left, right).

Notes

The x-axis may be inverted, in which case the left value will be greater than the right value.

`matplotlib.axes.Axes.set_ylim`

`Axes.set_ylim(bottom=None, top=None, emit=True, auto=False, **kw)`

Set the data limits for the y-axis

Parameters

`bottom` [scalar, optional] The bottom ylim (default: None, which leaves the bottom limit unchanged).

`top` [scalar, optional] The top ylim (default: None, which leaves the top limit unchanged).

`emit` [bool, optional] Whether to notify observers of limit change (default: True).

`auto` [bool or None, optional] Whether to turn on autoscaling of the y-axis. True turns on, False turns off (default action), None leaves unchanged.

`ylimits` [tuple, optional] The bottom and top ylimits may be passed as the tuple (bottom, top) as the first positional argument (or as the bottom keyword argument).

Returns

`ylimits` [tuple] Returns the new y-axis limits as (bottom, top).

Notes

The bottom value may be greater than the top value, in which case the y-axis values will decrease from bottom to top.

Examples

```
>>> set_ylim(bottom, top)
>>> set_ylim((bottom, top))
>>> bottom, top = set_ylim(bottom, top)
```

One limit may be left unchanged.

```
>>> set_ylim(top=top_lim)
```

Limits may be passed in reverse order to flip the direction of the y-axis. For example, suppose y represents depth of the ocean in m. The y-axis limits might be set like the following so 5000 m depth is at the bottom of the plot and the surface, 0 m, is at the top.

```
>>> set_ylim(5000, 0)
```

Examples using `matplotlib.axes.Axes.set_ylim`

- `sphx_glr_gallery_shapes_and_collections_donut.py`
- `sphx_glr_gallery_misc_custom_projection.py`

`matplotlib.axes.Axes.get_ylim`

`Axes.get_ylim()`

Get the y-axis range

Returns

ylimits [tuple] Returns the current y-axis limits as the tuple (bottom, top).

Notes

The y-axis may be inverted, in which case the bottom value will be greater than the top value.

`matplotlib.axes.Axes.update_datalim`

`Axes.update_datalim(xys, updatex=True, updatey=True)`

Update the data lim bbox with seq of xy tups or equiv. 2-D array

`matplotlib.axes.Axes.update_datalim_bounds`

`Axes.update_datalim_bounds(bounds)`

Update the datalim to include the given Bbox bounds

`matplotlib.axes.Axes.set_xbound`

`Axes.set_xbound(lower=None, upper=None)`

Set the lower and upper numerical bounds of the x-axis.

This method will honor axes inversion regardless of parameter order. It will not change the `__autoscaleXon` attribute.

`matplotlib.axes.Axes.get_xbound`

`Axes.get_xbound()`

Return the lower and upper x-axis bounds, in increasing order.

`matplotlib.axes.Axes.set_ybound`

`Axes.set_ybound(lower=None, upper=None)`

Set the lower and upper numerical bounds of the y-axis. This method will honor axes inversion regardless of parameter order. It will not change the `__autoscaleYon` attribute.

`matplotlib.axes.Axes.get_ybound`

`Axes.get_ybound()`

Return the lower and upper y-axis bounds, in increasing order.

35.5.4 Axis Labels, title, and legend

<code>Axes.set_xlabel</code>	Set the label for the x-axis.
<code>Axes.get_xlabel</code>	Get the xlabel text string.
<code>Axes.set_ylabel</code>	Set the label for the y-axis.
<code>Axes.get_ylabel</code>	Get the ylabel text string.
<code>Axes.set_title</code>	Set a title for the axes.
<code>Axes.get_title</code>	Get an axes title.
<code>Axes.legend</code>	Places a legend on the axes.
<code>Axes.get_legend</code>	Return the Legend instance, or None if no legend is defined.
<code>Axes.get_legend_handles_labels</code>	Return handles and labels for legend

`matplotlib.axes.Axes.set_xlabel`

`Axes.set_xlabel(xlabel, fontdict=None, labelpad=None, **kwargs)`

Set the label for the x-axis.

Parameters

`xlabel` [str] The label text.

`labelpad` [scalar, optional, default: None] Spacing in points between the label and the x-axis.

Other Parameters

`**kwargs` [Text properties] Text properties control the appearance of the label.

See also:

`text` for information on how override and the optional args work

Examples using `matplotlib.axes.Axes.set_xlabel`

- `sphx_glr_gallery_color_color_demo.py`
- `sphx_glr_gallery_statistics_histogram_features.py`

`matplotlib.axes.Axes.get_xlabel`

`Axes.get_xlabel()`

Get the xlabel text string.

`matplotlib.axes.Axes.set_ylabel`

`Axes.set_ylabel(ylabel, fontdict=None, labelpad=None, **kwargs)`

Set the label for the y-axis.

Parameters

`ylabel` [str] The label text.

`labelpad` [scalar, optional, default: None] Spacing in points between the label and the y-axis.

Other Parameters

`**kwargs` [Text properties] Text properties control the appearance of the label.

See also:

`text` for information on how override and the optional args work

Examples using `matplotlib.axes.Axes.set_ylabel`

- `sphinx_glr_gallery_color_color_demo.py`
- `sphinx_glr_gallery_statistics_histogram_features.py`

`matplotlib.axes.Axes.get_ylabel`

`Axes.get_ylabel()`

Get the ylabel text string.

`matplotlib.axes.Axes.set_title`

`Axes.set_title(label, fontdict=None, loc='center', pad=None, **kwargs)`

Set a title for the axes.

Set one of the three available axes titles. The available titles are positioned above the axes in the center, flush with the left edge, and flush with the right edge.

Parameters

`label` [str] Text to use for the title

`fontdict` [dict] A dictionary controlling the appearance of the title text, the default fontdict is:

```
{'fontsize': rcParams['axes.titlesize'],  
'fontweight': rcParams['axes.titleweight'],  
'verticalalignment': 'baseline',  
'horizontalalignment': loc}
```

`loc` [{ 'center', 'left', 'right' }, str, optional] Which title to set, defaults to 'center'

`pad` [float] The offset of the title from the top of the axes, in points. Default is None to use `rcParams['axes.titlepad']`.

Returns

`text` [Text] The matplotlib text instance representing the title

Other Parameters

`**kwargs` [Text properties] Other keyword arguments are text properties, see `Text` for a list of valid text properties.

Examples using `matplotlib.axes.Axes.set_title`

- `sphinx_glr_gallery_color_color_demo.py`
- `sphinx_glr_gallery_statistics_histogram_features.py`

- sphx_glr_gallery_shapes_and_collections_donut.py

matplotlib.axes.Axes.get_title

Axes.get_title(loc='center')

Get an axes title.

Get one of the three available axes titles. The available titles are positioned above the axes in the center, flush with the left edge, and flush with the right edge.

Parameters

loc [({'center', 'left', 'right'}, str, optional)] Which title to get, defaults to 'center'.

Returns

title [str] The title text string.

matplotlib.axes.Axes.legend

Axes.legend(*args, **kwargs)

Places a legend on the axes.

Call signatures:

```
legend()
legend(labels)
legend(handles, labels)
```

The call signatures correspond to three different ways how to use this method.

1. Automatic detection of elements to be shown in the legend

The elements to be added to the legend are automatically determined, when you do not pass in any extra arguments.

In this case, the labels are taken from the artist. You can specify them either at artist creation or by calling the `set_label()` method on the artist:

```
line, = ax.plot([1, 2, 3], label='Inline label')
ax.legend()
```

or:

```
line.set_label('Label via method')
line, = ax.plot([1, 2, 3])
ax.legend()
```

Specific lines can be excluded from the automatic legend element selection by defining a label starting with an underscore. This is default for all artists, so calling `Axes.legend` without any arguments and without setting the labels manually will result in no legend being drawn.

2. Labeling existing plot elements

To make a legend for lines which already exist on the axes (via `plot` for instance), simply call this function with an iterable of strings, one for each legend item. For example:

```
ax.plot([1, 2, 3])
ax.legend(['A simple line'])
```

Note: This way of using is discouraged, because the relation between plot elements and labels is only implicit by their order and can easily be mixed up.

3. Explicitly defining the elements in the legend

For full control of which artists have a legend entry, it is possible to pass an iterable of legend artists followed by an iterable of legend labels respectively:

```
legend((line1, line2, line3), ('label1', 'label2', 'label3'))
```

Parameters

handles [sequence of Artist, optional] A list of Artists (lines, patches) to be added to the legend. Use this together with labels, if you need full control on what is shown in the legend and the automatic mechanism described above is not sufficient.

The length of handles and labels should be the same in this case. If they are not, they are truncated to the smaller length.

labels [sequence of strings, optional] A list of labels to show next to the artists. Use this together with handles, if you need full control on what is shown in the legend and the automatic mechanism described above is not sufficient.

Returns

:class:‘matplotlib.legend.Legend‘ instance

Other Parameters

loc [int or string or pair of floats, default: ‘upper right’] The location of the legend. Possible codes are:

Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

Alternatively can be a 2-tuple giving x, y of the lower-left corner of the legend in axes coordinates (in which case `bbox_to_anchor` will be ignored).

`bbox_to_anchor` [BboxBase or pair of floats] Specify any arbitrary location for the legend in `bbox_transform` coordinates (default Axes coordinates).

For example, to put the legend's upper right hand corner in the center of the axes the following keywords can be used:

```
loc='upper right', bbox_to_anchor=(0.5, 0.5)
```

`ncol` [integer] The number of columns that the legend has. Default is 1.

`prop` [None or `matplotlib.font_manager.FontProperties` or dict] The font properties of the legend. If None (default), the current `matplotlib.rcParams` will be used.

`fontsize` [int or float or {'xx-small', 'x-small', 'small', 'medium', 'large', 'xx-large', 'xx-large'}] Controls the font size of the legend. If the value is numeric the size will be the absolute font size in points. String values are relative to the current default font size. This argument is only used if `prop` is not specified.

`numpoints` [None or int] The number of marker points in the legend when creating a legend entry for a `Line2D` (line). Default is None, which will take the value from `rcParams["legend.numpoints"]`.

`scatterpoints` [None or int] The number of marker points in the legend when creating a legend entry for a `PathCollection` (scatter plot). Default is None, which will take the value from `rcParams["legend.scatterpoints"]`.

`scatteryoffsets` [iterable of floats] The vertical offset (relative to the font size) for the markers created for a scatter plot legend entry. 0.0 is at the base the legend text, and 1.0 is at the top. To draw all markers at the same height, set to [0.5]. Default is [0.375, 0.5, 0.3125].

`markerscale` [None or int or float] The relative size of legend markers compared with the originally drawn ones. Default is None, which will take the value

`from rcParams["legend.markerscale"]`.

`markerfirst` [bool] If True, legend marker is placed to the left of the legend label. If False, legend marker is placed to the right of the legend label. Default is True.

`frameon` [None or bool] Control whether the legend should be drawn on a patch (frame). Default is None, which will take the value from `rcParams["legend.frameon"]`.

`fancybox` [None or bool] Control whether round edges should be enabled around the FancyBboxPatch which makes up the legend's background. Default is None, which will take the value from `rcParams["legend.fancybox"]`.

`shadow` [None or bool] Control whether to draw a shadow behind the legend. Default is None, which will take the value from `rcParams["legend.shadow"]`.

`framealpha` [None or float] Control the alpha transparency of the legend's background. Default is None, which will take the value from `rcParams["legend.framealpha"]`. If shadow is activated and framealpha is None, the default value is ignored.

`facecolor` [None or "inherit" or a color spec] Control the legend's background color. Default is None, which will take the value from `rcParams["legend.facecolor"]`. If "inherit", it will take `rcParams["axes.facecolor"]`.

`edgecolor` [None or "inherit" or a color spec] Control the legend's background patch edge color. Default is None, which will take the value from `rcParams["legend.edgecolor"]`. If "inherit", it will take `rcParams["axes.edgecolor"]`.

`mode` [{"expand", None}] If mode is set to "expand" the legend will be horizontally expanded to fill the axes area (or `bbox_to_anchor` if defines the legend's size).

`bbox_transform` [None or `matplotlib.transforms.Transform`] The transform for the bounding box (`bbox_to_anchor`). For a value of None (default) the Axes' `transAxes` transform will be used.

`title` [str or None] The legend's title. Default is no title (None).

`borderpad` [float or None] The fractional whitespace inside the legend border. Measured in font-size units. Default is None, which will take the value from `rcParams["legend.borderpad"]`.

`labelspacing` [float or None] The vertical space between the legend entries. Measured in font-size units. Default is None, which will take the value from `rcParams["legend.labelspacing"]`.

`handlelength` [float or None] The length of the legend handles. Measured in font-size units. Default is None, which will take the value from `rcParams["legend.handlelength"]`.

`handletextpad` [float or None] The pad between the legend handle and text. Measured in font-size units. Default is None, which will take the value from

`rcParams["legend.handletextpad"]`.

`borderaxespad` [float or None] The pad between the axes and legend border. Measured in font-size units. Default is None, which will take the value from `rcParams["legend.borderaxespad"]`.

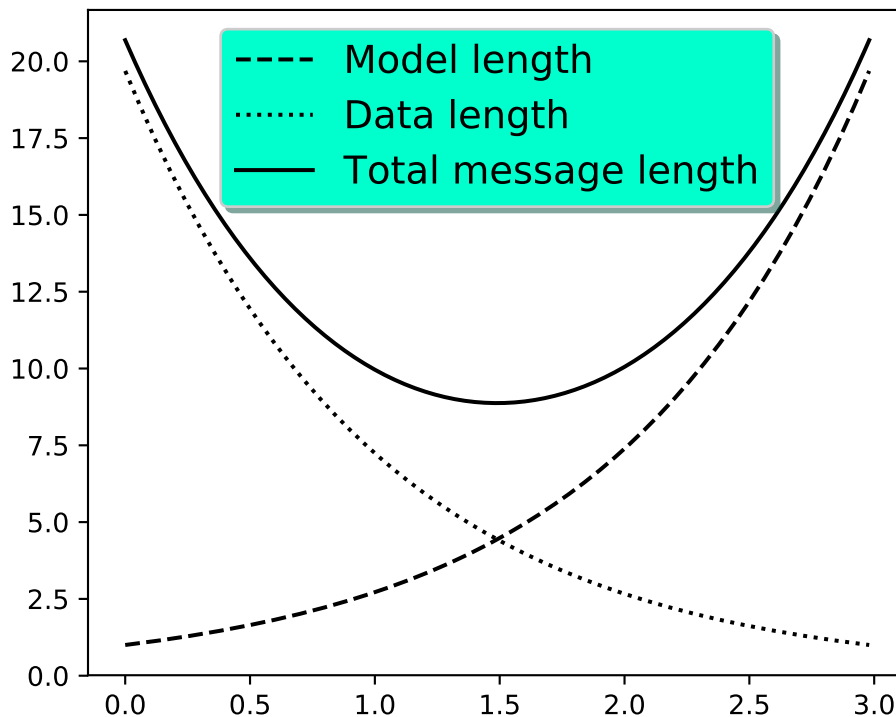
`columnspacing` [float or None] The spacing between columns. Measured in font-size units. Default is None, which will take the value from `rcParams["legend.columnspacing"]`.

`handler_map` [dict or None] The custom dictionary mapping instances or types to a legend handler. This `handler_map` updates the default handler map found at `matplotlib.legend.Legend.get_legend_handler_map()`.

Notes

Not all kinds of artist are supported by the legend command. See [Legend guide](#) for details.

Examples



Examples using `matplotlib.axes.Axes.legend`

- `sphinx_glr_gallery_images_contours_and_fields_contourf_hatching.py`
- `sphinx_glr_gallery_images_contours_and_fields_contourf_log.py`
- `sphinx_glr_gallery_pie_and_polar_charts_polar_legend.py`
- `sphinx_glr_gallery_pie_and_polar_charts_pie_and_donut_labels.py`

`matplotlib.axes.Axes.get_legend`

`Axes.get_legend()`

Return the Legend instance, or None if no legend is defined.

`matplotlib.axes.Axes.get_legend_handles_labels`

`Axes.get_legend_handles_labels(legend_handler_map=None)`

Return handles and labels for legend

`ax.legend()` is equivalent to

```
h, l = ax.get_legend_handles_labels()
ax.legend(h, l)
```

35.5.5 Axis scales

<code>Axes.set_xscale</code>	Set the x-axis scale.
<code>Axes.get_xscale</code>	Return the xaxis scale string: linear, log, logit, symlog
<code>Axes.set_yscale</code>	Set the y-axis scale.
<code>Axes.get_yscale</code>	Return the yaxis scale string: linear, log, logit, symlog

`matplotlib.axes.Axes.set_xscale`

`Axes.set_xscale(value, **kwargs)`

Set the x-axis scale.

Parameters

value [{"linear", "log", "symlog", "logit"}] scaling strategy to apply

See also:

`matplotlib.scale.LinearScale` linear transform

`matplotlib.scale.LogTransform` log transform

`matplotlib.scale.SymmetricalLogTransform` symlog transform

`matplotlib.scale.LogisticTransform` logit transform

Notes

Different kwargs are accepted, depending on the scale. See the scale module for more information.

`matplotlib.axes.Axes.get__xscale`

`Axes.get__xscale()`

Return the xaxis scale string: linear, log, logit, symlog

`matplotlib.axes.Axes.set__yscale`

`Axes.set__yscale(value, **kwargs)`

Set the y-axis scale.

Parameters

value [{ "linear", "log", "symlog", "logit"}] scaling strategy to apply

See also:

`matplotlib.scale.LinearScale` linear transform

`matplotlib.scale.LogTransform` log transform

`matplotlib.scale.SymmetricalLogTransform` symlog transform

`matplotlib.scale.LogisticTransform` logit transform

Notes

Different kwargs are accepted, depending on the scale. See the scale module for more information.

`matplotlib.axes.Axes.get__yscale`

`Axes.get__yscale()`

Return the yaxis scale string: linear, log, logit, symlog

35.5.6 Autoscaling and margins

<code>Axes.use_sticky_edges</code>	When autoscaling, whether to obey all <code>Artist.sticky_edges</code> .
<code>Axes.margins</code>	Set or retrieve autoscaling margins.
<code>Axes.set_xmargin</code>	Set padding of X data limits prior to autoscaling.
<code>Axes.set_ymargin</code>	Set padding of Y data limits prior to autoscaling.
<code>Axes.relim</code>	Recompute the data limits based on current artists.
<code>Axes.autoscale</code>	Autoscale the axis view to the data (toggle).
<code>Axes.autoscale_view</code>	Autoscale the view limits using the data limits.
<code>Axes.set_autoscale_on</code>	Set whether autoscaling is applied on plot commands
<code>Axes.get_autoscale_on</code>	Get whether autoscaling is applied for both axes on plot commands
<code>Axes.set_autoscalex_on</code>	Set whether autoscaling for the x-axis is applied on plot commands
<code>Axes.get_autoscalex_on</code>	Get whether autoscaling for the x-axis is applied on plot commands
<code>Axes.set_autoscaley_on</code>	Set whether autoscaling for the y-axis is applied on plot commands
<code>Axes.get_autoscaley_on</code>	Get whether autoscaling for the y-axis is applied on plot commands

`matplotlib.axes.Axes.use_sticky_edges`

`Axes.use_sticky_edges`

When autoscaling, whether to obey all `Artist.sticky_edges`.

Default is `True`.

Setting this to `False` ensures that the specified margins will be applied, even if the plot includes an image, for example, which would otherwise force a view limit to coincide with its data limit.

The changing this property does not change the plot until `autoscale` or `autoscale_view` is called.

`matplotlib.axes.Axes.margins`

`Axes.margins(*args, **kw)`

Set or retrieve autoscaling margins.

signatures:

`margins()`

returns `xmargin`, `ymargin`

```

margins(margin)

margins(xmargin, ymargin)

margins(x=xmargin, y=ymargin)

margins(..., tight=False)

```

All three forms above set the `xmargin` and `ymargin` parameters. All keyword parameters are optional. A single argument specifies both `xmargin` and `ymargin`. The padding added to the end of each interval is `margin` times the data interval. The `margin` must be a float in the range `[0, 1]`.

The `tight` parameter is passed to `autoscale_view()`, which is executed after a margin is changed; the default here is `True`, on the assumption that when margins are specified, no additional padding to match tick marks is usually desired. Setting `tight` to `None` will preserve the previous setting.

Specifying any margin changes only the autoscaling; for example, if `xmargin` is not `None`, then `xmargin` times the X data interval will be added to each end of that interval before it is used in autoscaling.

`matplotlib.axes.Axes.set_xmargin`

`Axes.set_xmargin(m)`

Set padding of X data limits prior to autoscaling.

`m` times the data interval will be added to each end of that interval before it is used in autoscaling. For example, if your data is in the range `[0, 2]`, a factor of `m = 0.1` will result in a range `[-0.2, 2.2]`.

Negative values $-0.5 < m < 0$ will result in clipping of the data range. I.e. for a data range `[0, 2]`, a factor of `m = -0.1` will result in a range `[0.2, 1.8]`.

Parameters

`m` [float greater than -0.5]

`matplotlib.axes.Axes.set_ymargin`

`Axes.set_ymargin(m)`

Set padding of Y data limits prior to autoscaling.

`m` times the data interval will be added to each end of that interval before it is used in autoscaling. For example, if your data is in the range `[0, 2]`, a factor of `m = 0.1` will result in a range `[-0.2, 2.2]`.

Negative values $-0.5 < m < 0$ will result in clipping of the data range. I.e. for a data range `[0, 2]`, a factor of `m = -0.1` will result in a range `[0.2, 1.8]`.

Parameters

m [float greater than -0.5]

matplotlib.axes.Axes.relim

Axes.relim(visible_only=False)

Recompute the data limits based on current artists. If you want to exclude invisible artists from the calculation, set `visible_only=True`

At present, Collection instances are not supported.

matplotlib.axes.Axes.autoscale

Axes.autoscale(enable=True, axis='both', tight=None)

Autoscale the axis view to the data (toggle).

Convenience method for simple axis view autoscaling. It turns autoscaling on or off, and then, if autoscaling for either axis is on, it performs the autoscaling on the specified axis or axes.

Parameters

enable [bool or None, optional] True (default) turns autoscaling on, False turns it off. None leaves the autoscaling state unchanged.

axis [['both' | 'x' | 'y'], optional] which axis to operate on; default is 'both'

tight: bool or None, optional If True, set view limits to data limits; if False, let the locator and margins expand the view limits; if None, use tight scaling if the only artist is an image, otherwise treat tight as False. The tight setting is retained for future autoscaling until it is explicitly changed.

matplotlib.axes.Axes.autoscale_view

Axes.autoscale_view(tight=None, scalex=True, scaley=True)

Autoscale the view limits using the data limits.

You can selectively autoscale only a single axis, e.g., the xaxis by setting `scaley` to False. The autoscaling preserves any axis direction reversal that has already been done.

If `tight` is False, the axis major locator will be used to expand the view limits if `rcParams['axes.autolimit_mode']` is 'round_numbers'. Note that any margins that are in effect will be applied first, regardless of whether `tight` is True or False. Specifying `tight` as True or False saves the setting as a private attribute of the Axes; specifying it as None (the default) applies the previously saved value.

The data limits are not updated automatically when artist data are changed after the artist has been added to an Axes instance. In that case, use `matplotlib.axes.Axes.relim()` prior to calling `autoscale_view`.

Examples using `matplotlib.axes.Axes.autoscale__view`

- `sphx_glr_gallery_shapes_and_collections_ellipse_collection.py`
- `sphx_glr_gallery_shapes_and_collections_compound_path.py`
- `sphx_glr_gallery_shapes_and_collections_collections.py`

`matplotlib.axes.Axes.set__autoscale__on`

`Axes.set__autoscale__on(b)`

Set whether autoscaling is applied on plot commands

Parameters

b [bool]

`matplotlib.axes.Axes.get__autoscale__on`

`Axes.get__autoscale__on()`

Get whether autoscaling is applied for both axes on plot commands

`matplotlib.axes.Axes.set__autoscalex__on`

`Axes.set__autoscalex__on(b)`

Set whether autoscaling for the x-axis is applied on plot commands

Parameters

b [bool]

`matplotlib.axes.Axes.get__autoscalex__on`

`Axes.get__autoscalex__on()`

Get whether autoscaling for the x-axis is applied on plot commands

`matplotlib.axes.Axes.set__autoscaley__on`

`Axes.set__autoscaley__on(b)`

Set whether autoscaling for the y-axis is applied on plot commands

Parameters

b [bool]

`matplotlib.axes.Axes.get_autoscaley_on`

`Axes.get_autoscaley_on()`

Get whether autoscaling for the y-axis is applied on plot commands

35.5.7 Aspect ratio

<code>Axes.apply_aspect</code>	Adjust the Axes for a specified data aspect ratio.
<code>Axes.set_aspect</code>	Set the aspect of the axis scaling, i.e.
<code>Axes.get_aspect</code>	
<code>Axes.set_adjustable</code>	Define which parameter the Axes will change to achieve a given aspect.
<code>Axes.get_adjustable</code>	

`matplotlib.axes.Axes.apply_aspect`

`Axes.apply_aspect(position=None)`

Adjust the Axes for a specified data aspect ratio.

Depending on `get_adjustable` this will modify either the Axes box (position) or the view limits. In the former case, `get_anchor` will affect the position.

See also:

`matplotlib.axes.Axes.set_aspect` for a description of aspect ratio handling.

`matplotlib.axes.Axes.set_adjustable` defining the parameter to adjust in order to meet the required aspect.

`matplotlib.axes.Axes.set_anchor` defining the position in case of extra space.

Notes

This is called automatically when each Axes is drawn. You may need to call it yourself if you need to update the Axes position and/or view limits before the Figure is drawn.

`matplotlib.axes.Axes.set_aspect`

`Axes.set_aspect(aspect, adjustable=None, anchor=None, share=False)`

Set the aspect of the axis scaling, i.e. the ratio of y-unit to x-unit.

Parameters

`aspect` `[['auto' | 'equal'] or num]` Possible values:

value	description
'auto'	automatic; fill the position rectangle with data
'equal'	same scaling from data to plot units for x and y
num	a circle will be stretched such that the height is num times the width. aspect=1 is the same as aspect='equal'.

`adjustable` [None or ['box' | 'datalim'], optional] If not None, this defines which parameter will be adjusted to meet the required aspect. See `set__adjustable` for further details.

`anchor` [None or str or 2-tuple of float, optional] If not None, this defines where the Axes will be drawn if there is extra space due to aspect constraints. The most common way to specify the anchor are abbreviations of cardinal directions:

value	description
'C'	centered
'SW'	lower left corner
'S'	middle of bottom edge
'SE'	lower right corner
etc.	

See `set__anchor` for further details.

`share` [bool, optional] If True, apply the settings to all shared Axes. Default is False.

See also:

`matplotlib.axes.Axes.set__adjustable` defining the parameter to adjust in order to meet the required aspect.

`matplotlib.axes.Axes.set__anchor` defining the position in case of extra space.

Examples using `matplotlib.axes.Axes.set__aspect`

- `sphx_glr_gallery_shapes_and_collections_donut.py`

`matplotlib.axes.Axes.get__aspect`

`Axes.get__aspect()`

matplotlib.axes.Axes.set_adjustable

Axes.set_adjustable(adjustable, share=False)

Define which parameter the Axes will change to achieve a given aspect.

Parameters

adjustable `[['box' | 'datalim']]` If 'box', change the physical dimensions of the Axes. If 'datalim', change the x or y data limits.

share `[bool, optional]` If True, apply the settings to all shared Axes. Default is False.

.. ACCEPTS: `['box' | 'datalim']`

See also:

matplotlib.axes.Axes.set_aspect for a description of aspect handling.

Notes

Shared Axes (of which twinned Axes are a special case) impose restrictions on how aspect ratios can be imposed. For twinned Axes, use 'datalim'. For Axes that share both x and y, use 'box'. Otherwise, either 'datalim' or 'box' may be used. These limitations are partly a requirement to avoid over-specification, and partly a result of the particular implementation we are currently using, in which the adjustments for aspect ratios are done sequentially and independently on each Axes as it is drawn.

matplotlib.axes.Axes.get_adjustable

Axes.get_adjustable()

35.5.8 Ticks and tick labels

Axes.set_xticks	Set the x ticks with list of ticks
Axes.get_xticks	Return the x ticks as a list of locations
Axes.set_xticklabels	Set the x-tick labels with list of string labels.
Axes.get_xticklabels	Get the x tick labels as a list of Text instances.
Axes.get_xmajorticklabels	Get the major x tick labels.
Axes.get_xminorticklabels	Get the minor x tick labels.
Axes.get_xgridlines	Get the x grid lines as a list of Line2D instances.
Axes.get_xticklines	Get the x tick lines as a list of Line2D instances.

Continued on next page

Table 40 – continued from previous page

<code>Axes.xaxis_date</code>	Sets up x-axis ticks and labels that treat the x data as dates.
<code>Axes.set_yticks</code>	Set the y ticks with list of ticks
<code>Axes.get_yticks</code>	Return the y ticks as a list of locations
<code>Axes.set_yticklabels</code>	Set the y-tick labels with list of strings labels.
<code>Axes.get_yticklabels</code>	Get the y tick labels as a list of Text instances.
<code>Axes.get_ymajorticklabels</code>	Get the major y tick labels.
<code>Axes.get_yminorticklabels</code>	Get the minor y tick labels.
<code>Axes.get_ygridlines</code>	Get the y grid lines as a list of Line2D instances.
<code>Axes.get_yticklines</code>	Get the y tick lines as a list of Line2D instances.
<code>Axes.yaxis_date</code>	Sets up y-axis ticks and labels that treat the y data as dates.
<code>Axes.minorticks_off</code>	Remove minor ticks from the axes.
<code>Axes.minorticks_on</code>	Add autoscaling minor ticks to the axes.
<code>Axes.ticklabel_format</code>	Change the ScalarFormatter used by default for linear axes.
<code>Axes.tick_params</code>	Change the appearance of ticks, tick labels, and gridlines.
<code>Axes.locator_params</code>	Control behavior of tick locators.

`matplotlib.axes.Axes.set_xticks`

`Axes.set_xticks(ticks, minor=False)`
Set the x ticks with list of ticks

Parameters

`ticks` [list] List of x-axis tick locations.

`minor` [bool, optional] If False sets major ticks, if True sets minor ticks. Default is False.

`matplotlib.axes.Axes.get_xticks`

`Axes.get_xticks(minor=False)`
Return the x ticks as a list of locations

`matplotlib.axes.Axes.set_xticklabels`

`Axes.set_xticklabels(labels, fontdict=None, minor=False, **kwargs)`
Set the x-tick labels with list of string labels.

Parameters

`labels` [list of str] List of string labels.

`fontdict` [dict, optional] A dictionary controlling the appearance of the ticklabels. The default fontdict is:

```
{'fontsize': rcParams['axes.titlesize'],  
'fontweight': rcParams['axes.titleweight'],  
'verticalalignment': 'baseline',  
'horizontalalignment': 'left'}
```

`minor` [bool, optional] Whether to set the minor ticklabels rather than the major ones.

Returns

A list of `~.text.Text` instances.

Other Parameters

`**kwargs` [Text properties.]

`matplotlib.axes.Axes.get_xticklabels`

`Axes.get_xticklabels(minor=False, which=None)`

Get the x tick labels as a list of Text instances.

Parameters

`minor` [bool, optional] If True return the minor ticklabels, else return the major ticklabels.

`which` [None, ('minor', 'major', 'both')] Overrides minor.

Selects which ticklabels to return

Returns

`ret` [list] List of Text instances.

`matplotlib.axes.Axes.get_xmajorticklabels`

`Axes.get_xmajorticklabels()`

Get the major x tick labels.

Returns

`labels` [list] List of Text instances

`matplotlib.axes.Axes.get_xminorticklabels`

`Axes.get_xminorticklabels()`

Get the minor x tick labels.

Returns

labels [list] List of Text instances

matplotlib.axes.Axes.get_xgridlines

Axes.get_xgridlines()

Get the x grid lines as a list of Line2D instances.

matplotlib.axes.Axes.get_xticklines

Axes.get_xticklines()

Get the x tick lines as a list of Line2D instances.

matplotlib.axes.Axes.xaxis_date

Axes.xaxis_date(tz=None)

Sets up x-axis ticks and labels that treat the x data as dates.

Parameters

tz [string or tzinfo instance, optional] Timezone string or timezone. Defaults to rc value.

matplotlib.axes.Axes.set_yticks

Axes.set_yticks(ticks, minor=False)

Set the y ticks with list of ticks

Parameters

ticks [sequence] List of y-axis tick locations

minor [bool, optional] If False sets major ticks, if True sets minor ticks. Default is False.

matplotlib.axes.Axes.get_yticks

Axes.get_yticks(minor=False)

Return the y ticks as a list of locations

matplotlib.axes.Axes.set_yticklabels

Axes.set_yticklabels(labels, fontdict=None, minor=False, **kwargs)

Set the y-tick labels with list of strings labels.

Parameters

labels [list of str] list of string labels

fontdict [dict, optional] A dictionary controlling the appearance of the ticklabels. The default fontdict is:

```
{'fontsize': rcParams['axes.titlesize'],  
'fontweight': rcParams['axes.titleweight'],  
'verticalalignment': 'baseline',  
'horizontalalignment': 'left'}
```

minor [bool, optional] Whether to set the minor ticklabels rather than the major ones.

Returns

A list of `~.text.Text` instances.

Other Parameters

****kwargs** [Text properties.]

`matplotlib.axes.Axes.get_yticklabels`

`Axes.get_yticklabels(minor=False, which=None)`

Get the y tick labels as a list of Text instances.

Parameters

minor [bool] If True return the minor ticklabels, else return the major ticklabels

which [None, ('minor', 'major', 'both')] Overrides minor.

Selects which ticklabels to return

Returns

ret [list] List of Text instances.

`matplotlib.axes.Axes.get_ymajorticklabels`

`Axes.get_ymajorticklabels()`

Get the major y tick labels.

Returns

labels [list] List of Text instances

`matplotlib.axes.Axes.get_yminorticklabels`

`Axes.get_yminorticklabels()`

Get the minor y tick labels.

Returns

labels [list] List of Text instances

`matplotlib.axes.Axes.get_ygridlines`

`Axes.get_ygridlines()`

Get the y grid lines as a list of Line2D instances.

`matplotlib.axes.Axes.get_yticklines`

`Axes.get_yticklines()`

Get the y tick lines as a list of Line2D instances.

`matplotlib.axes.Axes.yaxis__date`

`Axes.yaxis__date(tz=None)`

Sets up y-axis ticks and labels that treat the y data as dates.

Parameters

tz [string or tzinfo instance, optional] Timezone string or timezone. Defaults to rc value.

`matplotlib.axes.Axes.minorticks__off`

`Axes.minorticks__off()`

Remove minor ticks from the axes.

`matplotlib.axes.Axes.minorticks__on`

`Axes.minorticks__on()`

Add autoscaling minor ticks to the axes.

`matplotlib.axes.Axes.ticklabel__format`

`Axes.ticklabel__format(**kwargs)`

Change the ScalarFormatter used by default for linear axes.

Optional keyword arguments:

Key-word	Description
style	['sci' (or 'scientific') 'plain'] plain turns off scientific notation
scilimits	(m, n), pair of integers; if style is 'sci', scientific notation will be used for numbers outside the range 10^m to 10^n . Use (0,0) to include all numbers.
use-Offset	[bool offset]; if True, the offset will be calculated as needed; if False, no offset will be used; if a numeric offset is specified, it will be used.
axis	['x' 'y' 'both']
use-Locale	If True, format the number according to the current locale. This affects things such as the character used for the decimal separator. If False, use C-style (English) formatting. The default setting is controlled by the <code>axes.formatter.use_locale</code> rparam.
use-Math-Text	If True, render the offset and scientific notation in <code>mathtext</code>

Only the major ticks are affected. If the method is called when the `ScalarFormatter` is not the `Formatter` being used, an `AttributeError` will be raised.

`matplotlib.axes.Axes.tick_params`

`Axes.tick_params(axis='both', **kwargs)`

Change the appearance of ticks, tick labels, and gridlines.

Parameters

`axis` [{'x', 'y', 'both'}, optional] Which axis to apply the parameters to.

Other Parameters

`axis` [{'x', 'y', 'both'}] Axis on which to operate; default is 'both'.

`reset` [bool] If True, set all parameters to defaults before processing other keyword arguments. Default is False.

`which` [{'major', 'minor', 'both'}] Default is 'major'; apply arguments to which ticks.

`direction` [{'in', 'out', 'inout'}] Puts ticks inside the axes, outside the axes, or both.

`length` [float] Tick length in points.

`width` [float] Tick width in points.

`color` [color] Tick color; accepts any mpl color spec.

`pad` [float] Distance in points between tick and label.

`labelsize` [float or str] Tick label font size in points or as a string (e.g., 'large').

`labelcolor` [color] Tick label color; mpl color spec.
`colors` [color] Changes the tick color and the label color to the same value: mpl color spec.
`zorder` [float] Tick and label zorder.
`bottom, top, left, right` [bool] Whether to draw the respective ticks.
`labelbottom, labeltop, labelleft, labelright` [bool] Whether to draw the respective tick labels.
`labelrotation` [float] Tick label rotation
`grid_color` [color] Changes the gridline color to the given mpl color spec.
`grid_alpha` [float] Transparency of gridlines: 0 (transparent) to 1 (opaque).
`grid_linewidth` [float] Width of gridlines in points.
`grid_linestyle` [string] Any valid Line2D line style spec.

Examples

Usage

```
ax.tick_params(direction='out', length=6, width=2, colors='r',
               grid_color='r', grid_alpha=0.5)
```

This will make all major ticks be red, pointing out of the box, and with dimensions 6 points by 2 points. Tick labels will also be red. Gridlines will be red and translucent.

Examples using `matplotlib.axes.Axes.tick_params`

- `sphx_glr_gallery_color_color_demo.py`

`matplotlib.axes.Axes.locator_params`

`Axes.locator_params(axis='both', tight=None, **kwargs)`

Control behavior of tick locators.

Parameters

`axis` [['both' | 'x' | 'y'], optional] The axis on which to operate.
`tight` [bool or None, optional] Parameter passed to `autoscale_view()`. Default is None, for no change.

Other Parameters

`**kw` : Remaining keyword arguments are passed directly to the `set_params()` method.

Typically one might want to reduce the maximum number of ticks and use tight bounds when plotting small subplots, for example:: `ax.locator_params(tight=True, nbins=4)`

Because the locator is involved in autoscaling, :meth:'autoscale_view' is called automatically after the parameters are changed.

This presently works only for the :class:'~matplotlib.ticker.MaxNLocator' used by default on linear axes, but it may be generalized.

35.6 Units

<code>Axes.convert_xunits</code>	For artists in an axes, if the xaxis has units support, convert x using xaxis unit type
<code>Axes.convert_yunits</code>	For artists in an axes, if the yaxis has units support, convert y using yaxis unit type
<code>Axes.have_units</code>	Return True if units are set on the x or y axes

35.6.1 matplotlib.axes.Axes.convert_xunits

`Axes.convert_xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

35.6.2 matplotlib.axes.Axes.convert_yunits

`Axes.convert_yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

35.6.3 matplotlib.axes.Axes.have_units

`Axes.have_units()`

Return True if units are set on the x or y axes

35.7 Adding Artists

<code>Axes.add_artist</code>	Add any Artist to the axes.
<code>Axes.add_collection</code>	Add a Collection instance to the axes.

Continued on next page

Table 42 – continued from previous page

<code>Axes.add_container</code>	Add a Container instance to the axes.
<code>Axes.add_image</code>	Add a AxesImage to the axes.
<code>Axes.add_line</code>	Add a Line2D to the list of plot lines
<code>Axes.add_patch</code>	Add a Patch p to the list of axes patches; the clipbox will be set to the Axes clipping box.
<code>Axes.add_table</code>	Add a Table instance to the list of axes tables

35.7.1 matplotlib.axes.Axes.add_artist

`Axes.add_artist(a)`

Add any Artist to the axes.

Use `add_artist` only for artists for which there is no dedicated "add" method; and if necessary, use a method such as `update_dataLim` to manually update the dataLim if the artist is to be included in autoscaling.

Returns the artist.

Examples using `matplotlib.axes.Axes.add_artist`

- `sphx_glr_gallery_shapes_and_collections_ellipse_demo.py`

35.7.2 matplotlib.axes.Axes.add_collection

`Axes.add_collection(collection, autolim=True)`

Add a Collection instance to the axes.

Returns the collection.

Examples using `matplotlib.axes.Axes.add_collection`

- `sphx_glr_gallery_shapes_and_collections_ellipse_collection.py`
- `sphx_glr_gallery_shapes_and_collections_line_collection.py`
- `sphx_glr_gallery_shapes_and_collections_artist_reference.py`
- `sphx_glr_gallery_shapes_and_collections_collections.py`

35.7.3 matplotlib.axes.Axes.add_container

`Axes.add_container(container)`

Add a Container instance to the axes.

Returns the collection.

35.7.4 matplotlib.axes.Axes.add_image

`Axes.add_image(image)`

Add a `AxesImage` to the axes.

Returns the image.

35.7.5 matplotlib.axes.Axes.add_line

`Axes.add_line(line)`

Add a `Line2D` to the list of plot lines

Returns the line.

Examples using `matplotlib.axes.Axes.add_line`

- `sphx_glr_gallery_shapes_and_collections_artist_reference.py`

35.7.6 matplotlib.axes.Axes.add_patch

`Axes.add_patch(p)`

Add a `Patch` `p` to the list of axes patches; the clipbox will be set to the Axes clipping box. If the transform is not set, it will be set to `transData`.

Returns the patch.

Examples using `matplotlib.axes.Axes.add_patch`

- `sphx_glr_gallery_shapes_and_collections_compound_path.py`
- `sphx_glr_gallery_shapes_and_collections_path_patch.py`
- `sphx_glr_gallery_shapes_and_collections_hatch_demo.py`
- `sphx_glr_gallery_shapes_and_collections_donut.py`
- `sphx_glr_gallery_shapes_and_collections_dolphin.py`

35.7.7 matplotlib.axes.Axes.add_table

`Axes.add_table(tab)`

Add a `Table` instance to the list of axes tables

Parameters

tab: ‘matplotlib.table.Table’ Table instance

Returns

‘matplotlib.table.Table’: the table.

35.8 Twinning

<code>Axes.twinx</code>	Create a twin Axes sharing the xaxis
<code>Axes.twiny</code>	Create a twin Axes sharing the yaxis
<code>Axes.get_shared_x_axes</code>	Return a reference to the shared axes Grouper object for x axes.
<code>Axes.get_shared_y_axes</code>	Return a reference to the shared axes Grouper object for y axes.

35.8.1 `matplotlib.axes.Axes.twinx`

`Axes.twinx()`

Create a twin Axes sharing the xaxis

Create a new Axes instance with an invisible x-axis and an independent y-axis positioned opposite to the original one (i.e. at right). The x-axis autoscale setting will be inherited from the original Axes. To ensure that the tick marks of both y-axes align, see `LinearLocator`

Returns

`ax_twin` [`Axes`] The newly created Axes instance

Notes

For those who are 'picking' artists while using `twinx`, pick events are only called for the artists in the top-most axes.

35.8.2 `matplotlib.axes.Axes.twiny`

`Axes.twiny()`

Create a twin Axes sharing the yaxis

Create a new Axes instance with an invisible y-axis and an independent x-axis positioned opposite to the original one (i.e. at top). The y-axis autoscale setting will be inherited from the original Axes. To ensure that the tick marks of both x-axes align, see `LinearLocator`

Returns

`ax_twin` [`Axes`] The newly created Axes instance

Notes

For those who are 'picking' artists while using `twiny`, pick events are only called for the artists in the top-most axes.

35.8.3 matplotlib.axes.Axes.get_shared_x_axes

`Axes.get_shared_x_axes()`

Return a reference to the shared axes Grouper object for x axes.

35.8.4 matplotlib.axes.Axes.get_shared_y_axes

`Axes.get_shared_y_axes()`

Return a reference to the shared axes Grouper object for y axes.

35.9 Axes Position

<code>Axes.get_anchor</code>	Get the anchor location.
<code>Axes.set_anchor</code>	Define the anchor location.
<code>Axes.get_axes_locator</code>	Return the axes_locator.
<code>Axes.set_axes_locator</code>	Set the axes locator.
<code>Axes.reset_position</code>	Reset the active position to the original position.
<code>Axes.get_position</code>	Get a copy of the axes rectangle as a Bbox.
<code>Axes.set_position</code>	Set the axes position.

35.9.1 matplotlib.axes.Axes.get_anchor

`Axes.get_anchor()`

Get the anchor location.

See also:

`matplotlib.axes.Axes.set_anchor` for a description of the anchor.

`matplotlib.axes.Axes.set_aspect` for a description of aspect handling.

35.9.2 matplotlib.axes.Axes.set_anchor

`Axes.set_anchor(anchor, share=False)`

Define the anchor location.

The actual drawing area (active position) of the Axes may be smaller than the Bbox (original position) when a fixed aspect is required. The anchor defines where the drawing area will be located within the available space.

Parameters

`anchor` [str or 2-tuple of floats] The anchor position may be either:

- a sequence (cx, cy). cx and cy may range from 0 to 1, where 0 is left or bottom and 1 is right or top.

- a string using cardinal directions as abbreviation:
 - 'C' for centered
 - 'S' (south) for bottom-center
 - 'SW' (south west) for bottom-left
 - etc.

Here is an overview of the possible positions:

'NW'	'N'	'NE'
'W'	'C'	'E'
'SW'	'S'	'SE'

`share` [bool, optional] If True, apply the settings to all shared Axes. Default is False.

See also:

`matplotlib.axes.Axes.set_aspect` for a description of aspect handling.

35.9.3 `matplotlib.axes.Axes.get_axes_locator`

`Axes.get_axes_locator()`

Return the `axes_locator`.

35.9.4 `matplotlib.axes.Axes.set_axes_locator`

`Axes.set_axes_locator(locator)`

Set the axes locator.

Parameters

`locator` [callable] A locator function, which takes an axes and a renderer and returns a `bbox`.

35.9.5 `matplotlib.axes.Axes.reset_position`

`Axes.reset_position()`

Reset the active position to the original position.

This resets the a possible position change due to aspect constraints. For an explanation of the positions see `set_position`.

35.9.6 matplotlib.axes.Axes.get_position

`Axes.get_position(original=False)`

Get a copy of the axes rectangle as a Bbox.

Parameters

`original` [bool] If True, return the original position. Otherwise return the active position. For an explanation of the positions see `set_position`.

Returns

`pos` [Bbox]

Examples using `matplotlib.axes.Axes.get_position`

- `sphx_glr_gallery_images_contours_and_fields_contour_demo.py`

35.9.7 matplotlib.axes.Axes.set_position

`Axes.set_position(pos, which='both')`

Set the axes position.

Axes have two position attributes. The 'original' position is the position allocated for the Axes. The 'active' position is the position the Axes is actually drawn at. These positions are usually the same unless a fixed aspect is set to the Axes. See `set_aspect` for details.

Parameters

`pos` [[left, bottom, width, height] or Bbox] The new position of the in Figure coordinates.

`which` [['both' | 'active' | 'original'], optional] Determines which position variables to change.

Examples using `matplotlib.axes.Axes.set_position`

- `sphx_glr_gallery_images_contours_and_fields_contour_demo.py`

35.10 Async/Event based

`Axes.stale`

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

`Axes.pchanged`

Fire an event when property changed, calling all of the registered callbacks.

Continued on next page

Table 45 – continued from previous page

<code>Axes.add_callback</code>	Adds a callback function that will be called whenever one of the Artist's properties changes.
<code>Axes.remove_callback</code>	Remove a callback based on its id.

35.10.1 `matplotlib.axes.Axes.stale`

`Axes.stale`

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

35.10.2 `matplotlib.axes.Axes.pchanged`

`Axes.pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

35.10.3 `matplotlib.axes.Axes.add_callback`

`Axes.add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

35.10.4 `matplotlib.axes.Axes.remove_callback`

`Axes.remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

35.11 Interactive

<code>Axes.can_pan</code>	Return True if this axes supports any pan/zoom button functionality.
<code>Axes.can_zoom</code>	Return True if this axes supports the zoom box button functionality.
<code>Axes.get_navigate</code>	Get whether the axes responds to navigation commands
<code>Axes.set_navigate</code>	Set whether the axes responds to navigation toolbar commands

Continued on next page

Table 46 – continued from previous page

<code>Axes.get_navigate_mode</code>	Get the navigation toolbar button status: 'PAN', 'ZOOM', or None
<code>Axes.set_navigate_mode</code>	Set the navigation toolbar button status;
<code>Axes.start_pan</code>	Called when a pan operation has started.
<code>Axes.drag_pan</code>	Called when the mouse moves during a pan operation.
<code>Axes.end_pan</code>	Called when a pan operation completes (when the mouse button is up.)
<code>Axes.format_coord</code>	Return a format string formatting the x, y coord
<code>Axes.format_cursor_data</code>	Return cursor data string formatted.
<code>Axes.format_xdata</code>	Return x string formatted.
<code>Axes.format_ydata</code>	Return y string formatted.
<code>Axes.mouseover</code>	
<code>Axes.in_axes</code>	Return True if the given mouseevent (in display coords) is in the Axes
<code>Axes.pick</code>	Trigger pick event
<code>Axes.pickable</code>	Return True if Artist is pickable.
<code>Axes.get_picker</code>	Return the picker object used by this artist.
<code>Axes.set_picker</code>	Set the epsilon for picking used by this artist
<code>Axes.set_contains</code>	Replace the contains test used by this artist.
<code>Axes.get_contains</code>	Return the <code>_contains</code> test used by the artist, or None for default.
<code>Axes.contains</code>	Test whether the mouse event occurred in the axes.
<code>Axes.contains_point</code>	Returns True if the point (tuple of x,y) is inside the axes (the area defined by the its patch).
<code>Axes.get_cursor_data</code>	Get the cursor data for a given event.
<code>Axes.get_cursor_props</code>	Deprecated since version 2.1.
<code>Axes.set_cursor_props</code>	Deprecated since version 2.1.

35.11.1 matplotlib.axes.Axes.can_pan

`Axes.can_pan()`

Return True if this axes supports any pan/zoom button functionality.

35.11.2 matplotlib.axes.Axes.can_zoom

`Axes.can_zoom()`

Return True if this axes supports the zoom box button functionality.

35.11.3 matplotlib.axes.Axes.get_navigate

`Axes.get_navigate()`

Get whether the axes responds to navigation commands

35.11.4 matplotlib.axes.Axes.set_navigate

`Axes.set_navigate(b)`

Set whether the axes responds to navigation toolbar commands

Parameters

`b` [bool]

35.11.5 matplotlib.axes.Axes.get_navigate_mode

`Axes.get_navigate_mode()`

Get the navigation toolbar button status: 'PAN', 'ZOOM', or None

35.11.6 matplotlib.axes.Axes.set_navigate_mode

`Axes.set_navigate_mode(b)`

Set the navigation toolbar button status;

Warning: this is not a user-API function.

35.11.7 matplotlib.axes.Axes.start_pan

`Axes.start_pan(x, y, button)`

Called when a pan operation has started.

`x`, `y` are the mouse coordinates in display coords. `button` is the mouse button number:

- 1: LEFT
- 2: MIDDLE
- 3: RIGHT

Note: Intended to be overridden by new projection types.

35.11.8 matplotlib.axes.Axes.drag_pan

`Axes.drag_pan(button, key, x, y)`

Called when the mouse moves during a pan operation.

button is the mouse button number:

- 1: LEFT
- 2: MIDDLE
- 3: RIGHT

key is a "shift" key

x, y are the mouse coordinates in display coords.

Note: Intended to be overridden by new projection types.

35.11.9 matplotlib.axes.Axes.end_pan

`Axes.end_pan()`

Called when a pan operation completes (when the mouse button is up.)

Note: Intended to be overridden by new projection types.

35.11.10 matplotlib.axes.Axes.format_coord

`Axes.format_coord(x, y)`

Return a format string formatting the x, y coord

35.11.11 matplotlib.axes.Axes.format_cursor_data

`Axes.format_cursor_data(data)`

Return cursor data string formatted.

35.11.12 matplotlib.axes.Axes.format_xdata

`Axes.format_xdata(x)`

Return x string formatted. This function will use the attribute `self.fmt_xdata` if it is callable, else will fall back on the xaxis major formatter

35.11.13 matplotlib.axes.Axes.format_ydata

`Axes.format_ydata(y)`

Return y string formatted. This function will use the `fnt_ydata` attribute if it is callable, else will fall back on the yaxis major formatter

35.11.14 matplotlib.axes.Axes.mouseover

`Axes.mouseover`

35.11.15 matplotlib.axes.Axes.in__axes

`Axes.in__axes(mouseevent)`

Return True if the given mouseevent (in display coords) is in the Axes

35.11.16 matplotlib.axes.Axes.pick

`Axes.pick(*args)`

Trigger pick event

Call signature:

<code>pick(mouseevent)</code>

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

35.11.17 matplotlib.axes.Axes.pickable

`Axes.pickable()`

Return True if Artist is pickable.

35.11.18 matplotlib.axes.Axes.get_picker

`Axes.get_picker()`

Return the picker object used by this artist.

35.11.19 matplotlib.axes.Axes.set_picker

`Axes.set_picker(picker)`

Set the epsilon for picking used by this artist

picker can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if picker is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if picker is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

to determine the hit test. if the mouse event is over the artist, return hit=True and props is a dictionary of properties you want added to the PickEvent attributes.

Parameters

picker [None or bool or float or callable]

35.11.20 matplotlib.axes.Axes.set_contains

Axes.set_contains(picker)

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return hit = True and props is a dictionary of properties you want returned with the contains test.

Parameters

picker [callable]

35.11.21 matplotlib.axes.Axes.get_contains

Axes.get_contains()

Return the __contains test used by the artist, or None for default.

35.11.22 matplotlib.axes.Axes.contains

Axes.contains(mouseevent)

Test whether the mouse event occurred in the axes.

Returns True / False, {}

35.11.23 matplotlib.axes.Axes.contains_point

`Axes.contains_point(point)`

Returns True if the point (tuple of x,y) is inside the axes (the area defined by the its patch).
A pixel coordinate is required.

35.11.24 matplotlib.axes.Axes.get_cursor_data

`Axes.get_cursor_data(event)`

Get the cursor data for a given event.

35.11.25 matplotlib.axes.Axes.get_cursor_props

`Axes.get_cursor_props()`

Deprecated since version 2.1: The `get_cursor_props` function was deprecated in version 2.1.

Return the cursor propertiess as a (linewidth, color) tuple, where linewidth is a float and color is an RGBA tuple

35.11.26 matplotlib.axes.Axes.set_cursor_props

`Axes.set_cursor_props(*args)`

Deprecated since version 2.1: The `set_cursor_props` function was deprecated in version 2.1.

Set the cursor property as

Call signature

```
ax.set_cursor_props(linewidth, color)
```

or:

```
ax.set_cursor_props((linewidth, color))
```

ACCEPTS: a (float, color) tuple

35.12 Children

<code>Axes.get_children</code>	return a list of child artists
<code>Axes.get_images</code>	return a list of Axes images contained by the Axes
<code>Axes.get_lines</code>	Return a list of lines contained by the Axes
<code>Axes.findobjj</code>	Find artist objects.

35.12.1 matplotlib.axes.Axes.get_children

Axes.get_children()
return a list of child artists

35.12.2 matplotlib.axes.Axes.get_images

Axes.get_images()
return a list of Axes images contained by the Axes

35.12.3 matplotlib.axes.Axes.get_lines

Axes.get_lines()
Return a list of lines contained by the Axes

35.12.4 matplotlib.axes.Axes.findobj

Axes.findobj(match=None, include_self=True)
Find artist objects.
Recursively find all Artist instances contained in self.
match can be

- None: return all objects contained in artist.
- function with signature boolean = match(artist) used to filter matches
- class instance: e.g., Line2D. Only return artists of class type.

If include_self is True (default), include self in the list to be checked for a match.

35.13 Drawing

Axes.draw	Draw everything (plot lines, axes, labels)
Axes.draw_artist	This method can only be used after an initial draw which caches the renderer.
Axes.redraw_in_frame	This method can only be used after an initial draw which caches the renderer.
Axes.get_renderer_cache	
Axes.get_rasterization_zorder	Return the zorder value below which artists will be rasterized.
Axes.set_rasterization_zorder	
Parameters	

Continued on next page

Table 48 – continued from previous page

<code>Axes.get_window_extent</code>	get the axes bounding box in display space; args and kwargs are empty
<code>Axes.get_tightbbox</code>	Return the tight bounding box of the axes.

35.13.1 `matplotlib.axes.Axes.draw`

`Axes.draw(renderer=None, inframe=False)`
 Draw everything (plot lines, axes, labels)

Examples using `matplotlib.axes.Axes.draw`

- `sphx_glr_gallery_text_labels_and_annotations_usetex_baseline_test.py`

35.13.2 `matplotlib.axes.Axes.draw_artist`

`Axes.draw_artist(a)`
 This method can only be used after an initial draw which caches the renderer. It is used to efficiently update Axes data (axis ticks, labels, etc are not updated)

35.13.3 `matplotlib.axes.Axes.redraw_in_frame`

`Axes.redraw_in_frame()`
 This method can only be used after an initial draw which caches the renderer. It is used to efficiently update Axes data (axis ticks, labels, etc are not updated)

35.13.4 `matplotlib.axes.Axes.get_renderer_cache`

`Axes.get_renderer_cache()`

35.13.5 `matplotlib.axes.Axes.get_rasterization_zorder`

`Axes.get_rasterization_zorder()`
 Return the zorder value below which artists will be rasterized.

35.13.6 `matplotlib.axes.Axes.set_rasterization_zorder`

`Axes.set_rasterization_zorder(z)`

Parameters

`z` [float or None] zorder below which artists are rasterized. None means that artists do not get rasterized based on zorder.

35.13.7 matplotlib.axes.Axes.get_window_extent

`Axes.get_window_extent(*args, **kwargs)`

get the axes bounding box in display space; args and kwargs are empty

35.13.8 matplotlib.axes.Axes.get_tightbbox

`Axes.get_tightbbox(renderer, call_axes_locator=True)`

Return the tight bounding box of the axes. The dimension of the Bbox in canvas coordinate.

If `call_axes_locator` is False, it does not call the `_axes_locator` attribute, which is necessary to get the correct bounding box. `call_axes_locator==False` can be used if the caller is only interested in the relative size of the tightbbox compared to the axes bbox.

35.14 Bulk property manipulation

<code>Axes.set</code>	A property batch setter.
<code>Axes.update</code>	Update this artist's properties from the dictionary prop.
<code>Axes.properties</code>	return a dictionary mapping property name -> value for all Artist props
<code>Axes.update_from</code>	Copy properties from other to self.

35.14.1 matplotlib.axes.Axes.set

`Axes.set(**kwargs)`

A property batch setter. Pass kwargs to set properties.

Examples using `matplotlib.axes.Axes.set`

- `sphx_glr_gallery_pie_and_polar_charts_nested_pie.py`

35.14.2 matplotlib.axes.Axes.update

`Axes.update(props)`

Update this artist's properties from the dictionary prop.

35.14.3 matplotlib.axes.Axes.properties

`Axes.properties()`

return a dictionary mapping property name -> value for all Artist props

35.14.4 matplotlib.axes.Axes.update_from

`Axes.update_from(other)`

Copy properties from other to self.

35.15 General Artist Properties

<code>Axes.set_agg_filter</code>	Set the agg filter.
<code>Axes.set_alpha</code>	Set the alpha value used for blending - not supported on all backends.
<code>Axes.set_animated</code>	Set the artist's animation state.
<code>Axes.set_clip_box</code>	Set the artist's clip Bbox.
<code>Axes.set_clip_on</code>	Set whether artist uses clipping.
<code>Axes.set_clip_path</code>	Set the artist's clip path, which may be:
<code>Axes.set_gid</code>	Sets the (group) id for the artist.
<code>Axes.set_label</code>	Set the label to s for auto legend.
<code>Axes.set_path_effects</code>	Set the path effects.
<code>Axes.set_rasterized</code>	Force rasterized (bitmap) drawing in vector backend output.
<code>Axes.set_sketch_params</code>	Sets the sketch parameters.
<code>Axes.set_snap</code>	Sets the snap setting which may be:
<code>Axes.set_transform</code>	Set the artist transform.
<code>Axes.set_url</code>	Sets the url for the artist.
<code>Axes.set_visible</code>	Set the artist's visibility.
<code>Axes.set_zorder</code>	Set the zorder for the artist.
<code>Axes.get_agg_filter</code>	Return filter function to be used for agg filter.
<code>Axes.get_alpha</code>	Return the alpha value used for blending - not supported on all backends
<code>Axes.get_animated</code>	Return the artist's animated state
<code>Axes.get_clip_box</code>	Return artist clipbox
<code>Axes.get_clip_on</code>	Return whether artist uses clipping
<code>Axes.get_clip_path</code>	Return artist clip path
<code>Axes.get_gid</code>	Returns the group id.
<code>Axes.get_label</code>	Get the label used for this artist in the legend.
<code>Axes.get_path_effects</code>	
<code>Axes.get_rasterized</code>	Return whether the artist is to be rasterized.
<code>Axes.get_sketch_params</code>	Returns the sketch parameters for the artist.
<code>Axes.get_snap</code>	Returns the snap setting which may be:
<code>Axes.get_transform</code>	Return the Transform instance used by this artist.
<code>Axes.get_url</code>	Returns the url.
<code>Axes.get_visible</code>	Return the artist's visibility
<code>Axes.get_zorder</code>	Return the artist's zorder.

Continued on next page

Table 50 – continued from previous page

<code>Axes.axes</code>	The Axes instance the artist resides in, or None.
<code>Axes.set_figure</code>	Set the Figure for this Axes.
<code>Axes.get_figure</code>	Return the Figure instance the artist belongs to.

35.15.1 `matplotlib.axes.Axes.set_agg_filter`

`Axes.set_agg_filter(filter_func)`

Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

35.15.2 `matplotlib.axes.Axes.set_alpha`

`Axes.set_alpha(alpha)`

Set the alpha value used for blending - not supported on all backends.

Parameters

`alpha` [float]

35.15.3 `matplotlib.axes.Axes.set_animated`

`Axes.set_animated(b)`

Set the artist's animation state.

Parameters

`b` [bool]

35.15.4 `matplotlib.axes.Axes.set_clip_box`

`Axes.set_clip_box(clipbox)`

Set the artist's clip Bbox.

Parameters

`clipbox` [Bbox]

35.15.5 `matplotlib.axes.Axes.set_clip_on`

`Axes.set_clip_on(b)`

Set whether artist uses clipping.

When `False` artists will be visible out side of the axes which can lead to unexpected results.

Parameters

`b` [bool]

35.15.6 `matplotlib.axes.Axes.set_clip_path`

`Axes.set_clip_path(path, transform=None)`

Set the artist's clip path, which may be:

- a `Patch` (or subclass) instance; or
- a `Path` instance, in which case a `Transform` instance, which will be applied to the path before using it for clipping, must be provided; or
- `None`, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to `None`.

ACCEPTS: [(`Path`, `Transform`) | `Patch` | `None`]

35.15.7 `matplotlib.axes.Axes.set_gid`

`Axes.set_gid(gid)`

Sets the (group) id for the artist.

Parameters

`gid` [str]

35.15.8 `matplotlib.axes.Axes.set_label`

`Axes.set_label(s)`

Set the label to `s` for auto legend.

Parameters

`s` [object] `s` will be converted to a string by calling `str` (unicode on Py2).

35.15.9 `matplotlib.axes.Axes.set_path_effects`

`Axes.set_path_effects(path_effects)`

Set the path effects.

Parameters

`path_effects` [`AbstractPathEffect`]

35.15.10 matplotlib.axes.Axes.set__rasterized

`Axes.set__rasterized(rasterized)`

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

`rasterized` [bool or None]

35.15.11 matplotlib.axes.Axes.set__sketch__params

`Axes.set__sketch__params(scale=None, length=None, randomness=None)`

Sets the sketch parameters.

Parameters

`scale` [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If scale is `None`, or not provided, no sketch filter will be provided.

`length` [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

`randomness` [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

35.15.12 matplotlib.axes.Axes.set__snap

`Axes.set__snap(snap)`

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

`snap` [bool or None]

35.15.13 matplotlib.axes.Axes.set__transform

`Axes.set__transform(t)`

Set the artist transform.

Parameters

t [Transform]

35.15.14 matplotlib.axes.Axes.set_url

Axes.set_url(url)

Sets the url for the artist.

Parameters

url [str]

35.15.15 matplotlib.axes.Axes.set_visible

Axes.set_visible(b)

Set the artist's visibility.

Parameters

b [bool]

35.15.16 matplotlib.axes.Axes.set_zorder

Axes.set_zorder(level)

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

level [float]

35.15.17 matplotlib.axes.Axes.get_agg_filter

Axes.get_agg_filter()

Return filter function to be used for agg filter.

35.15.18 matplotlib.axes.Axes.get_alpha

Axes.get_alpha()

Return the alpha value used for blending - not supported on all backends

35.15.19 matplotlib.axes.Axes.get_animated

Axes.get_animated()

Return the artist's animated state

35.15.20 matplotlib.axes.Axes.get_clip_box

`Axes.get_clip_box()`
Return artist clipbox

35.15.21 matplotlib.axes.Axes.get_clip_on

`Axes.get_clip_on()`
Return whether artist uses clipping

35.15.22 matplotlib.axes.Axes.get_clip_path

`Axes.get_clip_path()`
Return artist clip path

35.15.23 matplotlib.axes.Axes.get_gid

`Axes.get_gid()`
Returns the group id.

35.15.24 matplotlib.axes.Axes.get_label

`Axes.get_label()`
Get the label used for this artist in the legend.

35.15.25 matplotlib.axes.Axes.get_path_effects

`Axes.get_path_effects()`

35.15.26 matplotlib.axes.Axes.get_rasterized

`Axes.get_rasterized()`
Return whether the artist is to be rasterized.

35.15.27 matplotlib.axes.Axes.get_sketch_params

`Axes.get_sketch_params()`
Returns the sketch parameters for the artist.

Returns

 `sketch_params` [tuple or `None`]

A 3-tuple with the following elements:

- scale: The amplitude of the wiggle perpendicular to the source line.
- length: The length of the wiggle along the line.
- randomness: The scale factor by which the length is shrunk or expanded.

May return 'None' if no sketch parameters were set.

35.15.28 `matplotlib.axes.Axes.get_snap`

`Axes.get_snap()`

Returns the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

35.15.29 `matplotlib.axes.Axes.get_transform`

`Axes.get_transform()`

Return the Transform instance used by this artist.

35.15.30 `matplotlib.axes.Axes.get_url`

`Axes.get_url()`

Returns the url.

35.15.31 `matplotlib.axes.Axes.get_visible`

`Axes.get_visible()`

Return the artist's visibility

35.15.32 `matplotlib.axes.Axes.get_zorder`

`Axes.get_zorder()`

Return the artist's zorder.

35.15.33 matplotlib.axes.Axes.axes

`Axes.axes`

The `Axes` instance the artist resides in, or `None`.

35.15.34 matplotlib.axes.Axes.set_figure

`Axes.set_figure(fig)`

Set the `Figure` for this `Axes`.

Parameters

`fig` [`Figure`]

35.15.35 matplotlib.axes.Axes.get_figure

`Axes.get_figure()`

Return the `Figure` instance the artist belongs to.

35.16 Artist Methods

<code>Axes.remove</code>	Remove the artist from the figure if possible.
<code>Axes.is_transform_set</code>	Returns <code>True</code> if Artist has a transform explicitly set.

35.16.1 matplotlib.axes.Axes.remove

`Axes.remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

35.16.2 matplotlib.axes.Axes.is_transform_set

`Axes.is_transform_set()`

Returns `True` if Artist has a transform explicitly set.

35.17 Projection

Methods used by Axis that must be overridden for non-rectilinear Axes.

Axes.name	
Axes.get_xaxis_transform	Get the transformation used for drawing x-axis labels, ticks and gridlines.
Axes.get_yaxis_transform	Get the transformation used for drawing y-axis labels, ticks and gridlines.
Axes.get_data_ratio	Returns the aspect ratio of the raw data.
Axes.get_data_ratio_log	Returns the aspect ratio of the raw data in log scale.
Axes.get_xaxis_text1_transform	Get the transformation used for drawing x-axis labels, which will add the given amount of padding (in points) between the axes and the label.
Axes.get_xaxis_text2_transform	Get the transformation used for drawing the secondary x-axis labels, which will add the given amount of padding (in points) between the axes and the label.
Axes.get_yaxis_text1_transform	Get the transformation used for drawing y-axis labels, which will add the given amount of padding (in points) between the axes and the label.
Axes.get_yaxis_text2_transform	Get the transformation used for drawing the secondary y-axis labels, which will add the given amount of padding (in points) between the axes and the label.

35.17.1 matplotlib.axes.Axes.name

Axes.name = 'rectilinear'

35.17.2 matplotlib.axes.Axes.get_xaxis_transform

Axes.get_xaxis_transform(which='grid')

Get the transformation used for drawing x-axis labels, ticks and gridlines. The x-direction is in data coordinates and the y-direction is in axis coordinates.

Note: This transformation is primarily used by the Axis class, and is meant to be overridden by new kinds of projections that may need to place axis elements in different locations.

35.17.3 matplotlib.axes.Axes.get_yaxis_transform

`Axes.get_yaxis_transform(which='grid')`

Get the transformation used for drawing y-axis labels, ticks and gridlines. The x-direction is in axis coordinates and the y-direction is in data coordinates.

Note: This transformation is primarily used by the Axis class, and is meant to be overridden by new kinds of projections that may need to place axis elements in different locations.

35.17.4 matplotlib.axes.Axes.get_data_ratio

`Axes.get_data_ratio()`

Returns the aspect ratio of the raw data.

This method is intended to be overridden by new projection types.

35.17.5 matplotlib.axes.Axes.get_data_ratio_log

`Axes.get_data_ratio_log()`

Returns the aspect ratio of the raw data in log scale. Will be used when both axis scales are in log.

35.17.6 matplotlib.axes.Axes.get_xaxis_text1_transform

`Axes.get_xaxis_text1_transform(pad_points)`

Get the transformation used for drawing x-axis labels, which will add the given amount of padding (in points) between the axes and the label. The x-direction is in data coordinates and the y-direction is in axis coordinates. Returns a 3-tuple of the form:

(transform, valign, halign)

where valign and halign are requested alignments for the text.

Note: This transformation is primarily used by the Axis class, and is meant to be overridden by new kinds of projections that may need to place axis elements in different locations.

35.17.7 matplotlib.axes.Axes.get_xaxis_text2_transform

`Axes.get_xaxis_text2_transform(pad_points)`

Get the transformation used for drawing the secondary x-axis labels, which will add the given amount of padding (in points) between the axes and the label. The x-direction is in data coordinates and the y-direction is in axis coordinates. Returns a 3-tuple of the form:

(transform, valign, halign)

where valign and halign are requested alignments for the text.

Note: This transformation is primarily used by the Axis class, and is meant to be overridden by new kinds of projections that may need to place axis elements in different locations.

35.17.8 matplotlib.axes.Axes.get_yaxis_text1_transform

`Axes.get_yaxis_text1_transform(pad_points)`

Get the transformation used for drawing y-axis labels, which will add the given amount of padding (in points) between the axes and the label. The x-direction is in axis coordinates and the y-direction is in data coordinates. Returns a 3-tuple of the form:

(transform, valign, halign)

where valign and halign are requested alignments for the text.

Note: This transformation is primarily used by the Axis class, and is meant to be overridden by new kinds of projections that may need to place axis elements in different locations.

35.17.9 matplotlib.axes.Axes.get_yaxis_text2_transform

`Axes.get_yaxis_text2_transform(pad_points)`

Get the transformation used for drawing the secondary y-axis labels, which will add the given amount of padding (in points) between the axes and the label. The x-direction is in axis coordinates and the y-direction is in data coordinates. Returns a 3-tuple of the form:

(transform, valign, halign)

where valign and halign are requested alignments for the text.

Note: This transformation is primarily used by the Axis class, and is meant to be overridden by new kinds of projections that may need to place axis elements in different locations.

35.18 Other

`Axes.zorder`

Continued on next page

Table 53 – continued from previous page

<code>Axes.aname</code>	
<code>Axes.get_default_bbox_extra_artists</code>	
<code>Axes.get_transformed_clip_path_and_affine</code>	Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.
<code>Axes.has_data</code>	Return True if any artists have been added to axes.
<code>Axes.hold</code>	Deprecated since version 2.0.
<code>Axes.ishold</code>	Deprecated since version 2.0.

35.18.1 `matplotlib.axes.Axes.zorder`

`Axes.zorder = 0`

35.18.2 `matplotlib.axes.Axes.aname`

`Axes.aname = 'Axes'`

35.18.3 `matplotlib.axes.Axes.get_default_bbox_extra_artists`

`Axes.get_default_bbox_extra_artists()`

35.18.4 `matplotlib.axes.Axes.get_transformed_clip_path_and_affine`

`Axes.get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

35.18.5 `matplotlib.axes.Axes.has_data`

`Axes.has_data()`

Return True if any artists have been added to axes.

This should not be used to determine whether the dataLim need to be updated, and may not actually be useful for anything.

35.18.6 matplotlib.axes.Axes.hold

`Axes.hold(b=None)`

Deprecated since version 2.0: `axes.hold` is deprecated. See the API Changes document (http://matplotlib.org/api/api_changes.html) for more details.

Set the hold state.

The hold mechanism is deprecated and will be removed in v3.0. The behavior will remain consistent with the long-time default value of `True`.

If `hold` is `None` (default), toggle the hold state. Else set the hold state to boolean value `b`.

Examples:

```
# toggle hold
hold()

# turn hold on
hold(True)

# turn hold off
hold(False)
```

When `hold` is `True`, subsequent plot commands will be added to the current axes. When `hold` is `False`, the current axes and figure will be cleared on the next plot command

35.18.7 matplotlib.axes.Axes.ishold

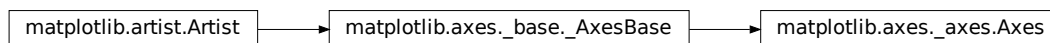
`Axes.ishold()`

Deprecated since version 2.0: The `ishold` function was deprecated in version 2.0.

return the `HOLD` status of the axes

The hold mechanism is deprecated and will be removed in v3.0.

35.19 Inheritance



AXIS AND TICK API

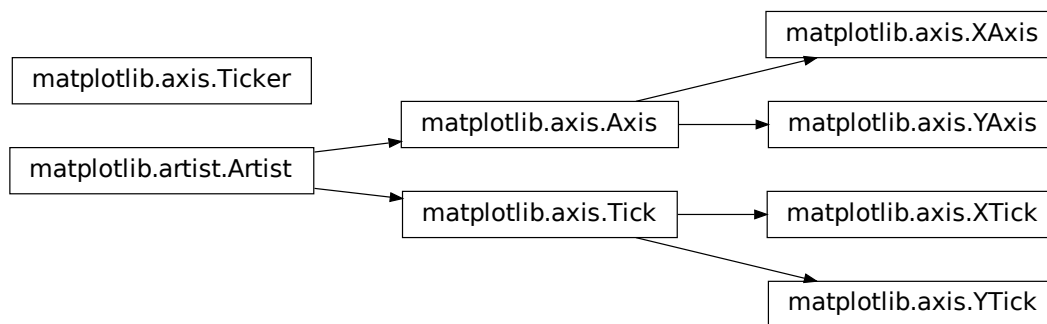
Table of Contents

- Inheritance
- Axis objects
 - Formatters and Locators
 - Axis Label
 - Ticks, tick labels and Offset text
 - Data and view intervals
 - Rendering helpers
 - Interactive
 - Units
 - Incremental navigation
 - YAxis Specific
 - XAxis Specific
 - Other
 - Discouraged
- Tick objects
- Common and inherited methods
 - XTick
 - YTick
 - YAxis
 - XAxis
 - Inherited from artist
 - * Ticks

* Axis

Classes for the ticks and x and y axis

36.1 Inheritance



36.2 Axis objects

```
class matplotlib.axis.Axis(axes, pickradius=15)
```

Public attributes

- `axes.transData` - transform data coords to display coords
- `axes.transAxes` - transform axis coords to display coords
- `labelpad` - number of points between the axis and its label

Init the axis with the parent Axes instance

```
class matplotlib.axis.XAxis(axes, pickradius=15)
```

Init the axis with the parent Axes instance

```
class matplotlib.axis.YAxis(axes, pickradius=15)
```

Init the axis with the parent Axes instance

```
class matplotlib.axis.Ticker
```

`Axis.cla`

clear the current axis

`Axis.get_scale`

36.2.1 matplotlib.axis.Axis.cla

Axis.cla()
clear the current axis

36.2.2 matplotlib.axis.Axis.get__scale

Axis.get__scale()

36.2.3 Formatters and Locators

Axis.get__major__formatter	Get the formatter of the major ticker
Axis.get__major__locator	Get the locator of the major ticker
Axis.get__minor__formatter	Get the formatter of the minor ticker
Axis.get__minor__locator	Get the locator of the minor ticker
Axis.set__major__formatter	Set the formatter of the major ticker
Axis.set__major__locator	Set the locator of the major ticker
Axis.set__minor__formatter	Set the formatter of the minor ticker
Axis.set__minor__locator	Set the locator of the minor ticker

matplotlib.axis.Axis.get__major__formatter

Axis.get__major__formatter()
Get the formatter of the major ticker

matplotlib.axis.Axis.get__major__locator

Axis.get__major__locator()
Get the locator of the major ticker

matplotlib.axis.Axis.get__minor__formatter

Axis.get__minor__formatter()
Get the formatter of the minor ticker

matplotlib.axis.Axis.get__minor__locator

Axis.get__minor__locator()
Get the locator of the minor ticker

`matplotlib.axis.Axis.set__major_formatter`

`Axis.set__major_formatter(formatter)`
Set the formatter of the major ticker
ACCEPTS: A Formatter instance

`matplotlib.axis.Axis.set__major_locator`

`Axis.set__major_locator(locator)`
Set the locator of the major ticker
ACCEPTS: a Locator instance

`matplotlib.axis.Axis.set__minor_formatter`

`Axis.set__minor_formatter(formatter)`
Set the formatter of the minor ticker
ACCEPTS: A Formatter instance

`matplotlib.axis.Axis.set__minor_locator`

`Axis.set__minor_locator(locator)`
Set the locator of the minor ticker
ACCEPTS: a Locator instance

36.2.4 Axis Label

<code>Axis.set_label_coords</code>	Set the coordinates of the label.
<code>Axis.set_label_position</code>	Set the label position (top or bottom)
<code>Axis.set_label_text</code>	Sets the text value of the axis label
<code>Axis.get_label_position</code>	Return the label position (top or bottom)
<code>Axis.get_label_text</code>	Get the text of the label

`matplotlib.axis.Axis.set_label_coords`

`Axis.set_label_coords(x, y, transform=None)`
Set the coordinates of the label. By default, the x coordinate of the y label is determined by the tick label bounding boxes, but this can lead to poor alignment of multiple ylabels if there are multiple axes. Ditto for the y coordinate of the x label.

You can also specify the coordinate system of the label with the transform. If None, the default coordinate system will be the axes coordinate system (0,0) is (left,bottom), (0.5, 0.5) is middle, etc

matplotlib.axis.Axis.set_label_position

Axis.set_label_position(position)
 Set the label position (top or bottom)
 ACCEPTS: ['top' | 'bottom']

matplotlib.axis.Axis.set_label_text

Axis.set_label_text(label, fontdict=None, **kwargs)
 Sets the text value of the axis label
 ACCEPTS: A string value for the label

matplotlib.axis.Axis.get_label_position

Axis.get_label_position()
 Return the label position (top or bottom)

matplotlib.axis.Axis.get_label_text

Axis.get_label_text()
 Get the text of the label

36.2.5 Ticks, tick labels and Offset text

Axis.get_major_ticks	get the tick instances; grow as necessary
Axis.get_majorticklabels	Return a list of Text instances for the major ticklabels
Axis.get_majorticklines	Return the major tick lines as a list of Line2D instances
Axis.get_majorticklocs	Get the major tick locations in data coordinates as a numpy array
Axis.get_minor_ticks	get the minor tick instances; grow as necessary
Axis.get_minorticklabels	Return a list of Text instances for the minor ticklabels
Axis.get_minorticklines	Return the minor tick lines as a list of Line2D instances
Axis.get_minorticklocs	Get the minor tick locations in data coordinates as a numpy array
Axis.get_offset_text	Return the axis offsetText as a Text instance
Axis.get_tick_padding	
Axis.get_ticklabels	Get the tick labels as a list of Text instances.

Continued on next page

Table 4 – continued from previous page

<code>Axis.get_ticklines</code>	Return the tick lines as a list of Line2D instances
<code>Axis.get_ticklocs</code>	Get the tick locations in data coordinates as a numpy array
<code>Axis.get_gridlines</code>	Return the grid lines as a list of Line2D instance
<code>Axis.grid</code>	Set the axis grid on or off; b is a boolean.
<code>Axis.iter_ticks</code>	Iterate through all of the major and minor ticks.
<code>Axis.set_tick_params</code>	Set appearance parameters for ticks, ticklabels, and gridlines.
<code>Axis.axis_date</code>	Sets up x-axis ticks and labels that treat the x data as dates.

`matplotlib.axis.Axis.get_major_ticks`

`Axis.get_major_ticks(numticks=None)`
get the tick instances; grow as necessary

`matplotlib.axis.Axis.get_majorticklabels`

`Axis.get_majorticklabels()`
Return a list of Text instances for the major ticklabels

`matplotlib.axis.Axis.get_majorticklines`

`Axis.get_majorticklines()`
Return the major tick lines as a list of Line2D instances

`matplotlib.axis.Axis.get_majorticklocs`

`Axis.get_majorticklocs()`
Get the major tick locations in data coordinates as a numpy array

`matplotlib.axis.Axis.get_minor_ticks`

`Axis.get_minor_ticks(numticks=None)`
get the minor tick instances; grow as necessary

`matplotlib.axis.Axis.get_minorticklabels`

`Axis.get_minorticklabels()`
Return a list of Text instances for the minor ticklabels

`matplotlib.axis.Axis.get__minorticklines`

`Axis.get__minorticklines()`

Return the minor tick lines as a list of Line2D instances

`matplotlib.axis.Axis.get__minorticklocs`

`Axis.get__minorticklocs()`

Get the minor tick locations in data coordinates as a numpy array

`matplotlib.axis.Axis.get__offset__text`

`Axis.get__offset__text()`

Return the axis offsetText as a Text instance

`matplotlib.axis.Axis.get__tick__padding`

`Axis.get__tick__padding()`

`matplotlib.axis.Axis.get__ticklabels`

`Axis.get__ticklabels(minor=False, which=None)`

Get the tick labels as a list of Text instances.

Parameters

`minor` [bool] If True return the minor ticklabels, else return the major ticklabels

`which` [None, ('minor', 'major', 'both')] Overrides minor.

Selects which ticklabels to return

Returns

`ret` [list] List of Text instances.

`matplotlib.axis.Axis.get__ticklines`

`Axis.get__ticklines(minor=False)`

Return the tick lines as a list of Line2D instances

`matplotlib.axis.Axis.get__ticklocs`

`Axis.get__ticklocs(minor=False)`

Get the tick locations in data coordinates as a numpy array

`matplotlib.axis.Axis.get__gridlines`

`Axis.get__gridlines()`

Return the grid lines as a list of `Line2D` instance

`matplotlib.axis.Axis.grid`

`Axis.grid(b=None, which='major', **kwargs)`

Set the axis grid on or off; `b` is a boolean. Use `which = 'major' | 'minor' | 'both'` to set the grid for major or minor ticks.

If `b` is `None` and `len(kwargs)==0`, toggle the grid state. If `kwargs` are supplied, it is assumed you want the grid on and `b` will be set to `True`.

`kwargs` are used to set the line properties of the grids, e.g.,

`xax.grid(color='r', linestyle='-', linewidth=2)`

`matplotlib.axis.Axis.iter__ticks`

`Axis.iter__ticks()`

Iterate through all of the major and minor ticks.

`matplotlib.axis.Axis.set__tick__params`

`Axis.set__tick__params(which='major', reset=False, **kw)`

Set appearance parameters for ticks, ticklabels, and gridlines.

For documentation of keyword arguments, see `matplotlib.axes.Axes.tick__params()`.

`matplotlib.axis.Axis.axis__date`

`Axis.axis__date(tz=None)`

Sets up x-axis ticks and labels that treat the x data as dates. `tz` is a `tzinfo` instance or a timezone string. This timezone is used to create date labels.

36.2.6 Data and view intervals

<code>Axis.get__data__interval</code>	return the <code>Interval</code> instance for this axis data limits
<code>Axis.get__view__interval</code>	return the <code>Interval</code> instance for this axis view limits
<code>Axis.set__data__interval</code>	set the axis data limits
<code>Axis.set__view__interval</code>	

matplotlib.axis.Axis.get_data_interval

Axis.get_data_interval()
return the Interval instance for this axis data limits

matplotlib.axis.Axis.get_view_interval

Axis.get_view_interval()
return the Interval instance for this axis view limits

matplotlib.axis.Axis.set_data_interval

Axis.set_data_interval()
set the axis data limits

matplotlib.axis.Axis.set_view_interval

Axis.set_view_interval(vmin, vmax, ignore=False)

36.2.7 Rendering helpers

Axis.get_minpos	
Axis.get_tick_space	Return the estimated number of ticks that can fit on the axis.
Axis.get_ticklabel_extents	Get the extents of the tick labels on either side of the axes.
Axis.get_tightbbox	Return a bounding box that encloses the axis.

matplotlib.axis.Axis.get_minpos

Axis.get_minpos()

matplotlib.axis.Axis.get_tick_space

Axis.get_tick_space()
Return the estimated number of ticks that can fit on the axis.

matplotlib.axis.Axis.get_ticklabel_extents

Axis.get_ticklabel_extents(renderer)
Get the extents of the tick labels on either side of the axes.

matplotlib.axis.Axis.get_tightbbox

Axis.get_tightbbox(renderer)

Return a bounding box that encloses the axis. It only accounts tick labels, axis label, and offsetText.

36.2.8 Interactive

Axis.get_pickradius	Return the depth of the axis used by the picker
Axis.set_pickradius	Set the depth of the axis used by the picker

matplotlib.axis.Axis.get_pickradius

Axis.get_pickradius()

Return the depth of the axis used by the picker

matplotlib.axis.Axis.set_pickradius

Axis.set_pickradius(pickradius)

Set the depth of the axis used by the picker

ACCEPTS: a distance in points

36.2.9 Units

Axis.convert_units	
Axis.set_units	set the units for axis
Axis.get_units	return the units for axis
Axis.update_units	introspect data for units converter and update the axis.converter instance if necessary.

matplotlib.axis.Axis.convert_units

Axis.convert_units(x)

matplotlib.axis.Axis.set_units

Axis.set_units(u)

set the units for axis

ACCEPTS: a units tag

matplotlib.axis.Axis.get__units

Axis.get__units()
return the units for axis

matplotlib.axis.Axis.update__units

Axis.update__units(data)
introspect data for units converter and update the axis.converter instance if necessary. Return True if data is registered for unit conversion.

36.2.10 Incremental navigation

Axis.pan	Pan numsteps (can be positive or negative)
Axis.zoom	Zoom in/out on axis; if direction is >0 zoom in, else zoom out

matplotlib.axis.Axis.pan

Axis.pan(numsteps)
Pan numsteps (can be positive or negative)

matplotlib.axis.Axis.zoom

Axis.zoom(direction)
Zoom in/out on axis; if direction is >0 zoom in, else zoom out

36.2.11 YAxis Specific

YAxis.axis__name	
YAxis.get__text__widths	
YAxis.get__ticks__position	Return the ticks position (left, right, both or unknown)
YAxis.set__offset__position	
YAxis.set__ticks__position	Set the ticks position (left, right, both, default or none) 'both' sets the ticks to appear on both positions, but does not change the tick labels.
YAxis.tick__left	Move ticks and ticklabels (if present) to the left of the axes.
YAxis.tick__right	Move ticks and ticklabels (if present) to the right of the axes.

matplotlib.axis.YAxis.axis__name

YAxis.axis__name = 'y'

matplotlib.axis.YAxis.get__text__widths

YAxis.get__text__widths(renderer)

matplotlib.axis.YAxis.get__ticks__position

YAxis.get__ticks__position()

Return the ticks position (left, right, both or unknown)

matplotlib.axis.YAxis.set__offset__position

YAxis.set__offset__position(position)

matplotlib.axis.YAxis.set__ticks__position

YAxis.set__ticks__position(position)

Set the ticks position (left, right, both, default or none) 'both' sets the ticks to appear on both positions, but does not change the tick labels. 'default' resets the tick positions to the default: ticks on both positions, labels at left. 'none' can be used if you don't want any ticks. 'none' and 'both' affect only the ticks, not the labels.

ACCEPTS: ['left' | 'right' | 'both' | 'default' | 'none']

matplotlib.axis.YAxis.tick__left

YAxis.tick__left()

Move ticks and ticklabels (if present) to the left of the axes.

matplotlib.axis.YAxis.tick__right

YAxis.tick__right()

Move ticks and ticklabels (if present) to the right of the axes.

36.2.12 XAxis Specific

<code>XAxis.axis_name</code>	
<code>XAxis.get_text_heights</code>	Returns the amount of space one should reserve for text above and below the axes.
<code>XAxis.get_ticks_position</code>	Return the ticks position (top, bottom, default or unknown)
<code>XAxis.set_ticks_position</code>	Set the ticks position (top, bottom, both, default or none) both sets the ticks to appear on both positions, but does not change the tick labels.
<code>XAxis.tick_bottom</code>	Move ticks and ticklabels (if present) to the bottom of the axes.
<code>XAxis.tick_top</code>	Move ticks and ticklabels (if present) to the top of the axes.

`matplotlib.axis.XAxis.axis_name`

`XAxis.axis_name = 'x'`

`matplotlib.axis.XAxis.get_text_heights`

`XAxis.get_text_heights(renderer)`

Returns the amount of space one should reserve for text above and below the axes. Returns a tuple (above, below)

`matplotlib.axis.XAxis.get_ticks_position`

`XAxis.get_ticks_position()`

Return the ticks position (top, bottom, default or unknown)

`matplotlib.axis.XAxis.set_ticks_position`

`XAxis.set_ticks_position(position)`

Set the ticks position (top, bottom, both, default or none) both sets the ticks to appear on both positions, but does not change the tick labels. 'default' resets the tick positions to the default: ticks on both positions, labels at bottom. 'none' can be used if you don't want any ticks. 'none' and 'both' affect only the ticks, not the labels.

ACCEPTS: ['top' | 'bottom' | 'both' | 'default' | 'none']

`matplotlib.axis.XAxis.tick_bottom`

`XAxis.tick_bottom()`

Move ticks and ticklabels (if present) to the bottom of the axes.

matplotlib.axis.XAxis.tick_top

XAxis.tick_top()

Move ticks and ticklabels (if present) to the top of the axes.

36.2.13 Other

Axis.OFFSETTEXTPAD	
Axis.limit_range_for_scale	
Axis.reset_ticks	Re-initialize the major and minor Tick lists.
Axis.set_default_intervals	set the default limits for the axis data and view interval if they are not mutated
Axis.get_smart_bounds	get whether the axis has smart bounds
Axis.set_smart_bounds	set the axis to have smart bounds

matplotlib.axis.Axis.OFFSETTEXTPAD

Axis.OFFSETTEXTPAD = 3

matplotlib.axis.Axis.limit_range_for_scale

Axis.limit_range_for_scale(vmin, vmax)

matplotlib.axis.Axis.reset_ticks

Axis.reset_ticks()

Re-initialize the major and minor Tick lists.

Each list starts with a single fresh Tick.

matplotlib.axis.Axis.set_default_intervals

Axis.set_default_intervals()

set the default limits for the axis data and view interval if they are not mutated

matplotlib.axis.Axis.get_smart_bounds

Axis.get_smart_bounds()

get whether the axis has smart bounds

matplotlib.axis.Axis.set_smart_bounds

Axis.set_smart_bounds(value)
 set the axis to have smart bounds

36.2.14 Discouraged

These methods implicitly use FixedLocator and FixedFormatter. They can be convenient, but if not used together may de-couple your tick labels from your data.

Axis.set_ticklabels	Set the text values of the tick labels.
Axis.set_ticks	Set the locations of the tick marks from sequence ticks

matplotlib.axis.Axis.set_ticklabels

Axis.set_ticklabels(ticklabels, *args, **kwargs)
 Set the text values of the tick labels. Return a list of Text instances. Use kwarg minor=True to select minor ticks. All other kwargs are used to update the text object properties. As for get_ticklabels, label1 (left or bottom) is affected for a given tick only if its label1On attribute is True, and similarly for label2. The list of returned label text objects consists of all such label1 objects followed by all such label2 objects.

The input ticklabels is assumed to match the set of tick locations, regardless of the state of label1On and label2On.

ACCEPTS: sequence of strings or Text objects

matplotlib.axis.Axis.set_ticks

Axis.set_ticks(ticks, minor=False)
 Set the locations of the tick marks from sequence ticks

ACCEPTS: sequence of floats

36.3 Tick objects

class matplotlib.axis.Tick(axes, loc, label, size=None, width=None, color=None, tickdir=None, pad=None, labelsz=None, labelcolor=None, zorder=None, gridOn=None, tick1On=True, tick2On=True, label1On=True, label2On=False, major=True, labelrotation=0, grid_color=None, grid_linestyle=None, grid_linewidth=None, grid_alpha=None, **kw)
 Abstract base class for the axis ticks, grid lines and labels

1 refers to the bottom of the plot for xticks and the left for yticks 2 refers to the top of the plot for xticks and the right for yticks

Attributes

tick1line [Line2D]
tick2line [Line2D]
gridline [Line2D]
label1 [Text]
label2 [Text]
gridOn [bool] Determines whether to draw the tickline.
tick1On [bool] Determines whether to draw the first tickline.
tick2On [bool] Determines whether to draw the second tickline.
label1On [bool] Determines whether to draw the first tick label.
label2On [bool] Determines whether to draw the second tick label.

bbox is the Bound2D bounding box in display coords of the Axes loc is the tick location in data coords size is the tick size in points

```
class matplotlib.axis.XTick(axes, loc, label, size=None, width=None, color=None,
                             tickdir=None, pad=None, labelsiz=None, labelcolor=None,
                             zorder=None, gridOn=None, tick1On=True, tick2On=True, la-
                             bel1On=True, label2On=False, major=True, labelrotation=0,
                             grid_color=None, grid_linestyle=None, grid_linewidth=None,
                             grid_alpha=None, **kw)
```

Contains all the Artists needed to make an x tick - the tick line, the label text and the grid line

bbox is the Bound2D bounding box in display coords of the Axes loc is the tick location in data coords size is the tick size in points

```
class matplotlib.axis.YTick(axes, loc, label, size=None, width=None, color=None,
                             tickdir=None, pad=None, labelsiz=None, labelcolor=None,
                             zorder=None, gridOn=None, tick1On=True, tick2On=True, la-
                             bel1On=True, label2On=False, major=True, labelrotation=0,
                             grid_color=None, grid_linestyle=None, grid_linewidth=None,
                             grid_alpha=None, **kw)
```

Contains all the Artists needed to make a Y tick - the tick line, the label text and the grid line

bbox is the Bound2D bounding box in display coords of the Axes loc is the tick location in data coords size is the tick size in points

Tick.apply_tickdir

Calculate self.__pad and self.__tickmarkers

Continued on next page

Table 14 – continued from previous page

Tick.get_loc	Return the tick location (data coords) as a scalar
Tick.get_pad	Get the value of the tick label pad in points
Tick.get_pad_pixels	
Tick.get_tick_padding	Get the length of the tick outside of the axes.
Tick.get_tickdir	
Tick.get_view_interval	return the view Interval instance for the axis this tick is ticking
Tick.set_label1	Set the text of ticklabel
Tick.set_label2	Set the text of ticklabel2
Tick.set_pad	Set the tick label pad in points
Tick.update_position	Set the location of tick in data coords with scalar loc

36.3.1 matplotlib.axis.Tick.apply_tickdir

Tick.apply_tickdir(tickdir)
 Calculate self.__pad and self.__tickmarkers

36.3.2 matplotlib.axis.Tick.get_loc

Tick.get_loc()
 Return the tick location (data coords) as a scalar

36.3.3 matplotlib.axis.Tick.get_pad

Tick.get_pad()
 Get the value of the tick label pad in points

36.3.4 matplotlib.axis.Tick.get_pad_pixels

Tick.get_pad_pixels()

36.3.5 matplotlib.axis.Tick.get_tick_padding

Tick.get_tick_padding()
 Get the length of the tick outside of the axes.

36.3.6 matplotlib.axis.Tick.get_tickdir

Tick.get_tickdir()

36.3.7 matplotlib.axis.Tick.get_view_interval

Tick.get_view_interval()

return the view Interval instance for the axis this tick is ticking

36.3.8 matplotlib.axis.Tick.set_label1

Tick.set_label1(s)

Set the text of ticklabel

ACCEPTS: str

36.3.9 matplotlib.axis.Tick.set_label2

Tick.set_label2(s)

Set the text of ticklabel2

ACCEPTS: str

36.3.10 matplotlib.axis.Tick.set_pad

Tick.set_pad(val)

Set the tick label pad in points

ACCEPTS: float

36.3.11 matplotlib.axis.Tick.update_position

Tick.update_position(loc)

Set the location of tick in data coords with scalar loc

36.4 Common and inherited methods

36.4.1 XTick

XTick.apply_tickdir

Calculate self._pad and self._tickmarkers

Continued on next page

Table 15 – continued from previous page

<code>XTick.get_loc</code>	Return the tick location (data coords) as a scalar
<code>XTick.get_pad</code>	Get the value of the tick label pad in points
<code>XTick.get_pad_pixels</code>	
<code>XTick.get_tick_padding</code>	Get the length of the tick outside of the axes.
<code>XTick.get_tickdir</code>	
<code>XTick.get_view_interval</code>	return the Interval instance for this axis view limits
<code>XTick.set_label1</code>	Set the text of ticklabel
<code>XTick.set_label2</code>	Set the text of ticklabel2
<code>XTick.set_pad</code>	Set the tick label pad in points
<code>XTick.update_position</code>	Set the location of tick in data coords with scalar loc

`matplotlib.axis.XTick.apply_tickdir`

`XTick.apply_tickdir(tickdir)`
 Calculate self._pad and self._tickmarkers

`matplotlib.axis.XTick.get_loc`

`XTick.get_loc()`
 Return the tick location (data coords) as a scalar

`matplotlib.axis.XTick.get_pad`

`XTick.get_pad()`
 Get the value of the tick label pad in points

`matplotlib.axis.XTick.get_pad_pixels`

`XTick.get_pad_pixels()`

`matplotlib.axis.XTick.get_tick_padding`

`XTick.get_tick_padding()`
 Get the length of the tick outside of the axes.

`matplotlib.axis.XTick.get_tickdir`

`XTick.get_tickdir()`

matplotlib.axis.XTick.get_view_interval

XTick.get_view_interval()
return the Interval instance for this axis view limits

matplotlib.axis.XTick.set_label1

XTick.set_label1(s)
Set the text of ticklabel
ACCEPTS: str

matplotlib.axis.XTick.set_label2

XTick.set_label2(s)
Set the text of ticklabel2
ACCEPTS: str

matplotlib.axis.XTick.set_pad

XTick.set_pad(val)
Set the tick label pad in points
ACCEPTS: float

matplotlib.axis.XTick.update_position

XTick.update_position(loc)
Set the location of tick in data coords with scalar loc

36.4.2 YTick

YTick.apply_tickdir	Calculate self._pad and self._tickmarkers
YTick.get_loc	Return the tick location (data coords) as a scalar
YTick.get_pad	Get the value of the tick label pad in points
YTick.get_pad_pixels	
YTick.get_tick_padding	Get the length of the tick outside of the axes.
YTick.get_tickdir	
YTick.get_view_interval	return the Interval instance for this axis view limits
YTick.set_label1	Set the text of ticklabel
YTick.set_label2	Set the text of ticklabel2

Continued on next page

Table 16 – continued from previous page

<code>YTick.set_pad</code>	Set the tick label pad in points
<code>YTick.update_position</code>	Set the location of tick in data coords with scalar loc

`matplotlib.axis.YTick.apply_tickdir`

`YTick.apply_tickdir(tickdir)`
Calculate self._pad and self._tickmarkers

`matplotlib.axis.YTick.get_loc`

`YTick.get_loc()`
Return the tick location (data coords) as a scalar

`matplotlib.axis.YTick.get_pad`

`YTick.get_pad()`
Get the value of the tick label pad in points

`matplotlib.axis.YTick.get_pad_pixels`

`YTick.get_pad_pixels()`

`matplotlib.axis.YTick.get_tick_padding`

`YTick.get_tick_padding()`
Get the length of the tick outside of the axes.

`matplotlib.axis.YTick.get_tickdir`

`YTick.get_tickdir()`

`matplotlib.axis.YTick.get_view_interval`

`YTick.get_view_interval()`
return the Interval instance for this axis view limits

matplotlib.axis.YTick.set_label1

YTick.set_label1(s)

Set the text of ticklabel

ACCEPTS: str

matplotlib.axis.YTick.set_label2

YTick.set_label2(s)

Set the text of ticklabel2

ACCEPTS: str

matplotlib.axis.YTick.set_pad

YTick.set_pad(val)

Set the tick label pad in points

ACCEPTS: float

matplotlib.axis.YTick.update_position

YTick.update_position(loc)

Set the location of tick in data coords with scalar loc

36.4.3 YAxis

YAxis.OFFSETTEXTPAD

YAxis.axis_date	Sets up x-axis ticks and labels that treat the x data as dates.
YAxis.cla	clear the current axis
YAxis.convert_units	
YAxis.get_data_interval	return the Interval instance for this axis data limits
YAxis.get_gridlines	Return the grid lines as a list of Line2D instance
YAxis.get_label_position	Return the label position (top or bottom)
YAxis.get_label_text	Get the text of the label
YAxis.get_major_formatter	Get the formatter of the major ticker
YAxis.get_major_locator	Get the locator of the major ticker
YAxis.get_major_ticks	get the tick instances; grow as necessary
YAxis.get_majorticklabels	Return a list of Text instances for the major ticklabels

Continued on next page

Table 17 – continued from previous page

<code>YAxis.get_majorticklines</code>	Return the major tick lines as a list of <code>Line2D</code> instances
<code>YAxis.get_majorticklocs</code>	Get the major tick locations in data coordinates as a numpy array
<code>YAxis.get_minor_formatter</code>	Get the formatter of the minor ticker
<code>YAxis.get_minor_locator</code>	Get the locator of the minor ticker
<code>YAxis.get_minor_ticks</code>	get the minor tick instances; grow as necessary
<code>YAxis.get_minorticklabels</code>	Return a list of <code>Text</code> instances for the minor ticklabels
<code>YAxis.get_minorticklines</code>	Return the minor tick lines as a list of <code>Line2D</code> instances
<code>YAxis.get_minorticklocs</code>	Get the minor tick locations in data coordinates as a numpy array
<code>YAxis.get_minpos</code>	
<code>YAxis.get_offset_text</code>	Return the axis offsetText as a <code>Text</code> instance
<code>YAxis.get_pickradius</code>	Return the depth of the axis used by the picker
<code>YAxis.get_scale</code>	
<code>YAxis.get_smart_bounds</code>	get whether the axis has smart bounds
<code>YAxis.get_tick_padding</code>	
<code>YAxis.get_tick_space</code>	Return the estimated number of ticks that can fit on the axis.
<code>YAxis.get_ticklabel_extents</code>	Get the extents of the tick labels on either side of the axes.
<code>YAxis.get_ticklabels</code>	Get the tick labels as a list of <code>Text</code> instances.
<code>YAxis.get_ticklines</code>	Return the tick lines as a list of <code>Line2D</code> instances
<code>YAxis.get_ticklocs</code>	Get the tick locations in data coordinates as a numpy array
<code>YAxis.get_tightbbox</code>	Return a bounding box that encloses the axis.
<code>YAxis.get_units</code>	return the units for axis
<code>YAxis.get_view_interval</code>	return the <code>Interval</code> instance for this axis view limits
<code>YAxis.grid</code>	Set the axis grid on or off; b is a boolean.
<code>YAxis.iter_ticks</code>	Iterate through all of the major and minor ticks.
<code>YAxis.limit_range_for_scale</code>	
<code>YAxis.pan</code>	Pan numsteps (can be positive or negative)
<code>YAxis.reset_ticks</code>	Re-initialize the major and minor Tick lists.
<code>YAxis.set_data_interval</code>	set the axis data limits
<code>YAxis.set_default_intervals</code>	set the default limits for the axis interval if they are not mutated
<code>YAxis.set_label_coords</code>	Set the coordinates of the label.
<code>YAxis.set_label_position</code>	Set the label position (left or right)
<code>YAxis.set_label_text</code>	Sets the text value of the axis label
<code>YAxis.set_major_formatter</code>	Set the formatter of the major ticker

Continued on next page

Table 17 – continued from previous page

<code>YAxis.set_major_locator</code>	Set the locator of the major ticker
<code>YAxis.set_minor_formatter</code>	Set the formatter of the minor ticker
<code>YAxis.set_minor_locator</code>	Set the locator of the minor ticker
<code>YAxis.set_pickradius</code>	Set the depth of the axis used by the picker
<code>YAxis.set_smart_bounds</code>	set the axis to have smart bounds
<code>YAxis.set_tick_params</code>	Set appearance parameters for ticks, ticklabels, and gridlines.
<code>YAxis.set_ticklabels</code>	Set the text values of the tick labels.
<code>YAxis.set_ticks</code>	Set the locations of the tick marks from sequence ticks
<code>YAxis.set_units</code>	set the units for axis
<code>YAxis.set_view_interval</code>	If ignore is False, the order of vmin, vmax does not matter; the original axis orientation will be preserved.
<code>YAxis.update_units</code>	introspect data for units converter and update the axis.converter instance if necessary.
<code>YAxis.zoom</code>	Zoom in/out on axis; if direction is >0 zoom in, else zoom out

`matplotlib.axis.YAxis.OFFSETTEXTPAD`

`YAxis.OFFSETTEXTPAD = 3`

`matplotlib.axis.YAxis.axis_date`

`YAxis.axis_date(tz=None)`

Sets up x-axis ticks and labels that treat the x data as dates. `tz` is a `tzinfo` instance or a timezone string. This timezone is used to create date labels.

`matplotlib.axis.YAxis.cla`

`YAxis.cla()`

clear the current axis

`matplotlib.axis.YAxis.convert_units`

`YAxis.convert_units(x)`

`matplotlib.axis.YAxis.get_data_interval`

`YAxis.get_data_interval()`

return the Interval instance for this axis data limits

`matplotlib.axis.YAxis.get__gridlines`

`YAxis.get__gridlines()`

Return the grid lines as a list of Line2D instance

`matplotlib.axis.YAxis.get__label__position`

`YAxis.get__label__position()`

Return the label position (top or bottom)

`matplotlib.axis.YAxis.get__label__text`

`YAxis.get__label__text()`

Get the text of the label

`matplotlib.axis.YAxis.get__major__formatter`

`YAxis.get__major__formatter()`

Get the formatter of the major ticker

`matplotlib.axis.YAxis.get__major__locator`

`YAxis.get__major__locator()`

Get the locator of the major ticker

`matplotlib.axis.YAxis.get__major__ticks`

`YAxis.get__major__ticks(numticks=None)`

get the tick instances; grow as necessary

`matplotlib.axis.YAxis.get__majorticklabels`

`YAxis.get__majorticklabels()`

Return a list of Text instances for the major ticklabels

`matplotlib.axis.YAxis.get__majorticklines`

`YAxis.get__majorticklines()`

Return the major tick lines as a list of Line2D instances

`matplotlib.axis.YAxis.get_majorticklocs`

`YAxis.get_majorticklocs()`

Get the major tick locations in data coordinates as a numpy array

`matplotlib.axis.YAxis.get_minor_formatter`

`YAxis.get_minor_formatter()`

Get the formatter of the minor ticker

`matplotlib.axis.YAxis.get_minor_locator`

`YAxis.get_minor_locator()`

Get the locator of the minor ticker

`matplotlib.axis.YAxis.get_minor_ticks`

`YAxis.get_minor_ticks(numticks=None)`

get the minor tick instances; grow as necessary

`matplotlib.axis.YAxis.get_minorticklabels`

`YAxis.get_minorticklabels()`

Return a list of Text instances for the minor ticklabels

`matplotlib.axis.YAxis.get_minorticklines`

`YAxis.get_minorticklines()`

Return the minor tick lines as a list of Line2D instances

`matplotlib.axis.YAxis.get_minorticklocs`

`YAxis.get_minorticklocs()`

Get the minor tick locations in data coordinates as a numpy array

`matplotlib.axis.YAxis.get_minpos`

`YAxis.get_minpos()`

`matplotlib.axis.YAxis.get__offset__text`

`YAxis.get__offset__text()`

Return the axis offsetText as a Text instance

`matplotlib.axis.YAxis.get__pickradius`

`YAxis.get__pickradius()`

Return the depth of the axis used by the picker

`matplotlib.axis.YAxis.get__scale`

`YAxis.get__scale()`

`matplotlib.axis.YAxis.get__smart__bounds`

`YAxis.get__smart__bounds()`

get whether the axis has smart bounds

`matplotlib.axis.YAxis.get__tick__padding`

`YAxis.get__tick__padding()`

`matplotlib.axis.YAxis.get__tick__space`

`YAxis.get__tick__space()`

Return the estimated number of ticks that can fit on the axis.

`matplotlib.axis.YAxis.get__ticklabel__extents`

`YAxis.get__ticklabel__extents(renderer)`

Get the extents of the tick labels on either side of the axes.

`matplotlib.axis.YAxis.get__ticklabels`

`YAxis.get__ticklabels(minor=False, which=None)`

Get the tick labels as a list of Text instances.

Parameters

`minor` [bool] If True return the minor ticklabels, else return the major ticklabels

which [None, ('minor', 'major', 'both')] Overrides minor.

Selects which ticklabels to return

Returns

ret [list] List of Text instances.

`matplotlib.axis.YAxis.get_ticklines`

`YAxis.get_ticklines(minor=False)`

Return the tick lines as a list of Line2D instances

`matplotlib.axis.YAxis.get_ticklocs`

`YAxis.get_ticklocs(minor=False)`

Get the tick locations in data coordinates as a numpy array

`matplotlib.axis.YAxis.get_tightbbox`

`YAxis.get_tightbbox(renderer)`

Return a bounding box that encloses the axis. It only accounts tick labels, axis label, and offsetText.

`matplotlib.axis.YAxis.get_units`

`YAxis.get_units()`

return the units for axis

`matplotlib.axis.YAxis.get_view_interval`

`YAxis.get_view_interval()`

return the Interval instance for this axis view limits

`matplotlib.axis.YAxis.grid`

`YAxis.grid(b=None, which='major', **kwargs)`

Set the axis grid on or off; b is a boolean. Use which = 'major' | 'minor' | 'both' to set the grid for major or minor ticks.

If b is None and len(kwargs)==0, toggle the grid state. If kwargs are supplied, it is assumed you want the grid on and b will be set to True.

kwargs are used to set the line properties of the grids, e.g.,

`xax.grid(color='r', linestyle='-', linewidth=2)`

`matplotlib.axis.YAxis.iter_ticks`

`YAxis.iter_ticks()`

Iterate through all of the major and minor ticks.

`matplotlib.axis.YAxis.limit_range_for_scale`

`YAxis.limit_range_for_scale(vmin, vmax)`

`matplotlib.axis.YAxis.pan`

`YAxis.pan(numsteps)`

Pan numsteps (can be positive or negative)

`matplotlib.axis.YAxis.reset_ticks`

`YAxis.reset_ticks()`

Re-initialize the major and minor Tick lists.

Each list starts with a single fresh Tick.

`matplotlib.axis.YAxis.set_data_interval`

`YAxis.set_data_interval(vmin, vmax, ignore=False)`

set the axis data limits

`matplotlib.axis.YAxis.set_default_intervals`

`YAxis.set_default_intervals()`

set the default limits for the axis interval if they are not mutated

`matplotlib.axis.YAxis.set_label_coords`

`YAxis.set_label_coords(x, y, transform=None)`

Set the coordinates of the label. By default, the x coordinate of the y label is determined by the tick label bounding boxes, but this can lead to poor alignment of multiple ylabels if there are multiple axes. Ditto for the y coordinate of the x label.

You can also specify the coordinate system of the label with the transform. If None, the default coordinate system will be the axes coordinate system (0,0) is (left,bottom), (0.5, 0.5) is middle, etc

`matplotlib.axis.YAxis.set_label_position`

`YAxis.set_label_position(position)`
Set the label position (left or right)
ACCEPTS: ['left' | 'right']

`matplotlib.axis.YAxis.set_label_text`

`YAxis.set_label_text(label, fontdict=None, **kwargs)`
Sets the text value of the axis label
ACCEPTS: A string value for the label

`matplotlib.axis.YAxis.set_major_formatter`

`YAxis.set_major_formatter(formatter)`
Set the formatter of the major ticker
ACCEPTS: A Formatter instance

`matplotlib.axis.YAxis.set_major_locator`

`YAxis.set_major_locator(locator)`
Set the locator of the major ticker
ACCEPTS: a Locator instance

`matplotlib.axis.YAxis.set_minor_formatter`

`YAxis.set_minor_formatter(formatter)`
Set the formatter of the minor ticker
ACCEPTS: A Formatter instance

`matplotlib.axis.YAxis.set_minor_locator`

`YAxis.set_minor_locator(locator)`
Set the locator of the minor ticker
ACCEPTS: a Locator instance

`matplotlib.axis.YAxis.set_pickradius`

`YAxis.set_pickradius(pickradius)`
Set the depth of the axis used by the picker

ACCEPTS: a distance in points

`matplotlib.axis.YAxis.set_smart_bounds`

`YAxis.set_smart_bounds(value)`
set the axis to have smart bounds

`matplotlib.axis.YAxis.set_tick_params`

`YAxis.set_tick_params(which='major', reset=False, **kw)`
Set appearance parameters for ticks, ticklabels, and gridlines.
For documentation of keyword arguments, see `matplotlib.axes.Axes.tick_params()`.

`matplotlib.axis.YAxis.set_ticklabels`

`YAxis.set_ticklabels(ticklabels, *args, **kwargs)`
Set the text values of the tick labels. Return a list of Text instances. Use kwarg `minor=True` to select minor ticks. All other kwargs are used to update the text object properties. As for `get_ticklabels`, `label1` (left or bottom) is affected for a given tick only if its `label1On` attribute is True, and similarly for `label2`. The list of returned label text objects consists of all such `label1` objects followed by all such `label2` objects.

The input `ticklabels` is assumed to match the set of tick locations, regardless of the state of `label1On` and `label2On`.

ACCEPTS: sequence of strings or Text objects

`matplotlib.axis.YAxis.set_ticks`

`YAxis.set_ticks(ticks, minor=False)`
Set the locations of the tick marks from sequence ticks

ACCEPTS: sequence of floats

`matplotlib.axis.YAxis.set_units`

`YAxis.set_units(u)`
set the units for axis

ACCEPTS: a units tag

matplotlib.axis.YAxis.set_view_interval

YAxis.set_view_interval(vmin, vmax, ignore=False)

If ignore is False, the order of vmin, vmax does not matter; the original axis orientation will be preserved. In addition, the view limits can be expanded, but will not be reduced. This method is for mpl internal use; for normal use, see set_ylim().

matplotlib.axis.YAxis.update_units

YAxis.update_units(data)

introspect data for units converter and update the axis.converter instance if necessary. Return True if data is registered for unit conversion.

matplotlib.axis.YAxis.zoom

YAxis.zoom(direction)

Zoom in/out on axis; if direction is >0 zoom in, else zoom out

36.4.4 XAxis

XAxis.OFFSETTEXTPAD	
XAxis.axis_date	Sets up x-axis ticks and labels that treat the x data as dates.
XAxis.cla	clear the current axis
XAxis.convert_units	
XAxis.get_data_interval	return the Interval instance for this axis data limits
XAxis.get_gridlines	Return the grid lines as a list of Line2D instance
XAxis.get_label_position	Return the label position (top or bottom)
XAxis.get_label_text	Get the text of the label
XAxis.get_major_formatter	Get the formatter of the major ticker
XAxis.get_major_locator	Get the locator of the major ticker
XAxis.get_major_ticks	get the tick instances; grow as necessary
XAxis.get_majorticklabels	Return a list of Text instances for the major ticklabels
XAxis.get_majorticklines	Return the major tick lines as a list of Line2D instances
XAxis.get_majorticklocs	Get the major tick locations in data coordinates as a numpy array
XAxis.get_minor_formatter	Get the formatter of the minor ticker
XAxis.get_minor_locator	Get the locator of the minor ticker
XAxis.get_minor_ticks	get the minor tick instances; grow as necessary

Continued on next page

Table 18 – continued from previous page

XAxis.get_minorticklabels	Return a list of Text instances for the minor ticklabels
XAxis.get_minorticklines	Return the minor tick lines as a list of Line2D instances
XAxis.get_minorticklocs	Get the minor tick locations in data coordinates as a numpy array
XAxis.get_minpos	
XAxis.get_offset_text	Return the axis offsetText as a Text instance
XAxis.get_pickradius	Return the depth of the axis used by the picker
XAxis.get_scale	
XAxis.get_smart_bounds	get whether the axis has smart bounds
XAxis.get_tick_padding	
XAxis.get_tick_space	Return the estimated number of ticks that can fit on the axis.
XAxis.get_ticklabel_extents	Get the extents of the tick labels on either side of the axes.
XAxis.get_ticklabels	Get the tick labels as a list of Text instances.
XAxis.get_ticklines	Return the tick lines as a list of Line2D instances
XAxis.get_ticklocs	Get the tick locations in data coordinates as a numpy array
XAxis.get_tightbbox	Return a bounding box that encloses the axis.
XAxis.get_units	return the units for axis
XAxis.get_view_interval	return the Interval instance for this axis view limits
XAxis.grid	Set the axis grid on or off; b is a boolean.
XAxis.iter_ticks	Iterate through all of the major and minor ticks.
XAxis.limit_range_for_scale	
XAxis.pan	Pan numsteps (can be positive or negative)
XAxis.reset_ticks	Re-initialize the major and minor Tick lists.
XAxis.set_data_interval	set the axis data limits
XAxis.set_default_intervals	set the default limits for the axis interval if they are not mutated
XAxis.set_label_coords	Set the coordinates of the label.
XAxis.set_label_position	Set the label position (top or bottom)
XAxis.set_label_text	Sets the text value of the axis label
XAxis.set_major_formatter	Set the formatter of the major ticker
XAxis.set_major_locator	Set the locator of the major ticker
XAxis.set_minor_formatter	Set the formatter of the minor ticker
XAxis.set_minor_locator	Set the locator of the minor ticker
XAxis.set_pickradius	Set the depth of the axis used by the picker
XAxis.set_smart_bounds	set the axis to have smart bounds
XAxis.set_tick_params	Set appearance parameters for ticks, ticklabels, and gridlines.

Continued on next page

Table 18 – continued from previous page

<code>XAxis.set_ticklabels</code>	Set the text values of the tick labels.
<code>XAxis.set_ticks</code>	Set the locations of the tick marks from sequence ticks
<code>XAxis.set_units</code>	set the units for axis
<code>XAxis.set_view_interval</code>	If ignore is False, the order of vmin, vmax does not matter; the original axis orientation will be preserved.
<code>XAxis.update_units</code>	introspect data for units converter and update the axis.converter instance if necessary.
<code>XAxis.zoom</code>	Zoom in/out on axis; if direction is >0 zoom in, else zoom out

`matplotlib.axis.XAxis.OFFSETTEXTPAD`

`XAxis.OFFSETTEXTPAD = 3`

`matplotlib.axis.XAxis.axis_date`

`XAxis.axis_date(tz=None)`

Sets up x-axis ticks and labels that treat the x data as dates. `tz` is a `tzinfo` instance or a timezone string. This timezone is used to create date labels.

`matplotlib.axis.XAxis.cla`

`XAxis.cla()`

clear the current axis

`matplotlib.axis.XAxis.convert_units`

`XAxis.convert_units(x)`

`matplotlib.axis.XAxis.get_data_interval`

`XAxis.get_data_interval()`

return the Interval instance for this axis data limits

`matplotlib.axis.XAxis.get_gridlines`

`XAxis.get_gridlines()`

Return the grid lines as a list of Line2D instance

`matplotlib.axis.XAxis.get_label_position`

`XAxis.get_label_position()`

Return the label position (top or bottom)

`matplotlib.axis.XAxis.get_label_text`

`XAxis.get_label_text()`

Get the text of the label

`matplotlib.axis.XAxis.get_major_formatter`

`XAxis.get_major_formatter()`

Get the formatter of the major ticker

`matplotlib.axis.XAxis.get_major_locator`

`XAxis.get_major_locator()`

Get the locator of the major ticker

`matplotlib.axis.XAxis.get_major_ticks`

`XAxis.get_major_ticks(numticks=None)`

get the tick instances; grow as necessary

`matplotlib.axis.XAxis.get_majorticklabels`

`XAxis.get_majorticklabels()`

Return a list of Text instances for the major ticklabels

`matplotlib.axis.XAxis.get_majorticklines`

`XAxis.get_majorticklines()`

Return the major tick lines as a list of Line2D instances

`matplotlib.axis.XAxis.get_majorticklocs`

`XAxis.get_majorticklocs()`

Get the major tick locations in data coordinates as a numpy array

`matplotlib.axis.XAxis.get__minor__formatter`

`XAxis.get__minor__formatter()`

Get the formatter of the minor ticker

`matplotlib.axis.XAxis.get__minor__locator`

`XAxis.get__minor__locator()`

Get the locator of the minor ticker

`matplotlib.axis.XAxis.get__minor__ticks`

`XAxis.get__minor__ticks(numticks=None)`

get the minor tick instances; grow as necessary

`matplotlib.axis.XAxis.get__minorticklabels`

`XAxis.get__minorticklabels()`

Return a list of Text instances for the minor ticklabels

`matplotlib.axis.XAxis.get__minorticklines`

`XAxis.get__minorticklines()`

Return the minor tick lines as a list of Line2D instances

`matplotlib.axis.XAxis.get__minorticklocs`

`XAxis.get__minorticklocs()`

Get the minor tick locations in data coordinates as a numpy array

`matplotlib.axis.XAxis.get__minpos`

`XAxis.get__minpos()`

`matplotlib.axis.XAxis.get__offset__text`

`XAxis.get__offset__text()`

Return the axis offsetText as a Text instance

`matplotlib.axis.XAxis.get_pickradius`

`XAxis.get_pickradius()`

Return the depth of the axis used by the picker

`matplotlib.axis.XAxis.get__scale`

`XAxis.get__scale()`

`matplotlib.axis.XAxis.get_smart_bounds`

`XAxis.get_smart_bounds()`

get whether the axis has smart bounds

`matplotlib.axis.XAxis.get_tick_padding`

`XAxis.get_tick_padding()`

`matplotlib.axis.XAxis.get_tick_space`

`XAxis.get_tick_space()`

Return the estimated number of ticks that can fit on the axis.

`matplotlib.axis.XAxis.get_ticklabel_extents`

`XAxis.get_ticklabel_extents(renderer)`

Get the extents of the tick labels on either side of the axes.

`matplotlib.axis.XAxis.get_ticklabels`

`XAxis.get_ticklabels(minor=False, which=None)`

Get the tick labels as a list of Text instances.

Parameters

`minor` [bool] If True return the minor ticklabels, else return the major ticklabels

`which` [None, ('minor', 'major', 'both')] Overrides minor.

Selects which ticklabels to return

Returns

`ret` [list] List of Text instances.

`matplotlib.axis.XAxis.get_ticklines`

`XAxis.get_ticklines(minor=False)`

Return the tick lines as a list of `Line2D` instances

`matplotlib.axis.XAxis.get_ticklocs`

`XAxis.get_ticklocs(minor=False)`

Get the tick locations in data coordinates as a numpy array

`matplotlib.axis.XAxis.get_tightbbox`

`XAxis.get_tightbbox(renderer)`

Return a bounding box that encloses the axis. It only accounts tick labels, axis label, and `offsetText`.

`matplotlib.axis.XAxis.get_units`

`XAxis.get_units()`

return the units for axis

`matplotlib.axis.XAxis.get_view_interval`

`XAxis.get_view_interval()`

return the `Interval` instance for this axis view limits

`matplotlib.axis.XAxis.grid`

`XAxis.grid(b=None, which='major', **kwargs)`

Set the axis grid on or off; `b` is a boolean. Use `which = 'major' | 'minor' | 'both'` to set the grid for major or minor ticks.

If `b` is `None` and `len(kwargs)==0`, toggle the grid state. If `kwargs` are supplied, it is assumed you want the grid on and `b` will be set to `True`.

`kwargs` are used to set the line properties of the grids, e.g.,

`xax.grid(color='r', linestyle='-', linewidth=2)`

`matplotlib.axis.XAxis.iter_ticks`

`XAxis.iter_ticks()`

Iterate through all of the major and minor ticks.

`matplotlib.axis.XAxis.limit_range_for_scale`

`XAxis.limit_range_for_scale(vmin, vmax)`

`matplotlib.axis.XAxis.pan`

`XAxis.pan(numsteps)`

Pan numsteps (can be positive or negative)

`matplotlib.axis.XAxis.reset_ticks`

`XAxis.reset_ticks()`

Re-initialize the major and minor Tick lists.

Each list starts with a single fresh Tick.

`matplotlib.axis.XAxis.set_data_interval`

`XAxis.set_data_interval(vmin, vmax, ignore=False)`

set the axis data limits

`matplotlib.axis.XAxis.set_default_intervals`

`XAxis.set_default_intervals()`

set the default limits for the axis interval if they are not mutated

`matplotlib.axis.XAxis.set_label_coors`

`XAxis.set_label_coors(x, y, transform=None)`

Set the coordinates of the label. By default, the x coordinate of the y label is determined by the tick label bounding boxes, but this can lead to poor alignment of multiple ylabels if there are multiple axes. Ditto for the y coordinate of the x label.

You can also specify the coordinate system of the label with the transform. If None, the default coordinate system will be the axes coordinate system (0,0) is (left,bottom), (0.5, 0.5) is middle, etc

`matplotlib.axis.XAxis.set_label_position`

`XAxis.set_label_position(position)`

Set the label position (top or bottom)

ACCEPTS: ['top' | 'bottom']

`matplotlib.axis.XAxis.set_label_text`

`XAxis.set_label_text(label, fontdict=None, **kwargs)`

Sets the text value of the axis label

ACCEPTS: A string value for the label

`matplotlib.axis.XAxis.set_major_formatter`

`XAxis.set_major_formatter(formatter)`

Set the formatter of the major ticker

ACCEPTS: A Formatter instance

`matplotlib.axis.XAxis.set_major_locator`

`XAxis.set_major_locator(locator)`

Set the locator of the major ticker

ACCEPTS: a Locator instance

`matplotlib.axis.XAxis.set_minor_formatter`

`XAxis.set_minor_formatter(formatter)`

Set the formatter of the minor ticker

ACCEPTS: A Formatter instance

`matplotlib.axis.XAxis.set_minor_locator`

`XAxis.set_minor_locator(locator)`

Set the locator of the minor ticker

ACCEPTS: a Locator instance

`matplotlib.axis.XAxis.set_pickradius`

`XAxis.set_pickradius(pickradius)`

Set the depth of the axis used by the picker

ACCEPTS: a distance in points

`matplotlib.axis.XAxis.set_smart_bounds`

`XAxis.set_smart_bounds(value)`

set the axis to have smart bounds

`matplotlib.axis.XAxis.set_tick_params`

`XAxis.set_tick_params(which='major', reset=False, **kw)`

Set appearance parameters for ticks, ticklabels, and gridlines.

For documentation of keyword arguments, see `matplotlib.axes.Axes.tick_params()`.

`matplotlib.axis.XAxis.set_ticklabels`

`XAxis.set_ticklabels(ticklabels, *args, **kwargs)`

Set the text values of the tick labels. Return a list of Text instances. Use kwarg `minor=True` to select minor ticks. All other kwargs are used to update the text object properties. As for `get_ticklabels`, `label1` (left or bottom) is affected for a given tick only if its `label1On` attribute is True, and similarly for `label2`. The list of returned label text objects consists of all such `label1` objects followed by all such `label2` objects.

The input `ticklabels` is assumed to match the set of tick locations, regardless of the state of `label1On` and `label2On`.

ACCEPTS: sequence of strings or Text objects

`matplotlib.axis.XAxis.set_ticks`

`XAxis.set_ticks(ticks, minor=False)`

Set the locations of the tick marks from sequence `ticks`

ACCEPTS: sequence of floats

`matplotlib.axis.XAxis.set_units`

`XAxis.set_units(u)`

set the units for axis

ACCEPTS: a units tag

`matplotlib.axis.XAxis.set_view_interval`

`XAxis.set_view_interval(vmin, vmax, ignore=False)`

If `ignore` is False, the order of `vmin`, `vmax` does not matter; the original axis orientation will be preserved. In addition, the view limits can be expanded, but will not be reduced. This method is for mpl internal use; for normal use, see `set_xlim()`.

`matplotlib.axis.XAxis.update_units`

`XAxis.update_units(data)`

introspect data for units converter and update the `axis.converter` instance if necessary. Return True if data is registered for unit conversion.

matplotlib.axis.XAxis.zoom

XAxis.zoom(direction)

Zoom in/out on axis; if direction is >0 zoom in, else zoom out

36.4.5 Inherited from artist

Ticks

Tick.add_callback	Adds a callback function that will be called whenever one of the Artist's properties changes.
Tick.aname	
Tick.axes	The Axes instance the artist resides in, or None.
Tick.contains	Test whether the mouse event occurred in the Tick marks.
Tick.convert_xunits	For artists in an axes, if the xaxis has units support, convert x using xaxis unit type
Tick.convert_yunits	For artists in an axes, if the yaxis has units support, convert y using yaxis unit type
Tick.draw	Derived classes drawing method
Tick.findobj	Find artist objects.
Tick.format_cursor_data	Return cursor data string formatted.
Tick.get_agg_filter	Return filter function to be used for agg filter.
Tick.get_alpha	Return the alpha value used for blending - not supported on all backends
Tick.get_animated	Return the artist's animated state
Tick.get_children	Return a list of the child Artist's this :class:`Artist` contains.
Tick.get_clip_box	Return artist clipbox
Tick.get_clip_on	Return whether artist uses clipping
Tick.get_clip_path	Return artist clip path
Tick.get_contains	Return the _contains test used by the artist, or None for default.
Tick.get_cursor_data	Get the cursor data for a given event.
Tick.get_figure	Return the Figure instance the artist belongs to.
Tick.get_gid	Returns the group id.
Tick.get_label	Get the label used for this artist in the legend.
Tick.get_path_effects	
Tick.get_picker	Return the picker object used by this artist.
Tick.get_rasterized	Return whether the artist is to be rasterized.
Tick.get_sketch_params	Returns the sketch parameters for the artist.

Continued on next page

Table 19 – continued from previous page

Tick.get_snap	Returns the snap setting which may be:
Tick.get_transform	Return the Transform instance used by this artist.
Tick.get_transformed_clip_path_and_affine	Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.
Tick.get_url	Returns the url.
Tick.get_visible	Return the artist's visibility
Tick.get_window_extent	Get the axes bounding box in display space.
Tick.get_zorder	Return the artist's zorder.
Tick.have_units	Return True if units are set on the x or y axes
Tick.is_transform_set	Returns True if Artist has a transform explicitly set.
Tick.mouseover	
Tick.pchanged	Fire an event when property changed, calling all of the registered callbacks.
Tick.pick	Process pick event
Tick.pickable	Return True if Artist is pickable.
Tick.properties	return a dictionary mapping property name -> value for all Artist props
Tick.remove	Remove the artist from the figure if possible.
Tick.remove_callback	Remove a callback based on its id.
Tick.set	A property batch setter.
Tick.set_agg_filter	Set the agg filter.
Tick.set_alpha	Set the alpha value used for blending - not supported on all backends.
Tick.set_animated	Set the artist's animation state.
Tick.set_clip_box	Set the artist's clip Bbox.
Tick.set_clip_on	Set whether artist uses clipping.
Tick.set_clip_path	Set the artist's clip path, which may be:
Tick.set_contains	Replace the contains test used by this artist.
Tick.set_figure	Set the Figure instance the artist belongs to.
Tick.set_gid	Sets the (group) id for the artist.
Tick.set_label	Set the text of ticklabel
Tick.set_path_effects	Set the path effects.
Tick.set_picker	Set the epsilon for picking used by this artist
Tick.set_rasterized	Force rasterized (bitmap) drawing in vector backend output.
Tick.set_sketch_params	Sets the sketch parameters.
Tick.set_snap	Sets the snap setting which may be:
Tick.set_transform	Set the artist transform.
Tick.set_url	Sets the url for the artist.
Tick.set_visible	Set the artist's visibility.
Tick.set_zorder	Set the zorder for the artist.

Continued on next page

Table 19 – continued from previous page

Tick.stale	If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.
Tick.update	Update this artist's properties from the dictionary prop.
Tick.update_from	Copy properties from other to self.
Tick.zorder	
XTick.add_callback	Adds a callback function that will be called whenever one of the Artist's properties changes.
XTick.aname	
XTick.axes	The Axes instance the artist resides in, or None.
XTick.contains	Test whether the mouse event occurred in the Tick marks.
XTick.convert_xunits	For artists in an axes, if the xaxis has units support, convert x using xaxis unit type
XTick.convert_yunits	For artists in an axes, if the yaxis has units support, convert y using yaxis unit type
XTick.draw	Derived classes drawing method
XTick.findobj	Find artist objects.
XTick.format_cursor_data	Return cursor data string formatted.
XTick.get_agg_filter	Return filter function to be used for agg filter.
XTick.get_alpha	Return the alpha value used for blending - not supported on all backends
XTick.get_animated	Return the artist's animated state
XTick.get_children	Return a list of the child Artist's this :class:`Artist` contains.
XTick.get_clip_box	Return artist clipbox
XTick.get_clip_on	Return whether artist uses clipping
XTick.get_clip_path	Return artist clip path
XTick.get_contains	Return the _contains test used by the artist, or None for default.
XTick.get_cursor_data	Get the cursor data for a given event.
XTick.get_figure	Return the Figure instance the artist belongs to.
XTick.get_gid	Returns the group id.
XTick.get_label	Get the label used for this artist in the legend.
XTick.get_path_effects	
XTick.get_picker	Return the picker object used by this artist.
XTick.get_rasterized	Return whether the artist is to be rasterized.
XTick.get_sketch_params	Returns the sketch parameters for the artist.
XTick.get_snap	Returns the snap setting which may be:
XTick.get_transform	Return the Transform instance used by this artist.

Continued on next page

Table 19 – continued from previous page

<code>XTick.get_transformed_clip_path_and_affine</code>	Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.
<code>XTick.get_url</code>	Returns the url.
<code>XTick.get_visible</code>	Return the artist's visibility
<code>XTick.get_window_extent</code>	Get the axes bounding box in display space.
<code>XTick.get_zorder</code>	Return the artist's zorder.
<code>XTick.have_units</code>	Return True if units are set on the x or y axes
<code>XTick.is_transform_set</code>	Returns True if Artist has a transform explicitly set.
<code>XTick.mouseover</code>	
<code>XTick.pchanged</code>	Fire an event when property changed, calling all of the registered callbacks.
<code>XTick.pick</code>	Process pick event
<code>XTick.pickable</code>	Return True if Artist is pickable.
<code>XTick.properties</code>	return a dictionary mapping property name -> value for all Artist props
<code>XTick.remove</code>	Remove the artist from the figure if possible.
<code>XTick.remove_callback</code>	Remove a callback based on its id.
<code>XTick.set</code>	A property batch setter.
<code>XTick.set_agg_filter</code>	Set the agg filter.
<code>XTick.set_alpha</code>	Set the alpha value used for blending - not supported on all backends.
<code>XTick.set_animated</code>	Set the artist's animation state.
<code>XTick.set_clip_box</code>	Set the artist's clip Bbox.
<code>XTick.set_clip_on</code>	Set whether artist uses clipping.
<code>XTick.set_clip_path</code>	Set the artist's clip path, which may be:
<code>XTick.set_contains</code>	Replace the contains test used by this artist.
<code>XTick.set_figure</code>	Set the Figure instance the artist belongs to.
<code>XTick.set_gid</code>	Sets the (group) id for the artist.
<code>XTick.set_label</code>	Set the text of ticklabel
<code>XTick.set_path_effects</code>	Set the path effects.
<code>XTick.set_picker</code>	Set the epsilon for picking used by this artist
<code>XTick.set_rasterized</code>	Force rasterized (bitmap) drawing in vector backend output.
<code>XTick.set_sketch_params</code>	Sets the sketch parameters.
<code>XTick.set_snap</code>	Sets the snap setting which may be:
<code>XTick.set_transform</code>	Set the artist transform.
<code>XTick.set_url</code>	Sets the url for the artist.
<code>XTick.set_visible</code>	Set the artist's visibility.
<code>XTick.set_zorder</code>	Set the zorder for the artist.
<code>XTick.stale</code>	If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

Continued on next page

Table 19 – continued from previous page

XTick.update	Update this artist's properties from the dictionary prop.
XTick.update_from	Copy properties from other to self.
XTick.zorder	
YTick.add_callback	Adds a callback function that will be called whenever one of the Artist's properties changes.
YTick.aname	
YTick.axes	The Axes instance the artist resides in, or None.
YTick.contains	Test whether the mouse event occurred in the Tick marks.
YTick.convert_xunits	For artists in an axes, if the xaxis has units support, convert x using xaxis unit type
YTick.convert_yunits	For artists in an axes, if the yaxis has units support, convert y using yaxis unit type
YTick.draw	Derived classes drawing method
YTick.findobj	Find artist objects.
YTick.format_cursor_data	Return cursor data string formatted.
YTick.get_agg_filter	Return filter function to be used for agg filter.
YTick.get_alpha	Return the alpha value used for blending - not supported on all backends
YTick.get_animated	Return the artist's animated state
YTick.get_children	Return a list of the child Artist's this :class:`Artist` contains.
YTick.get_clip_box	Return artist clipbox
YTick.get_clip_on	Return whether artist uses clipping
YTick.get_clip_path	Return artist clip path
YTick.get_contains	Return the _contains test used by the artist, or None for default.
YTick.get_cursor_data	Get the cursor data for a given event.
YTick.get_figure	Return the Figure instance the artist belongs to.
YTick.get_gid	Returns the group id.
YTick.get_label	Get the label used for this artist in the legend.
YTick.get_path_effects	
YTick.get_picker	Return the picker object used by this artist.
YTick.get_rasterized	Return whether the artist is to be rasterized.
YTick.get_sketch_params	Returns the sketch parameters for the artist.
YTick.get_snap	Returns the snap setting which may be:
YTick.get_transform	Return the Transform instance used by this artist.
YTick.get_transformed_clip_path_and_affine	Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

Continued on next page

Table 19 – continued from previous page

YTick.get_url	Returns the url.
YTick.get_visible	Return the artist's visibility
YTick.get_window_extent	Get the axes bounding box in display space.
YTick.get_zorder	Return the artist's zorder.
YTick.have_units	Return True if units are set on the x or y axes
YTick.is_transform_set	Returns True if Artist has a transform explicitly set.
YTick.mouseover	
YTick.pchanged	Fire an event when property changed, calling all of the registered callbacks.
YTick.pick	Process pick event
YTick.pickable	Return True if Artist is pickable.
YTick.properties	return a dictionary mapping property name -> value for all Artist props
YTick.remove	Remove the artist from the figure if possible.
YTick.remove_callback	Remove a callback based on its id.
YTick.set	A property batch setter.
YTick.set_agg_filter	Set the agg filter.
YTick.set_alpha	Set the alpha value used for blending - not supported on all backends.
YTick.set_animated	Set the artist's animation state.
YTick.set_clip_box	Set the artist's clip Bbox.
YTick.set_clip_on	Set whether artist uses clipping.
YTick.set_clip_path	Set the artist's clip path, which may be:
YTick.set_contains	Replace the contains test used by this artist.
YTick.set_figure	Set the Figure instance the artist belongs to.
YTick.set_gid	Sets the (group) id for the artist.
YTick.set_label	Set the text of ticklabel
YTick.set_path_effects	Set the path effects.
YTick.set_picker	Set the epsilon for picking used by this artist
YTick.set_rasterized	Force rasterized (bitmap) drawing in vector backend output.
YTick.set_sketch_params	Sets the sketch parameters.
YTick.set_snap	Sets the snap setting which may be:
YTick.set_transform	Set the artist transform.
YTick.set_url	Sets the url for the artist.
YTick.set_visible	Set the artist's visibility.
YTick.set_zorder	Set the zorder for the artist.
YTick.stale	If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.
YTick.update	Update this artist's properties from the dictionary prop.
YTick.update_from	Copy properties from other to self.

Continued on next page

Table 19 – continued from previous page

`YTick.zorder`

`matplotlib.axis.Tick.add_callback``Tick.add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`matplotlib.axis.Tick.aname``Tick.aname = 'Artist'``matplotlib.axis.Tick.axes``Tick.axes`

The Axes instance the artist resides in, or None.

`matplotlib.axis.Tick.contains``Tick.contains(mouseevent)`

Test whether the mouse event occurred in the Tick marks.

This function always returns false. It is more useful to test if the axis as a whole contains the mouse rather than the set of tick marks.

`matplotlib.axis.Tick.convert_xunits``Tick.convert_xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`matplotlib.axis.Tick.convert_yunits``Tick.convert_yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`matplotlib.axis.Tick.draw``Tick.draw(renderer)`

Derived classes drawing method

`matplotlib.axis.Tick.findobj`

`Tick.findobj(match=None, include_self=True)`

Find artist objects.

Recursively find all Artist instances contained in self.

match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is True (default), include self in the list to be checked for a match.

`matplotlib.axis.Tick.format_cursor_data`

`Tick.format_cursor_data(data)`

Return cursor data string formatted.

`matplotlib.axis.Tick.get_agg_filter`

`Tick.get_agg_filter()`

Return filter function to be used for agg filter.

`matplotlib.axis.Tick.get_alpha`

`Tick.get_alpha()`

Return the alpha value used for blending - not supported on all backends

`matplotlib.axis.Tick.get_animated`

`Tick.get_animated()`

Return the artist's animated state

`matplotlib.axis.Tick.get_children`

`Tick.get_children()`

Return a list of the child Artist's this :class:`Artist` contains.

`matplotlib.axis.Tick.get_clip_box`

`Tick.get_clip_box()`
Return artist clipbox

`matplotlib.axis.Tick.get_clip_on`

`Tick.get_clip_on()`
Return whether artist uses clipping

`matplotlib.axis.Tick.get_clip_path`

`Tick.get_clip_path()`
Return artist clip path

`matplotlib.axis.Tick.get_contains`

`Tick.get_contains()`
Return the `_contains` test used by the artist, or `None` for default.

`matplotlib.axis.Tick.get_cursor_data`

`Tick.get_cursor_data(event)`
Get the cursor data for a given event.

`matplotlib.axis.Tick.get_figure`

`Tick.get_figure()`
Return the `Figure` instance the artist belongs to.

`matplotlib.axis.Tick.get_gid`

`Tick.get_gid()`
Returns the group id.

`matplotlib.axis.Tick.get_label`

`Tick.get_label()`
Get the label used for this artist in the legend.

`matplotlib.axis.Tick.get__path_effects`

`Tick.get__path_effects()`

`matplotlib.axis.Tick.get__picker`

`Tick.get__picker()`

Return the picker object used by this artist.

`matplotlib.axis.Tick.get__rasterized`

`Tick.get__rasterized()`

Return whether the artist is to be rasterized.

`matplotlib.axis.Tick.get__sketch_params`

`Tick.get__sketch_params()`

Returns the sketch parameters for the artist.

Returns

`sketch_params` [tuple or `None`]

A 3-tuple with the following elements:

- `scale`: The amplitude of the wiggle perpendicular to the source line.
- `length`: The length of the wiggle along the line.
- `randomness`: The scale factor by which the length is shrunk or expanded.

May return ‘None’ if no sketch parameters were set.

`matplotlib.axis.Tick.get__snap`

`Tick.get__snap()`

Returns the snap setting which may be:

- `True`: snap vertices to the nearest pixel center
- `False`: leave vertices as-is
- `None`: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`matplotlib.axis.Tick.get__transform`

`Tick.get__transform()`

Return the Transform instance used by this artist.

`matplotlib.axis.Tick.get__transformed_clip_path_and__affine`

`Tick.get__transformed_clip_path_and__affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`matplotlib.axis.Tick.get__url`

`Tick.get__url()`

Returns the url.

`matplotlib.axis.Tick.get__visible`

`Tick.get__visible()`

Return the artist's visibility

`matplotlib.axis.Tick.get__window_extent`

`Tick.get__window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`matplotlib.axis.Tick.get__zorder`

`Tick.get__zorder()`

Return the artist's zorder.

`matplotlib.axis.Tick.have__units`

`Tick.have__units()`

Return True if units are set on the x or y axes

`matplotlib.axis.Tick.is_transform_set`

`Tick.is_transform_set()`

Returns True if Artist has a transform explicitly set.

`matplotlib.axis.Tick.mouseover`

`Tick.mouseover`

`matplotlib.axis.Tick.pchanged`

`Tick.pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`matplotlib.axis.Tick.pick`

`Tick.pick(mouseevent)`

Process pick event

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

`matplotlib.axis.Tick.pickable`

`Tick.pickable()`

Return True if Artist is pickable.

`matplotlib.axis.Tick.properties`

`Tick.properties()`

return a dictionary mapping property name -> value for all Artist props

`matplotlib.axis.Tick.remove`

`Tick.remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`matplotlib.axis.Tick.remove_callback`

`Tick.remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`matplotlib.axis.Tick.set`

`Tick.set(**kwargs)`

A property batch setter. Pass kwargs to set properties.

`matplotlib.axis.Tick.set_agg_filter`

`Tick.set_agg_filter(filter_func)`

Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`matplotlib.axis.Tick.set_alpha`

`Tick.set_alpha(alpha)`

Set the alpha value used for blending - not supported on all backends.

Parameters

`alpha` [float]

`matplotlib.axis.Tick.set_animated`

`Tick.set_animated(b)`

Set the artist's animation state.

Parameters

`b` [bool]

matplotlib.axis.Tick.set_clip_box

Tick.set_clip_box(clipbox)

Set the artist's clip Bbox.

Parameters

clipbox [Bbox]

matplotlib.axis.Tick.set_clip_on

Tick.set_clip_on(b)

Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

b [bool]

matplotlib.axis.Tick.set_clip_path

Tick.set_clip_path(clippath, transform=None)

Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

matplotlib.axis.Tick.set_contains

Tick.set_contains(picker)

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return hit = True and props is a dictionary of properties you want returned with the contains test.

Parameters

picker [callable]

`matplotlib.axis.Tick.set_figure`

`Tick.set_figure(fig)`

Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`matplotlib.axis.Tick.set_gid`

`Tick.set_gid(gid)`

Sets the (group) id for the artist.

Parameters

`gid` [str]

`matplotlib.axis.Tick.set_label`

`Tick.set_label(s)`

Set the text of ticklabel

ACCEPTS: str

`matplotlib.axis.Tick.set_path_effects`

`Tick.set_path_effects(path_effects)`

Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`matplotlib.axis.Tick.set_picker`

`Tick.set_picker(picker)`

Set the epsilon for picking used by this artist

`picker` can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if `picker` is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some

artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event

- A function: if picker is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

to determine the hit test. if the mouse event is over the artist, return hit=True and props is a dictionary of properties you want added to the PickEvent attributes.

Parameters

picker [None or bool or float or callable]

matplotlib.axis.Tick.set__rasterized

Tick.set__rasterized(rasterized)

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

rasterized [bool or None]

matplotlib.axis.Tick.set__sketch_params

Tick.set__sketch_params(scale=None, length=None, randomness=None)

Sets the sketch parameters.

Parameters

scale [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If scale is [None](#), or not provided, no sketch filter will be provided.

length [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

randomness [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

matplotlib.axis.Tick.set__snap

Tick.set__snap(snap)

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is

- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

 snap [bool or None]

`matplotlib.axis.Tick.set__transform`

`Tick.set__transform(t)`

Set the artist transform.

Parameters

 t [Transform]

`matplotlib.axis.Tick.set__url`

`Tick.set__url(url)`

Sets the url for the artist.

Parameters

 url [str]

`matplotlib.axis.Tick.set__visible`

`Tick.set__visible(b)`

Set the artist's visibility.

Parameters

 b [bool]

`matplotlib.axis.Tick.set__zorder`

`Tick.set__zorder(level)`

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

 level [float]

`matplotlib.axis.Tick.stale`

`Tick.stale`

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

`matplotlib.axis.Tick.update`

`Tick.update(props)`

Update this artist's properties from the dictionary prop.

`matplotlib.axis.Tick.update_from`

`Tick.update_from(other)`

Copy properties from other to self.

`matplotlib.axis.Tick.zorder`

`Tick.zorder = 0`

`matplotlib.axis.XTick.add_callback`

`XTick.add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`matplotlib.axis.XTick.aname`

`XTick.aname = 'Artist'`

`matplotlib.axis.XTick.axes`

`XTick.axes`

The Axes instance the artist resides in, or None.

`matplotlib.axis.XTick.contains`

`XTick.contains(mouseevent)`

Test whether the mouse event occurred in the Tick marks.

This function always returns false. It is more useful to test if the axis as a whole contains the mouse rather than the set of tick marks.

`matplotlib.axis.XTick.convert_xunits`

`XTick.convert_xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`matplotlib.axis.XTick.convert_yunits`

`XTick.convert_yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`matplotlib.axis.XTick.draw`

`XTick.draw(renderer)`

Derived classes drawing method

`matplotlib.axis.XTick.findobj`

`XTick.findobj(match=None, include_self=True)`

Find artist objects.

Recursively find all Artist instances contained in self.

match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is `True` (default), include self in the list to be checked for a match.

`matplotlib.axis.XTick.format_cursor_data`

`XTick.format_cursor_data(data)`

Return cursor data string formatted.

`matplotlib.axis.XTick.get_agg_filter`

`XTick.get_agg_filter()`
Return filter function to be used for agg filter.

`matplotlib.axis.XTick.get_alpha`

`XTick.get_alpha()`
Return the alpha value used for blending - not supported on all backends

`matplotlib.axis.XTick.get_animated`

`XTick.get_animated()`
Return the artist's animated state

`matplotlib.axis.XTick.get_children`

`XTick.get_children()`
Return a list of the child Artist's this :class:`Artist` contains.

`matplotlib.axis.XTick.get_clip_box`

`XTick.get_clip_box()`
Return artist clipbox

`matplotlib.axis.XTick.get_clip_on`

`XTick.get_clip_on()`
Return whether artist uses clipping

`matplotlib.axis.XTick.get_clip_path`

`XTick.get_clip_path()`
Return artist clip path

`matplotlib.axis.XTick.get_contains`

`XTick.get_contains()`
Return the `_contains` test used by the artist, or `None` for default.

`matplotlib.axis.XTick.get_cursor_data`

`XTick.get_cursor_data(event)`
Get the cursor data for a given event.

`matplotlib.axis.XTick.get_figure`

`XTick.get_figure()`
Return the Figure instance the artist belongs to.

`matplotlib.axis.XTick.get_gid`

`XTick.get_gid()`
Returns the group id.

`matplotlib.axis.XTick.get_label`

`XTick.get_label()`
Get the label used for this artist in the legend.

`matplotlib.axis.XTick.get_path_effects`

`XTick.get_path_effects()`

`matplotlib.axis.XTick.get_picker`

`XTick.get_picker()`
Return the picker object used by this artist.

`matplotlib.axis.XTick.get_rasterized`

`XTick.get_rasterized()`
Return whether the artist is to be rasterized.

`matplotlib.axis.XTick.get_sketch_params`

`XTick.get_sketch_params()`
Returns the sketch parameters for the artist.

Returns

sketch_params [tuple or `None`]

A 3-tuple with the following elements:

- scale: The amplitude of the wiggle perpendicular to the source line.
- length: The length of the wiggle along the line.
- randomness: The scale factor by which the length is shrunk or expanded.

May return `'None'` if no sketch parameters were set.

`matplotlib.axis.XTick.get_snap`

`XTick.get_snap()`

Returns the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`matplotlib.axis.XTick.get_transform`

`XTick.get_transform()`

Return the Transform instance used by this artist.

`matplotlib.axis.XTick.get_transformed_clip_path_and_affine`

`XTick.get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`matplotlib.axis.XTick.get_url`

`XTick.get_url()`

Returns the url.

`matplotlib.axis.XTick.get_visible`

`XTick.get_visible()`

Return the artist's visibility

`matplotlib.axis.XTick.get_window_extent`

`XTick.get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`matplotlib.axis.XTick.get_zorder`

`XTick.get_zorder()`

Return the artist's zorder.

`matplotlib.axis.XTick.have_units`

`XTick.have_units()`

Return True if units are set on the x or y axes

`matplotlib.axis.XTick.is_transform_set`

`XTick.is_transform_set()`

Returns True if Artist has a transform explicitly set.

`matplotlib.axis.XTick.mouseover`

`XTick.mouseover`

`matplotlib.axis.XTick.pchanged`

`XTick.pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`matplotlib.axis.XTick.pick`

`XTick.pick(mouseevent)`

Process pick event

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

`matplotlib.axis.XTick.pickable`

`XTick.pickable()`

Return True if Artist is pickable.

`matplotlib.axis.XTick.properties`

`XTick.properties()`

return a dictionary mapping property name -> value for all Artist props

`matplotlib.axis.XTick.remove`

`XTick.remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`matplotlib.axis.XTick.remove_callback`

`XTick.remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`matplotlib.axis.XTick.set`

`XTick.set(**kwargs)`

A property batch setter. Pass kwargs to set properties.

`matplotlib.axis.XTick.set_agg_filter`

`XTick.set_agg_filter(filter_func)`

Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`matplotlib.axis.XTick.set_alpha`

`XTick.set_alpha(alpha)`

Set the alpha value used for blending - not supported on all backends.

Parameters

`alpha` [float]

`matplotlib.axis.XTick.set_animated`

`XTick.set_animated(b)`

Set the artist's animation state.

Parameters

`b` [bool]

`matplotlib.axis.XTick.set_clip_box`

`XTick.set_clip_box(clipbox)`

Set the artist's clip Bbox.

Parameters

`clipbox` [Bbox]

`matplotlib.axis.XTick.set_clip_on`

`XTick.set_clip_on(b)`

Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

`b` [bool]

`matplotlib.axis.XTick.set_clip_path`

`XTick.set_clip_path(clippath, transform=None)`

Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

`matplotlib.axis.XTick.set_contains`

`XTick.set_contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

`picker` [callable]

`matplotlib.axis.XTick.set_figure`

`XTick.set_figure(fig)`

Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`matplotlib.axis.XTick.set_gid`

`XTick.set_gid(gid)`

Sets the (group) id for the artist.

Parameters

`gid` [str]

`matplotlib.axis.XTick.set_label`

`XTick.set_label(s)`

Set the text of ticklabel

ACCEPTS: str

`matplotlib.axis.XTick.set_path_effects`

`XTick.set_path_effects(path_effects)`

Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`matplotlib.axis.XTick.set_picker`

`XTick.set_picker(picker)`

Set the epsilon for picking used by this artist

`picker` can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if `picker` is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if `picker` is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

`hit, props = picker(artist, mouseevent)`

to determine the hit test. if the mouse event is over the artist, return `hit=True` and `props` is a dictionary of properties you want added to the `PickEvent` attributes.

Parameters

`picker` [None or bool or float or callable]

`matplotlib.axis.XTick.set_rasterized`

`XTick.set_rasterized(rasterized)`

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

`rasterized` [bool or None]

`matplotlib.axis.XTick.set_sketch_params`

`XTick.set_sketch_params(scale=None, length=None, randomness=None)`

Sets the sketch parameters.

Parameters

`scale` [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If `scale` is `None`, or not provided, no sketch filter will be provided.

`length` [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

`randomness` [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

`matplotlib.axis.XTick.set_snap`

`XTick.set_snap(snap)`

Sets the snap setting which may be:

- `True`: snap vertices to the nearest pixel center
- `False`: leave vertices as-is
- `None`: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

`snap` [bool or None]

`matplotlib.axis.XTick.set_transform`

`XTick.set_transform(t)`

Set the artist transform.

Parameters

`t` [Transform]

`matplotlib.axis.XTick.set_url`

`XTick.set_url(url)`

Sets the url for the artist.

Parameters

url [str]

matplotlib.axis.XTick.set_visible

XTick.set_visible(b)

Set the artist's visibility.

Parameters

b [bool]

matplotlib.axis.XTick.set_zorder

XTick.set_zorder(level)

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

level [float]

matplotlib.axis.XTick.stale

XTick.stale

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

matplotlib.axis.XTick.update

XTick.update(props)

Update this artist's properties from the dictionary prop.

matplotlib.axis.XTick.update_from

XTick.update_from(other)

Copy properties from other to self.

matplotlib.axis.XTick.zorder

XTick.zorder = 0

`matplotlib.axis.YTick.add_callback`

`YTick.add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`matplotlib.axis.YTick.aname`

`YTick.aname = 'Artist'`

`matplotlib.axis.YTick.axes`

`YTick.axes`

The Axes instance the artist resides in, or None.

`matplotlib.axis.YTick.contains`

`YTick.contains(mouseevent)`

Test whether the mouse event occurred in the Tick marks.

This function always returns false. It is more useful to test if the axis as a whole contains the mouse rather than the set of tick marks.

`matplotlib.axis.YTick.convert_xunits`

`YTick.convert_xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`matplotlib.axis.YTick.convert_yunits`

`YTick.convert_yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`matplotlib.axis.YTick.draw`

`YTick.draw(renderer)`

Derived classes drawing method

`matplotlib.axis.YTick.findobj`

`YTick.findobj(match=None, include_self=True)`

Find artist objects.

Recursively find all Artist instances contained in self.

match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is `True` (default), include self in the list to be checked for a match.

`matplotlib.axis.YTick.format_cursor_data`

`YTick.format_cursor_data(data)`

Return cursor data string formatted.

`matplotlib.axis.YTick.get_agg_filter`

`YTick.get_agg_filter()`

Return filter function to be used for agg filter.

`matplotlib.axis.YTick.get_alpha`

`YTick.get_alpha()`

Return the alpha value used for blending - not supported on all backends

`matplotlib.axis.YTick.get_animated`

`YTick.get_animated()`

Return the artist's animated state

`matplotlib.axis.YTick.get_children`

`YTick.get_children()`

Return a list of the child Artist's this :class:`Artist` contains.

`matplotlib.axis.YTick.get_clip_box`

`YTick.get_clip_box()`
Return artist clipbox

`matplotlib.axis.YTick.get_clip_on`

`YTick.get_clip_on()`
Return whether artist uses clipping

`matplotlib.axis.YTick.get_clip_path`

`YTick.get_clip_path()`
Return artist clip path

`matplotlib.axis.YTick.get_contains`

`YTick.get_contains()`
Return the `_contains` test used by the artist, or `None` for default.

`matplotlib.axis.YTick.get_cursor_data`

`YTick.get_cursor_data(event)`
Get the cursor data for a given event.

`matplotlib.axis.YTick.get_figure`

`YTick.get_figure()`
Return the `Figure` instance the artist belongs to.

`matplotlib.axis.YTick.get_gid`

`YTick.get_gid()`
Returns the group id.

`matplotlib.axis.YTick.get_label`

`YTick.get_label()`
Get the label used for this artist in the legend.

`matplotlib.axis.YTick.get_path_effects`

`YTick.get_path_effects()`

`matplotlib.axis.YTick.get_picker`

`YTick.get_picker()`

Return the picker object used by this artist.

`matplotlib.axis.YTick.get_rasterized`

`YTick.get_rasterized()`

Return whether the artist is to be rasterized.

`matplotlib.axis.YTick.get_sketch_params`

`YTick.get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

`sketch_params` [tuple or `None`]

A 3-tuple with the following elements:

- `scale`: The amplitude of the wiggle perpendicular to the source line.
- `length`: The length of the wiggle along the line.
- `randomness`: The scale factor by which the length is shrunk or expanded.

May return `'None'` if no sketch parameters were set.

`matplotlib.axis.YTick.get_snap`

`YTick.get_snap()`

Returns the snap setting which may be:

- `True`: snap vertices to the nearest pixel center
- `False`: leave vertices as-is
- `None`: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`matplotlib.axis.YTick.get_transform`

`YTick.get_transform()`

Return the Transform instance used by this artist.

`matplotlib.axis.YTick.get_transformed_clip_path_and_affine`

`YTick.get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`matplotlib.axis.YTick.get_url`

`YTick.get_url()`

Returns the url.

`matplotlib.axis.YTick.get_visible`

`YTick.get_visible()`

Return the artist's visibility

`matplotlib.axis.YTick.get_window_extent`

`YTick.get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`matplotlib.axis.YTick.get_zorder`

`YTick.get_zorder()`

Return the artist's zorder.

`matplotlib.axis.YTick.have_units`

`YTick.have_units()`

Return True if units are set on the x or y axes

`matplotlib.axis.YTick.is_transform_set`

`YTick.is_transform_set()`

Returns True if Artist has a transform explicitly set.

`matplotlib.axis.YTick.mouseover`

`YTick.mouseover`

`matplotlib.axis.YTick.pchanged`

`YTick.pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`matplotlib.axis.YTick.pick`

`YTick.pick(mouseevent)`

Process pick event

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

`matplotlib.axis.YTick.pickable`

`YTick.pickable()`

Return True if Artist is pickable.

`matplotlib.axis.YTick.properties`

`YTick.properties()`

return a dictionary mapping property name -> value for all Artist props

`matplotlib.axis.YTick.remove`

`YTick.remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`matplotlib.axis.YTick.remove_callback`

`YTick.remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`matplotlib.axis.YTick.set`

`YTick.set(**kwargs)`

A property batch setter. Pass kwargs to set properties.

`matplotlib.axis.YTick.set_agg_filter`

`YTick.set_agg_filter(filter_func)`

Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`matplotlib.axis.YTick.set_alpha`

`YTick.set_alpha(alpha)`

Set the alpha value used for blending - not supported on all backends.

Parameters

`alpha` [float]

`matplotlib.axis.YTick.set_animated`

`YTick.set_animated(b)`

Set the artist's animation state.

Parameters

`b` [bool]

`matplotlib.axis.YTick.set_clip_box`

`YTick.set_clip_box(clipbox)`

Set the artist's clip Bbox.

Parameters

`clipbox` [Bbox]

`matplotlib.axis.YTick.set_clip_on`

`YTick.set_clip_on(b)`

Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

`b` [bool]

`matplotlib.axis.YTick.set_clip_path`

`YTick.set_clip_path(clippath, transform=None)`

Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

`matplotlib.axis.YTick.set_contains`

`YTick.set_contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

`picker` [callable]

`matplotlib.axis.YTick.set_figure`

`YTick.set_figure(fig)`

Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`matplotlib.axis.YTick.set_gid`

`YTick.set_gid(gid)`

Sets the (group) id for the artist.

Parameters

`gid` [str]

`matplotlib.axis.YTick.set_label`

`YTick.set_label(s)`

Set the text of ticklabel

ACCEPTS: str

`matplotlib.axis.YTick.set_path_effects`

`YTick.set_path_effects(path_effects)`

Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`matplotlib.axis.YTick.set_picker`

`YTick.set_picker(picker)`

Set the epsilon for picking used by this artist

`picker` can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if `picker` is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some

artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event

- A function: if picker is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

to determine the hit test. if the mouse event is over the artist, return `hit=True` and `props` is a dictionary of properties you want added to the `PickEvent` attributes.

Parameters

picker [None or bool or float or callable]

matplotlib.axis.YTick.set__rasterized

YTick.set__rasterized(rasterized)

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

rasterized [bool or None]

matplotlib.axis.YTick.set__sketch__params

YTick.set__sketch__params(scale=None, length=None, randomness=None)

Sets the sketch parameters.

Parameters

scale [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If scale is [None](#), or not provided, no sketch filter will be provided.

length [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

randomness [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

matplotlib.axis.YTick.set__snap

YTick.set__snap(snap)

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is

- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

 snap [bool or None]

`matplotlib.axis.YTick.set__transform`

`YTick.set__transform(t)`

Set the artist transform.

Parameters

 t [Transform]

`matplotlib.axis.YTick.set__url`

`YTick.set__url(url)`

Sets the url for the artist.

Parameters

 url [str]

`matplotlib.axis.YTick.set__visible`

`YTick.set__visible(b)`

Set the artist's visibility.

Parameters

 b [bool]

`matplotlib.axis.YTick.set__zorder`

`YTick.set__zorder(level)`

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

 level [float]

matplotlib.axis.YTick.stale

YTick.stale
If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

matplotlib.axis.YTick.update

YTick.update(props)
Update this artist's properties from the dictionary prop.

matplotlib.axis.YTick.update_from

YTick.update_from(other)
Copy properties from other to self.

matplotlib.axis.YTick.zorder

YTick.zorder = 0

Axis

Axis.add_callback	Adds a callback function that will be called whenever one of the Artist's properties changes.
Axis.aname	
Axis.axes	The Axes instance the artist resides in, or None.
Axis.contains	Test whether the artist contains the mouse event.
Axis.convert_xunits	For artists in an axes, if the xaxis has units support, convert x using xaxis unit type
Axis.convert_yunits	For artists in an axes, if the yaxis has units support, convert y using yaxis unit type
Axis.draw	Draw the axis lines, grid lines, tick lines and labels
Axis.findobjj	Find artist objects.
Axis.format_cursor_data	Return cursor data string formatted.
Axis.get_agg_filter	Return filter function to be used for agg filter.
Axis.get_alpha	Return the alpha value used for blending - not supported on all backends

Continued on next page

Table 20 – continued from previous page

Axis.get_animated	Return the artist's animated state
Axis.get_children	Return a list of the child Artist's this :class:`Artist` contains.
Axis.get_clip_box	Return artist clipbox
Axis.get_clip_on	Return whether artist uses clipping
Axis.get_clip_path	Return artist clip path
Axis.get_contains	Return the <code>_contains</code> test used by the artist, or <code>None</code> for default.
Axis.get_cursor_data	Get the cursor data for a given event.
Axis.get_figure	Return the Figure instance the artist belongs to.
Axis.get_gid	Returns the group id.
Axis.get_label	Return the axis label as a Text instance
Axis.get_path_effects	
Axis.get_picker	Return the picker object used by this artist.
Axis.get_rasterized	Return whether the artist is to be rasterized.
Axis.get_sketch_params	Returns the sketch parameters for the artist.
Axis.get_snap	Returns the snap setting which may be:
Axis.get_transform	Return the Transform instance used by this artist.
Axis.get_transformed_clip_path_and_affine	Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.
Axis.get_url	Returns the url.
Axis.get_visible	Return the artist's visibility
Axis.get_window_extent	Get the axes bounding box in display space.
Axis.get_zorder	Return the artist's zorder.
Axis.have_units	Return True if units are set on the x or y axes
Axis.is_transform_set	Returns True if Artist has a transform explicitly set.
Axis.mouseover	
Axis.pchanged	Fire an event when property changed, calling all of the registered callbacks.
Axis.pick	Process pick event
Axis.pickable	Return True if Artist is pickable.
Axis.properties	return a dictionary mapping property name -> value for all Artist props
Axis.remove	Remove the artist from the figure if possible.
Axis.remove_callback	Remove a callback based on its id.
Axis.set	A property batch setter.
Axis.set_agg_filter	Set the agg filter.
Axis.set_alpha	Set the alpha value used for blending - not supported on all backends.
Axis.set_animated	Set the artist's animation state.
Axis.set_clip_box	Set the artist's clip Bbox.

Continued on next page

Table 20 – continued from previous page

Axis.set_clip_on	Set whether artist uses clipping.
Axis.set_clip_path	Set the artist's clip path, which may be:
Axis.set_contains	Replace the contains test used by this artist.
Axis.set_figure	Set the Figure instance the artist belongs to.
Axis.set_gid	Sets the (group) id for the artist.
Axis.set_label	Set the label to s for auto legend.
Axis.set_path_effects	Set the path effects.
Axis.set_picker	Set the epsilon for picking used by this artist
Axis.set_rasterized	Force rasterized (bitmap) drawing in vector backend output.
Axis.set_sketch_params	Sets the sketch parameters.
Axis.set_snap	Sets the snap setting which may be:
Axis.set_transform	Set the artist transform.
Axis.set_url	Sets the url for the artist.
Axis.set_visible	Set the artist's visibility.
Axis.set_zorder	Set the zorder for the artist.
Axis.stale	If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.
Axis.update	Update this artist's properties from the dictionary prop.
Axis.update_from	Copy properties from other to self.
Axis.zorder	
XAxis.add_callback	Adds a callback function that will be called whenever one of the Artist's properties changes.
XAxis.aname	
XAxis.axes	The Axes instance the artist resides in, or None.
XAxis.contains	Test whether the mouse event occurred in the x axis.
XAxis.convert_xunits	For artists in an axes, if the xaxis has units support, convert x using xaxis unit type
XAxis.convert_yunits	For artists in an axes, if the yaxis has units support, convert y using yaxis unit type
XAxis.draw	Draw the axis lines, grid lines, tick lines and labels
XAxis.findobj	Find artist objects.
XAxis.format_cursor_data	Return cursor data string formatted.
XAxis.get_agg_filter	Return filter function to be used for agg filter.
XAxis.get_alpha	Return the alpha value used for blending - not supported on all backends
XAxis.get_animated	Return the artist's animated state
XAxis.get_children	Return a list of the child Artist's this :class:`Artist` contains.

Continued on next page

Table 20 – continued from previous page

XAxis.get_clip_box	Return artist clipbox
XAxis.get_clip_on	Return whether artist uses clipping
XAxis.get_clip_path	Return artist clip path
XAxis.get_contains	Return the _contains test used by the artist, or None for default.
XAxis.get_cursor_data	Get the cursor data for a given event.
XAxis.get_figure	Return the Figure instance the artist belongs to.
XAxis.get_gid	Returns the group id.
XAxis.get_label	Return the axis label as a Text instance
XAxis.get_path_effects	
XAxis.get_picker	Return the picker object used by this artist.
XAxis.get_rasterized	Return whether the artist is to be rasterized.
XAxis.get_sketch_params	Returns the sketch parameters for the artist.
XAxis.get_snap	Returns the snap setting which may be:
XAxis.get_transform	Return the Transform instance used by this artist.
XAxis.get_transformed_clip_path_and_affine	Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.
XAxis.get_url	Returns the url.
XAxis.get_visible	Return the artist's visibility
XAxis.get_window_extent	Get the axes bounding box in display space.
XAxis.get_zorder	Return the artist's zorder.
XAxis.have_units	Return True if units are set on the x or y axes
XAxis.is_transform_set	Returns True if Artist has a transform explicitly set.
XAxis.mouseover	
XAxis.pchanged	Fire an event when property changed, calling all of the registered callbacks.
XAxis.pick	Process pick event
XAxis.pickable	Return True if Artist is pickable.
XAxis.properties	return a dictionary mapping property name -> value for all Artist props
XAxis.remove	Remove the artist from the figure if possible.
XAxis.remove_callback	Remove a callback based on its id.
XAxis.set	A property batch setter.
XAxis.set_agg_filter	Set the agg filter.
XAxis.set_alpha	Set the alpha value used for blending - not supported on all backends.
XAxis.set_animated	Set the artist's animation state.
XAxis.set_clip_box	Set the artist's clip Bbox.
XAxis.set_clip_on	Set whether artist uses clipping.
XAxis.set_clip_path	Set the artist's clip path, which may be:
XAxis.set_contains	Replace the contains test used by this artist.

Continued on next page

Table 20 – continued from previous page

XAxis.set_figure	Set the Figure instance the artist belongs to.
XAxis.set_gid	Sets the (group) id for the artist.
XAxis.set_label	Set the label to s for auto legend.
XAxis.set_path_effects	Set the path effects.
XAxis.set_picker	Set the epsilon for picking used by this artist
XAxis.set_rasterized	Force rasterized (bitmap) drawing in vector backend output.
XAxis.set_sketch_params	Sets the sketch parameters.
XAxis.set_snap	Sets the snap setting which may be:
XAxis.set_transform	Set the artist transform.
XAxis.set_url	Sets the url for the artist.
XAxis.set_visible	Set the artist's visibility.
XAxis.set_zorder	Set the zorder for the artist.
XAxis.stale	If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.
XAxis.update	Update this artist's properties from the dictionary prop.
XAxis.update_from	Copy properties from other to self.
XAxis.zorder	
YAxis.add_callback	Adds a callback function that will be called whenever one of the Artist's properties changes.
YAxis.aname	
YAxis.axes	The Axes instance the artist resides in, or None.
YAxis.contains	Test whether the mouse event occurred in the y axis.
YAxis.convert_xunits	For artists in an axes, if the xaxis has units support, convert x using xaxis unit type
YAxis.convert_yunits	For artists in an axes, if the yaxis has units support, convert y using yaxis unit type
YAxis.draw	Draw the axis lines, grid lines, tick lines and labels
YAxis.findobj	Find artist objects.
YAxis.format_cursor_data	Return cursor data string formatted.
YAxis.get_agg_filter	Return filter function to be used for agg filter.
YAxis.get_alpha	Return the alpha value used for blending - not supported on all backends
YAxis.get_animated	Return the artist's animated state
YAxis.get_children	Return a list of the child Artist's this :class:`Artist` contains.
YAxis.get_clip_box	Return artist clipbox
YAxis.get_clip_on	Return whether artist uses clipping
YAxis.get_clip_path	Return artist clip path

Continued on next page

Table 20 – continued from previous page

<code>YAxis.get_contains</code>	Return the <code>_contains</code> test used by the artist, or <code>None</code> for default.
<code>YAxis.get_cursor_data</code>	Get the cursor data for a given event.
<code>YAxis.get_figure</code>	Return the <code>Figure</code> instance the artist belongs to.
<code>YAxis.get_gid</code>	Returns the group id.
<code>YAxis.get_label</code>	Return the axis label as a <code>Text</code> instance
<code>YAxis.get_path_effects</code>	
<code>YAxis.get_picker</code>	Return the picker object used by this artist.
<code>YAxis.get_rasterized</code>	Return whether the artist is to be rasterized.
<code>YAxis.get_sketch_params</code>	Returns the sketch parameters for the artist.
<code>YAxis.get_snap</code>	Returns the snap setting which may be:
<code>YAxis.get_transform</code>	Return the <code>Transform</code> instance used by this artist.
<code>YAxis.get_transformed_clip_path_and_affine</code>	Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.
<code>YAxis.get_url</code>	Returns the url.
<code>YAxis.get_visible</code>	Return the artist's visibility
<code>YAxis.get_window_extent</code>	Get the axes bounding box in display space.
<code>YAxis.get_zorder</code>	Return the artist's zorder.
<code>YAxis.have_units</code>	Return <code>True</code> if units are set on the x or y axes
<code>YAxis.is_transform_set</code>	Returns <code>True</code> if Artist has a transform explicitly set.
<code>YAxis.mouseover</code>	
<code>YAxis.pchanged</code>	Fire an event when property changed, calling all of the registered callbacks.
<code>YAxis.pick</code>	Process pick event
<code>YAxis.pickable</code>	Return <code>True</code> if Artist is pickable.
<code>YAxis.properties</code>	return a dictionary mapping property name -> value for all Artist props
<code>YAxis.remove</code>	Remove the artist from the figure if possible.
<code>YAxis.remove_callback</code>	Remove a callback based on its id.
<code>YAxis.set</code>	A property batch setter.
<code>YAxis.set_agg_filter</code>	Set the agg filter.
<code>YAxis.set_alpha</code>	Set the alpha value used for blending - not supported on all backends.
<code>YAxis.set_animated</code>	Set the artist's animation state.
<code>YAxis.set_clip_box</code>	Set the artist's clip Bbox.
<code>YAxis.set_clip_on</code>	Set whether artist uses clipping.
<code>YAxis.set_clip_path</code>	Set the artist's clip path, which may be:
<code>YAxis.set_contains</code>	Replace the contains test used by this artist.
<code>YAxis.set_figure</code>	Set the <code>Figure</code> instance the artist belongs to.
<code>YAxis.set_gid</code>	Sets the (group) id for the artist.
<code>YAxis.set_label</code>	Set the label to s for auto legend.

Continued on next page

Table 20 – continued from previous page

<code>YAxis.set_path_effects</code>	Set the path effects.
<code>YAxis.set_picker</code>	Set the epsilon for picking used by this artist
<code>YAxis.set_rasterized</code>	Force rasterized (bitmap) drawing in vector backend output.
<code>YAxis.set_sketch_params</code>	Sets the sketch parameters.
<code>YAxis.set_snap</code>	Sets the snap setting which may be:
<code>YAxis.set_transform</code>	Set the artist transform.
<code>YAxis.set_url</code>	Sets the url for the artist.
<code>YAxis.set_visible</code>	Set the artist's visibility.
<code>YAxis.set_zorder</code>	Set the zorder for the artist.
<code>YAxis.stale</code>	If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.
<code>YAxis.update</code>	Update this artist's properties from the dictionary prop.
<code>YAxis.update_from</code>	Copy properties from other to self.
<code>YAxis.zorder</code>	

`matplotlib.axis.Axis.add_callback`

`Axis.add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`matplotlib.axis.Axis.aname`

`Axis.aname = 'Artist'`

`matplotlib.axis.Axis.axes`

`Axis.axes`

The Axes instance the artist resides in, or None.

`matplotlib.axis.Axis.contains`

`Axis.contains(mouseevent)`

Test whether the artist contains the mouse event.

Returns the truth value and a dictionary of artist specific details of selection, such as which points are contained in the pick radius. See individual artists for details.

`matplotlib.axis.Axis.convert_xunits`

`Axis.convert_xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`matplotlib.axis.Axis.convert_yunits`

`Axis.convert_yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`matplotlib.axis.Axis.draw`

`Axis.draw(renderer, *args, **kwargs)`

Draw the axis lines, grid lines, tick lines and labels

`matplotlib.axis.Axis.findobj`

`Axis.findobj(match=None, include_self=True)`

Find artist objects.

Recursively find all Artist instances contained in self.

match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is True (default), include self in the list to be checked for a match.

`matplotlib.axis.Axis.format_cursor_data`

`Axis.format_cursor_data(data)`

Return cursor data string formatted.

`matplotlib.axis.Axis.get_agg_filter`

`Axis.get_agg_filter()`

Return filter function to be used for agg filter.

`matplotlib.axis.Axis.get__alpha`

`Axis.get__alpha()`

Return the alpha value used for blending - not supported on all backends

`matplotlib.axis.Axis.get__animated`

`Axis.get__animated()`

Return the artist's animated state

`matplotlib.axis.Axis.get__children`

`Axis.get__children()`

Return a list of the child Artist's this :class:`Artist` contains.

`matplotlib.axis.Axis.get__clip__box`

`Axis.get__clip__box()`

Return artist clipbox

`matplotlib.axis.Axis.get__clip__on`

`Axis.get__clip__on()`

Return whether artist uses clipping

`matplotlib.axis.Axis.get__clip__path`

`Axis.get__clip__path()`

Return artist clip path

`matplotlib.axis.Axis.get__contains`

`Axis.get__contains()`

Return the __contains test used by the artist, or None for default.

`matplotlib.axis.Axis.get__cursor__data`

`Axis.get__cursor__data(event)`

Get the cursor data for a given event.

`matplotlib.axis.Axis.get_figure`

`Axis.get_figure()`

Return the Figure instance the artist belongs to.

`matplotlib.axis.Axis.get_gid`

`Axis.get_gid()`

Returns the group id.

`matplotlib.axis.Axis.get_label`

`Axis.get_label()`

Return the axis label as a Text instance

`matplotlib.axis.Axis.get_path_effects`

`Axis.get_path_effects()`

`matplotlib.axis.Axis.get_picker`

`Axis.get_picker()`

Return the picker object used by this artist.

`matplotlib.axis.Axis.get_rasterized`

`Axis.get_rasterized()`

Return whether the artist is to be rasterized.

`matplotlib.axis.Axis.get_sketch_params`

`Axis.get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

`sketch_params` [tuple or `None`]

A 3-tuple with the following elements:

- `scale`: The amplitude of the wiggle perpendicular to the source line.
- `length`: The length of the wiggle along the line.

- randomness: The scale factor by which the length is shrunk or expanded.

May return ‘None’ if no sketch parameters were set.

`matplotlib.axis.Axis.get__snap`

`Axis.get__snap()`

Returns the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`matplotlib.axis.Axis.get__transform`

`Axis.get__transform()`

Return the Transform instance used by this artist.

`matplotlib.axis.Axis.get__transformed_clip_path_and__affine`

`Axis.get__transformed_clip_path_and__affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`matplotlib.axis.Axis.get__url`

`Axis.get__url()`

Returns the url.

`matplotlib.axis.Axis.get__visible`

`Axis.get__visible()`

Return the artist’s visibility

`matplotlib.axis.Axis.get__window_extent`

`Axis.get__window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box “tight” calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`matplotlib.axis.Axis.get_zorder`

`Axis.get_zorder()`
Return the artist's zorder.

`matplotlib.axis.Axis.have_units`

`Axis.have_units()`
Return True if units are set on the x or y axes

`matplotlib.axis.Axis.is_transform_set`

`Axis.is_transform_set()`
Returns True if Artist has a transform explicitly set.

`matplotlib.axis.Axis.mouseover`

`Axis.mouseover`

`matplotlib.axis.Axis.pchanged`

`Axis.pchanged()`
Fire an event when property changed, calling all of the registered callbacks.

`matplotlib.axis.Axis.pick`

`Axis.pick(mouseevent)`
Process pick event

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

`matplotlib.axis.Axis.pickable`

`Axis.pickable()`

Return True if Artist is pickable.

`matplotlib.axis.Axis.properties`

`Axis.properties()`

return a dictionary mapping property name -> value for all Artist props

`matplotlib.axis.Axis.remove`

`Axis.remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`matplotlib.axis.Axis.remove_callback`

`Axis.remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`matplotlib.axis.Axis.set`

`Axis.set(**kwargs)`

A property batch setter. Pass kwargs to set properties.

`matplotlib.axis.Axis.set_agg_filter`

`Axis.set_agg_filter(filter_func)`

Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`matplotlib.axis.Axis.set_alpha`

`Axis.set_alpha(alpha)`

Set the alpha value used for blending - not supported on all backends.

Parameters

alpha [float]

`matplotlib.axis.Axis.set_animated`

`Axis.set_animated(b)`

Set the artist's animation state.

Parameters

b [bool]

`matplotlib.axis.Axis.set_clip_box`

`Axis.set_clip_box(clipbox)`

Set the artist's clip Bbox.

Parameters

clipbox [Bbox]

`matplotlib.axis.Axis.set_clip_on`

`Axis.set_clip_on(b)`

Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

b [bool]

`matplotlib.axis.Axis.set_clip_path`

`Axis.set_clip_path(clippath, transform=None)`

Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

`matplotlib.axis.Axis.set__contains`

`Axis.set__contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

`picker` [callable]

`matplotlib.axis.Axis.set__figure`

`Axis.set__figure(fig)`

Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`matplotlib.axis.Axis.set__gid`

`Axis.set__gid(gid)`

Sets the (group) id for the artist.

Parameters

`gid` [str]

`matplotlib.axis.Axis.set__label`

`Axis.set__label(s)`

Set the label to `s` for auto legend.

Parameters

`s` [object] `s` will be converted to a string by calling `str` (unicode on Py2).

`matplotlib.axis.Axis.set_path_effects`

`Axis.set_path_effects(path_effects)`

Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`matplotlib.axis.Axis.set_picker`

`Axis.set_picker(picker)`

Set the epsilon for picking used by this artist

`picker` can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if `picker` is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if `picker` is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

`hit, props = picker(artist, mouseevent)`

to determine the hit test. if the mouse event is over the artist, return `hit=True` and `props` is a dictionary of properties you want added to the `PickEvent` attributes.

Parameters

`picker` [None or bool or float or callable]

`matplotlib.axis.Axis.set_rasterized`

`Axis.set_rasterized(rasterized)`

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

`rasterized` [bool or None]

`matplotlib.axis.Axis.set_sketch_params`

`Axis.set_sketch_params(scale=None, length=None, randomness=None)`

Sets the sketch parameters.

Parameters

`scale` [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If `scale` is `None`, or not provided, no sketch filter will be provided.

`length` [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

`randomness` [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

`matplotlib.axis.Axis.set_snap`

`Axis.set_snap(snap)`

Sets the snap setting which may be:

- `True`: snap vertices to the nearest pixel center
- `False`: leave vertices as-is
- `None`: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

`snap` [bool or None]

`matplotlib.axis.Axis.set_transform`

`Axis.set_transform(t)`

Set the artist transform.

Parameters

`t` [Transform]

`matplotlib.axis.Axis.set_url`

`Axis.set_url(url)`

Sets the url for the artist.

Parameters

url [str]

matplotlib.axis.Axis.set_visible

Axis.set_visible(b)

Set the artist's visibility.

Parameters

b [bool]

matplotlib.axis.Axis.set_zorder

Axis.set_zorder(level)

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

level [float]

matplotlib.axis.Axis.stale

Axis.stale

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

matplotlib.axis.Axis.update

Axis.update(props)

Update this artist's properties from the dictionary prop.

matplotlib.axis.Axis.update_from

Axis.update_from(other)

Copy properties from other to self.

matplotlib.axis.Axis.zorder

Axis.zorder = 0

`matplotlib.axis.XAxis.add_callback`

`XAxis.add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`matplotlib.axis.XAxis.aname`

`XAxis.aname = 'Artist'`

`matplotlib.axis.XAxis.axes`

`XAxis.axes`

The Axes instance the artist resides in, or None.

`matplotlib.axis.XAxis.contains`

`XAxis.contains(mouseevent)`

Test whether the mouse event occurred in the x axis.

`matplotlib.axis.XAxis.convert_xunits`

`XAxis.convert_xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`matplotlib.axis.XAxis.convert_yunits`

`XAxis.convert_yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`matplotlib.axis.XAxis.draw`

`XAxis.draw(renderer, *args, **kwargs)`

Draw the axis lines, grid lines, tick lines and labels

`matplotlib.axis.XAxis.findobj`

`XAxis.findobj(match=None, include_self=True)`

Find artist objects.

Recursively find all Artist instances contained in self.

match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is `True` (default), include self in the list to be checked for a match.

`matplotlib.axis.XAxis.format_cursor_data`

`XAxis.format_cursor_data(data)`
Return cursor data string formatted.

`matplotlib.axis.XAxis.get_agg_filter`

`XAxis.get_agg_filter()`
Return filter function to be used for agg filter.

`matplotlib.axis.XAxis.get_alpha`

`XAxis.get_alpha()`
Return the alpha value used for blending - not supported on all backends

`matplotlib.axis.XAxis.get_animated`

`XAxis.get_animated()`
Return the artist's animated state

`matplotlib.axis.XAxis.get_children`

`XAxis.get_children()`
Return a list of the child Artist's this :class:`Artist` contains.

`matplotlib.axis.XAxis.get_clip_box`

`XAxis.get_clip_box()`
Return artist clipbox

`matplotlib.axis.XAxis.get_clip_on`

`XAxis.get_clip_on()`

Return whether artist uses clipping

`matplotlib.axis.XAxis.get_clip_path`

`XAxis.get_clip_path()`

Return artist clip path

`matplotlib.axis.XAxis.get_contains`

`XAxis.get_contains()`

Return the `_contains` test used by the artist, or `None` for default.

`matplotlib.axis.XAxis.get_cursor_data`

`XAxis.get_cursor_data(event)`

Get the cursor data for a given event.

`matplotlib.axis.XAxis.get_figure`

`XAxis.get_figure()`

Return the Figure instance the artist belongs to.

`matplotlib.axis.XAxis.get_gid`

`XAxis.get_gid()`

Returns the group id.

`matplotlib.axis.XAxis.get_label`

`XAxis.get_label()`

Return the axis label as a Text instance

`matplotlib.axis.XAxis.get_path_effects`

`XAxis.get_path_effects()`

`matplotlib.axis.XAxis.get_picker`

`XAxis.get_picker()`

Return the picker object used by this artist.

`matplotlib.axis.XAxis.get_rasterized`

`XAxis.get_rasterized()`

Return whether the artist is to be rasterized.

`matplotlib.axis.XAxis.get_sketch_params`

`XAxis.get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

sketch_params [tuple or `None`]

A 3-tuple with the following elements:

- scale: The amplitude of the wiggle perpendicular to the source line.
- length: The length of the wiggle along the line.
- randomness: The scale factor by which the length is shrunk or expanded.

May return ‘None’ if no sketch parameters were set.

`matplotlib.axis.XAxis.get_snap`

`XAxis.get_snap()`

Returns the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`matplotlib.axis.XAxis.get_transform`

`XAxis.get_transform()`

Return the Transform instance used by this artist.

`matplotlib.axis.XAxis.get_transformed_clip_path_and_affine`

`XAxis.get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`matplotlib.axis.XAxis.get_url`

`XAxis.get_url()`

Returns the url.

`matplotlib.axis.XAxis.get_visible`

`XAxis.get_visible()`

Return the artist's visibility

`matplotlib.axis.XAxis.get_window_extent`

`XAxis.get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`matplotlib.axis.XAxis.get_zorder`

`XAxis.get_zorder()`

Return the artist's zorder.

`matplotlib.axis.XAxis.have_units`

`XAxis.have_units()`

Return True if units are set on the x or y axes

`matplotlib.axis.XAxis.is_transform_set`

`XAxis.is_transform_set()`

Returns True if Artist has a transform explicitly set.

`matplotlib.axis.XAxis.mouseover`

`XAxis.mouseover`

`matplotlib.axis.XAxis.pchanged`

`XAxis.pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`matplotlib.axis.XAxis.pick`

`XAxis.pick(mouseevent)`

Process pick event

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

`matplotlib.axis.XAxis.pickable`

`XAxis.pickable()`

Return True if Artist is pickable.

`matplotlib.axis.XAxis.properties`

`XAxis.properties()`

return a dictionary mapping property name -> value for all Artist props

`matplotlib.axis.XAxis.remove`

`XAxis.remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`matplotlib.axis.XAxis.remove__callback`

`XAxis.remove__callback(oid)`

Remove a callback based on its id.

See also:

`add__callback()` For adding callbacks

`matplotlib.axis.XAxis.set`

`XAxis.set(**kwargs)`

A property batch setter. Pass kwargs to set properties.

`matplotlib.axis.XAxis.set__agg__filter`

`XAxis.set__agg__filter(filter_func)`

Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`matplotlib.axis.XAxis.set__alpha`

`XAxis.set__alpha(alpha)`

Set the alpha value used for blending - not supported on all backends.

Parameters

`alpha` [float]

`matplotlib.axis.XAxis.set__animated`

`XAxis.set__animated(b)`

Set the artist's animation state.

Parameters

`b` [bool]

`matplotlib.axis.XAxis.set__clip__box`

`XAxis.set__clip__box(clipbox)`

Set the artist's clip Bbox.

Parameters

clipbox [Bbox]

`matplotlib.axis.XAxis.set_clip_on`

`XAxis.set_clip_on(b)`

Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

b [bool]

`matplotlib.axis.XAxis.set_clip_path`

`XAxis.set_clip_path(clippath, transform=None)`

Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

`matplotlib.axis.XAxis.set_contains`

`XAxis.set_contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

hit, props = picker(artist, mouseevent)

If the mouse event is over the artist, return hit = True and props is a dictionary of properties you want returned with the contains test.

Parameters

picker [callable]

`matplotlib.axis.XAxis.set_figure`

`XAxis.set_figure(fig)`

Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`matplotlib.axis.XAxis.set_gid`

`XAxis.set_gid(gid)`

Sets the (group) id for the artist.

Parameters

`gid` [str]

`matplotlib.axis.XAxis.set_label`

`XAxis.set_label(s)`

Set the label to `s` for auto legend.

Parameters

`s` [object] `s` will be converted to a string by calling `str` (unicode on Py2).

`matplotlib.axis.XAxis.set_path_effects`

`XAxis.set_path_effects(path_effects)`

Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`matplotlib.axis.XAxis.set_picker`

`XAxis.set_picker(picker)`

Set the epsilon for picking used by this artist

`picker` can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist

- A float: if picker is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if picker is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

to determine the hit test. if the mouse event is over the artist, return hit=True and props is a dictionary of properties you want added to the PickEvent attributes.

Parameters

picker [None or bool or float or callable]

matplotlib.axis.XAxis.set_rasterized

XAxis.set_rasterized(rasterized)

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

rasterized [bool or None]

matplotlib.axis.XAxis.set_sketch_params

XAxis.set_sketch_params(scale=None, length=None, randomness=None)

Sets the sketch parameters.

Parameters

scale [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If scale is `None`, or not provided, no sketch filter will be provided.

length [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

randomness [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

matplotlib.axis.XAxis.set_snap

XAxis.set_snap(snap)

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

 snap [bool or None]

`matplotlib.axis.XAxis.set__transform`

`XAxis.set__transform(t)`

Set the artist transform.

Parameters

 t [Transform]

`matplotlib.axis.XAxis.set__url`

`XAxis.set__url(url)`

Sets the url for the artist.

Parameters

 url [str]

`matplotlib.axis.XAxis.set__visible`

`XAxis.set__visible(b)`

Set the artist's visibility.

Parameters

 b [bool]

`matplotlib.axis.XAxis.set__zorder`

`XAxis.set__zorder(level)`

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

 level [float]

`matplotlib.axis.XAxis.stale`

`XAxis.stale`

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

`matplotlib.axis.XAxis.update`

`XAxis.update(props)`

Update this artist's properties from the dictionary prop.

`matplotlib.axis.XAxis.update_from`

`XAxis.update_from(other)`

Copy properties from other to self.

`matplotlib.axis.XAxis.zorder`

`XAxis.zorder = 0`

`matplotlib.axis.YAxis.add_callback`

`YAxis.add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`matplotlib.axis.YAxis.aname`

`YAxis.aname = 'Artist'`

`matplotlib.axis.YAxis.axes`

`YAxis.axes`

The Axes instance the artist resides in, or None.

`matplotlib.axis.YAxis.contains`

`YAxis.contains(mouseevent)`

Test whether the mouse event occurred in the y axis.

Returns True | False

`matplotlib.axis.YAxis.convert_xunits`

`YAxis.convert_xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`matplotlib.axis.YAxis.convert_yunits`

`YAxis.convert_yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`matplotlib.axis.YAxis.draw`

`YAxis.draw(renderer, *args, **kwargs)`

Draw the axis lines, grid lines, tick lines and labels

`matplotlib.axis.YAxis.findobj`

`YAxis.findobj(match=None, include_self=True)`

Find artist objects.

Recursively find all Artist instances contained in self.

match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is True (default), include self in the list to be checked for a match.

`matplotlib.axis.YAxis.format_cursor_data`

`YAxis.format_cursor_data(data)`

Return cursor data string formatted.

`matplotlib.axis.YAxis.get_agg_filter`

`YAxis.get_agg_filter()`

Return filter function to be used for agg filter.

`matplotlib.axis.YAxis.get_alpha`

`YAxis.get_alpha()`

Return the alpha value used for blending - not supported on all backends

`matplotlib.axis.YAxis.get_animated`

`YAxis.get_animated()`

Return the artist's animated state

`matplotlib.axis.YAxis.get_children`

`YAxis.get_children()`

Return a list of the child Artist's this :class:`Artist` contains.

`matplotlib.axis.YAxis.get_clip_box`

`YAxis.get_clip_box()`

Return artist clipbox

`matplotlib.axis.YAxis.get_clip_on`

`YAxis.get_clip_on()`

Return whether artist uses clipping

`matplotlib.axis.YAxis.get_clip_path`

`YAxis.get_clip_path()`

Return artist clip path

`matplotlib.axis.YAxis.get_contains`

`YAxis.get_contains()`

Return the `_contains` test used by the artist, or `None` for default.

`matplotlib.axis.YAxis.get_cursor_data`

`YAxis.get_cursor_data(event)`
Get the cursor data for a given event.

`matplotlib.axis.YAxis.get_figure`

`YAxis.get_figure()`
Return the Figure instance the artist belongs to.

`matplotlib.axis.YAxis.get_gid`

`YAxis.get_gid()`
Returns the group id.

`matplotlib.axis.YAxis.get_label`

`YAxis.get_label()`
Return the axis label as a Text instance

`matplotlib.axis.YAxis.get_path_effects`

`YAxis.get_path_effects()`

`matplotlib.axis.YAxis.get_picker`

`YAxis.get_picker()`
Return the picker object used by this artist.

`matplotlib.axis.YAxis.get_rasterized`

`YAxis.get_rasterized()`
Return whether the artist is to be rasterized.

`matplotlib.axis.YAxis.get_sketch_params`

`YAxis.get_sketch_params()`
Returns the sketch parameters for the artist.

Returns

sketch_params [tuple or `None`]

A 3-tuple with the following elements:

- scale: The amplitude of the wiggle perpendicular to the source line.
- length: The length of the wiggle along the line.
- randomness: The scale factor by which the length is shrunk or expanded.

May return ‘None’ if no sketch parameters were set.

`matplotlib.axis.YAxis.get_snap`

`YAxis.get_snap()`

Returns the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`matplotlib.axis.YAxis.get_transform`

`YAxis.get_transform()`

Return the Transform instance used by this artist.

`matplotlib.axis.YAxis.get_transformed_clip_path_and_affine`

`YAxis.get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`matplotlib.axis.YAxis.get_url`

`YAxis.get_url()`

Returns the url.

`matplotlib.axis.YAxis.get_visible`

`YAxis.get_visible()`

Return the artist’s visibility

`matplotlib.axis.YAxis.get_window_extent`

`YAxis.get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`matplotlib.axis.YAxis.get_zorder`

`YAxis.get_zorder()`

Return the artist's zorder.

`matplotlib.axis.YAxis.have_units`

`YAxis.have_units()`

Return True if units are set on the x or y axes

`matplotlib.axis.YAxis.is_transform_set`

`YAxis.is_transform_set()`

Returns True if Artist has a transform explicitly set.

`matplotlib.axis.YAxis.mouseover`

`YAxis.mouseover`

`matplotlib.axis.YAxis.pchanged`

`YAxis.pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`matplotlib.axis.YAxis.pick`

`YAxis.pick(mouseevent)`

Process pick event

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

`matplotlib.axis.YAxis.pickable`

`YAxis.pickable()`

Return True if Artist is pickable.

`matplotlib.axis.YAxis.properties`

`YAxis.properties()`

return a dictionary mapping property name -> value for all Artist props

`matplotlib.axis.YAxis.remove`

`YAxis.remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`matplotlib.axis.YAxis.remove_callback`

`YAxis.remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`matplotlib.axis.YAxis.set`

`YAxis.set(**kwargs)`

A property batch setter. Pass kwargs to set properties.

`matplotlib.axis.YAxis.set_agg_filter`

`YAxis.set_agg_filter(filter_func)`

Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`matplotlib.axis.YAxis.set_alpha`

`YAxis.set_alpha(alpha)`

Set the alpha value used for blending - not supported on all backends.

Parameters

`alpha` [float]

`matplotlib.axis.YAxis.set_animated`

`YAxis.set_animated(b)`

Set the artist's animation state.

Parameters

`b` [bool]

`matplotlib.axis.YAxis.set_clip_box`

`YAxis.set_clip_box(clipbox)`

Set the artist's clip Bbox.

Parameters

`clipbox` [Bbox]

`matplotlib.axis.YAxis.set_clip_on`

`YAxis.set_clip_on(b)`

Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

`b` [bool]

`matplotlib.axis.YAxis.set_clip_path`

`YAxis.set_clip_path(clippath, transform=None)`

Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

`matplotlib.axis.YAxis.set__contains`

`YAxis.set__contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

`picker` [callable]

`matplotlib.axis.YAxis.set__figure`

`YAxis.set__figure(fig)`

Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`matplotlib.axis.YAxis.set__gid`

`YAxis.set__gid(gid)`

Sets the (group) id for the artist.

Parameters

`gid` [str]

`matplotlib.axis.YAxis.set__label`

`YAxis.set__label(s)`

Set the label to `s` for auto legend.

Parameters

s [object] s will be converted to a string by calling `str` (unicode on Py2).

`matplotlib.axis.YAxis.set__path_effects``YAxis.set__path_effects(path_effects)`

Set the path effects.

Parameters

path_effects [AbstractPathEffect]

`matplotlib.axis.YAxis.set__picker``YAxis.set__picker(picker)`

Set the epsilon for picking used by this artist

picker can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if picker is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if picker is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

`hit, props = picker(artist, mouseevent)`

to determine the hit test. if the mouse event is over the artist, return `hit=True` and `props` is a dictionary of properties you want added to the `PickEvent` attributes.

Parameters

picker [None or bool or float or callable]

`matplotlib.axis.YAxis.set__rasterized``YAxis.set__rasterized(rasterized)`

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

rasterized [bool or None]

matplotlib.axis.YAxis.set_sketch_params

YAxis.set_sketch_params(scale=None, length=None, randomness=None)

Sets the sketch parameters.

Parameters

scale [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If scale is `None`, or not provided, no sketch filter will be provided.

length [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

randomness [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

matplotlib.axis.YAxis.set_snap

YAxis.set_snap(snap)

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

snap [bool or None]

matplotlib.axis.YAxis.set_transform

YAxis.set_transform(t)

Set the artist transform.

Parameters

t [Transform]

matplotlib.axis.YAxis.set_url

YAxis.set_url(url)

Sets the url for the artist.

Parameters

url [str]

`matplotlib.axis.YAxis.set__visible`

`YAxis.set__visible(b)`

Set the artist's visibility.

Parameters

b [bool]

`matplotlib.axis.YAxis.set__zorder`

`YAxis.set__zorder(level)`

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

level [float]

`matplotlib.axis.YAxis.stale`

`YAxis.stale`

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

`matplotlib.axis.YAxis.update`

`YAxis.update(props)`

Update this artist's properties from the dictionary prop.

`matplotlib.axis.YAxis.update__from`

`YAxis.update__from(other)`

Copy properties from other to self.

`matplotlib.axis.YAxis.zorder`

`YAxis.zorder = 0`

37.1 matplotlib.backend_bases

Abstract base classes define the primitives that renderers and graphics contexts must implement to serve as a matplotlib backend

RendererBase An abstract base class to handle drawing/rendering operations.

FigureCanvasBase The abstraction layer that separates the matplotlib.figure.Figure from the backend specific details like a user interface drawing area

GraphicsContextBase An abstract base class that provides color, line styles, etc...

Event The base class for all of the matplotlib event handling. Derived classes such as KeyEvent and MouseEvent store the meta data like keys and buttons pressed, x and y locations in pixel and Axes coordinates.

ShowBase The base class for the Show class of each interactive backend; the 'show' callable is then set to Show.__call__, inherited from ShowBase.

ToolContainerBase The base class for the Toolbar class of each interactive backend.

StatusbarBase The base class for the messaging area.

```
class matplotlib.backend_bases.CloseEvent(name, canvas, guiEvent=None)
    Bases: matplotlib.backend_bases.Event
```

An event triggered by a figure being closed

```
class matplotlib.backend_bases.DrawEvent(name, canvas, renderer)
    Bases: matplotlib.backend_bases.Event
```

An event triggered by a draw operation on the canvas

In most backends callbacks subscribed to this callback will be fired after the rendering is complete but before the screen is updated. Any extra artists drawn to the canvas's renderer will be reflected without an explicit call to blit.

Warning: Calling canvas.draw and canvas.blit in these callbacks may not be safe with all backends and may cause infinite recursion.

In addition to the Event attributes, the following event attributes are defined:

Attributes

renderer [RendererBase] the renderer for the draw event

class matplotlib.backend_bases.Event(name, canvas, guiEvent=None)

Bases: `object`

A matplotlib event. Attach additional attributes as defined in FigureCanvasBase.mpl_connect(). The following attributes are defined and shown with their default values

Attributes

name [str] the event name

canvas [FigureCanvasBase] the backend-specific canvas instance generating the event

guiEvent the GUI event that triggered the matplotlib event

class matplotlib.backend_bases.FigureCanvasBase(figure)

Bases: `object`

The canvas the figure renders into.

Public attributes

Attributes

figure [matplotlib.figure.Figure] A high-level figure instance

blit(bbox=None)

Blit the canvas in bbox (default entire canvas).

button_press_event(x, y, button, dblclick=False, guiEvent=None)

Backend derived classes should call this function on any mouse button press. x,y are the canvas coords: 0,0 is lower, left. button and key are as defined in MouseEvent.

This method will be call all functions connected to the 'button_press_event' with a MouseEvent instance.

button_release_event(x, y, button, guiEvent=None)

Backend derived classes should call this function on any mouse button release.

This method will call all functions connected to the 'button_release_event' with a MouseEvent instance.

Parameters

x [scalar] the canvas coordinates where 0=left

y [scalar] the canvas coordinates where 0=bottom

guiEvent the native UI event that generated the mpl event

close_event(guiEvent=None)

Pass a CloseEvent to all functions connected to close_event.

`draw(*args, **kwargs)`
 Render the Figure.

`draw_cursor(event)`
 Draw a cursor in the `event.axes` if `inaxes` is not `None`. Use native GUI drawing for efficiency if possible

`draw_event(renderer)`
 Pass a `DrawEvent` to all functions connected to `draw_event`.

`draw_idle(*args, **kwargs)`
`draw()` only if idle; defaults to draw but backends can override

`enter_notify_event(guiEvent=None, xy=None)`
 Backend derived classes should call this function when entering canvas

Parameters

`guiEvent` the native UI event that generated the mpl event

`xy` [tuple of 2 scalars] the coordinate location of the pointer when the canvas is entered

`events` = ['resize_event', 'draw_event', 'key_press_event', 'key_release_event', 'button_press_event', 'button_release_event']

`filetypes` = {'eps': 'Encapsulated Postscript', 'jpeg': 'Joint Photographic Experts Group', 'jpg': 'Joint Photographic Experts Group'}

`fixed_dpi` = None

`flush_events()`
 Flush the GUI events for the figure.

Interactive backends need to reimplement this method.

`get_default_filename()`
 Return a string, which includes extension, suitable for use as a default filename.

`classmethod get_default_filetype()`
 Get the default savefig file format as specified in `rcParam savefig.format`. Returned string excludes period. Overridden in backends that only support a single file type.

`classmethod get_supported_filetypes()`
 Return dict of savefig file formats supported by this backend

`classmethod get_supported_filetypes_grouped()`
 Return a dict of savefig file formats supported by this backend, where the keys are a file type name, such as 'Joint Photographic Experts Group', and the values are a list of filename extensions used for that filetype, such as ['jpg', 'jpeg'].

`get_width_height()`
 Return the figure width and height in points or pixels (depending on the backend), truncated to integers

`get_window_title()`

Get the title text of the window containing the figure. Return None if there is no window (e.g., a PS backend).

`grab_mouse(ax)`

Set the child axes which are currently grabbing the mouse events. Usually called by the widgets themselves. It is an error to call this if the mouse is already grabbed by another axes.

`idle_event(guiEvent=None)`

Deprecated since version 2.1: The `idle_event` function was deprecated in version 2.1.

Called when GUI is idle.

`is_saving()`

Returns whether the renderer is in the process of saving to a file, rather than rendering for an on-screen buffer.

`key_press_event(key, guiEvent=None)`

Pass a `KeyEvent` to all functions connected to `key_press_event`.

`key_release_event(key, guiEvent=None)`

Pass a `KeyEvent` to all functions connected to `key_release_event`.

`leave_notify_event(guiEvent=None)`

Backend derived classes should call this function when leaving canvas

Parameters

`guiEvent` the native UI event that generated the mpl event

`motion_notify_event(x, y, guiEvent=None)`

Backend derived classes should call this function on any motion-notify-event.

This method will call all functions connected to the `'motion_notify_event'` with a `MouseEvent` instance.

Parameters

`x` [scalar] the canvas coordinates where 0=left

`y` [scalar] the canvas coordinates where 0=bottom

`guiEvent` the native UI event that generated the mpl event

`mpl_connect(s, func)`

Connect event with string `s` to `func`. The signature of `func` is:

`def func(event)`

where `event` is a `matplotlib.backend_bases.Event`. The following events are recognized

- `'button_press_event'`
- `'button_release_event'`
- `'draw_event'`

- 'key_press_event'
- 'key_release_event'
- 'motion_notify_event'
- 'pick_event'
- 'resize_event'
- 'scroll_event'
- 'figure_enter_event',
- 'figure_leave_event',
- 'axes_enter_event',
- 'axes_leave_event'
- 'close_event'

For the location events (button and key press/release), if the mouse is over the axes, the variable `event.inaxes` will be set to the Axes the event occurs is over, and additionally, the variables `event.xdata` and `event.ydata` will be defined. This is the mouse location in data coords. See `KeyEvent` and `MouseEvent` for more info.

Return value is a connection id that can be used with `mpl_disconnect()`.

Examples

Usage:

```
def on_press(event):
    print('you pressed', event.button, event.xdata, event.ydata)

cid = canvas.mpl_connect('button_press_event', on_press)
```

`mpl_disconnect(cid)`
Disconnect callback id `cid`

Examples

Usage:

```
cid = canvas.mpl_connect('button_press_event', on_press)
#...later
canvas.mpl_disconnect(cid)
```

`new_timer(*args, **kwargs)`
Creates a new backend-specific subclass of `backend_bases.Timer`. This is useful for getting periodic events through the backend's native event loop. Implemented only for backends with GUIs.

Other Parameters

`interval` [scalar] Timer interval in milliseconds

`callbacks` [List[Tuple[callable, Tuple, Dict]]] Sequence of (func, args, kwargs) where func(*args, **kwargs) will be executed by the timer every interval.

callbacks which return False or 0 will be removed from the timer.

Examples

```
>>> timer = fig.canvas.new_timer(callbacks=[(f1, (1, ), {'a': 3}),])
```

`onRemove(ev)`

Deprecated since version 2.2: The `onRemove` function was deprecated in version 2.2.

Mouse event processor which removes the top artist under the cursor. Connect this to the `'mouse_press_event'` using:

```
canvas.mpl_connect('mouse_press_event', canvas.onRemove)
```

`pick(mouseevent)`

`pick_event(mouseevent, artist, **kwargs)`

This method will be called by artists who are picked and will fire off `PickEvent` callbacks registered listeners

`print_figure(filename, dpi=None, facecolor=None, edgecolor=None, orientation='portrait', format=None, **kwargs)`

Render the figure to hardcopy. Set the figure patch face and edge colors. This is useful because some of the GUIs have a gray figure face color background and you'll probably want to override this on hardcopy.

Parameters

`filename` can also be a file object on image backends

`orientation` [{'landscape', 'portrait'}, optional] only currently applies to PostScript printing.

`dpi` [scalar, optional] the dots per inch to save the figure in; if None, use `savefig.dpi`

`facecolor` [color spec or None, optional] the facecolor of the figure; if None, defaults to `savefig.facecolor`

`edgecolor` [color spec or None, optional] the edgecolor of the figure; if None, defaults to `savefig.edgecolor`

`format` [str, optional] when set, forcibly set the file format to save to

`bbox_inches` [str or Bbox, optional] Bbox in inches. Only the given portion of the figure is saved. If 'tight', try to figure out the tight bbox of the figure. If None, use `savefig.bbox`

`pad_inches` [scalar, optional] Amount of padding around the figure when `bbox_inches` is 'tight'. If None, use `savefig.pad_inches`

`bbox_extra_artists` [list of Artist, optional] A list of extra artists that will be considered when the tight bbox is calculated.

`release_mouse(ax)`

Release the mouse grab held by the axes, `ax`. Usually called by the widgets. It is ok to call this even if you `ax` doesn't have the mouse grab currently.

`resize(w, h)`

Set the canvas size in pixels.

`resize_event()`

Pass a `ResizeEvent` to all functions connected to `resize_event`.

`scroll_event(x, y, step, guiEvent=None)`

Backend derived classes should call this function on any scroll wheel event. `x,y` are the canvas coords: 0,0 is lower, left. `button` and `key` are as defined in `MouseEvent`.

This method will be call all functions connected to the 'scroll_event' with a `MouseEvent` instance.

`set_window_title(title)`

Set the title text of the window containing the figure. Note that this has no effect if there is no window (e.g., a PS backend).

`start_event_loop(timeout=0)`

Start a blocking event loop.

Such an event loop is used by interactive functions, such as `ginput` and `waitforbuttonpress`, to wait for events.

The event loop blocks until a callback function triggers `stop_event_loop`, or `timeout` is reached.

If `timeout` is negative, never `timeout`.

Only interactive backends need to reimplement this method and it relies on `flush_events` being properly implemented.

Interactive backends should implement this in a more native way.

`start_event_loop_default(timeout=0)`

Deprecated since version 2.1: The `start_event_loop_default` function was deprecated in version 2.1.

Start a blocking event loop.

Such an event loop is used by interactive functions, such as `ginput` and `waitforbuttonpress`, to wait for events.

The event loop blocks until a callback function triggers `stop_event_loop`, or timeout is reached.

If timeout is negative, never timeout.

Only interactive backends need to reimplement this method and it relies on `flush_events` being properly implemented.

Interactive backends should implement this in a more native way.

`stop_event_loop()`

Stop the current blocking event loop.

Interactive backends need to reimplement this to match `start_event_loop`

`stop_event_loop_default()`

Deprecated since version 2.1: The `stop_event_loop_default` function was deprecated in version 2.1.

Stop the current blocking event loop.

Interactive backends need to reimplement this to match `start_event_loop`

`supports_blit = True`

`switch_backends(FigureCanvasClass)`

Instantiate an instance of `FigureCanvasClass`

This is used for backend switching, e.g., to instantiate a `FigureCanvasPS` from a `FigureCanvasGTK`. Note, deep copying is not done, so any changes to one of the instances (e.g., setting figure size or line props), will be reflected in the other

`class matplotlib.backend_bases.FigureManagerBase(canvas, num)`

Bases: `object`

Helper class for pyplot mode, wraps everything up into a neat bundle

Attributes

`canvas` [`FigureCanvasBase`] The backend-specific canvas instance

`num` [`int` or `str`] The figure number

`key_press_handler_id` [`int`] The default key handler cid, when using the tool-manager. Can be used to disable default key press handling

```
figure.canvas.mpl_disconnect(  
    figure.canvas.manager.key_press_handler_id)
```

`destroy()`

`full_screen_toggle()`

`get_window_title()`

Get the title text of the window containing the figure.

Return None for non-GUI (e.g., PS) backends.

`key_press(event)`

Implement the default mpl key bindings defined at [Navigation Keyboard Shortcuts](#)

`resize(w, h)`

”For GUI backends, resize the window (in pixels).

`set_window_title(title)`

Set the title text of the window containing the figure.

This has no effect for non-GUI (e.g., PS) backends.

`show()`

For GUI backends, show the figure window and redraw. For non-GUI backends, raise an exception to be caught by `show()`, for an optional warning.

`show_popup(msg)`

Deprecated since version 2.2: The `show_popup` function was deprecated in version 2.2.

Display message in a popup – GUI only.

class `matplotlib.backend_bases.GraphicsContextBase`

Bases: `object`

An abstract base class that provides color, line styles, etc...

`copy_properties(gc)`

Copy properties from `gc` to self

`get_alpha()`

Return the alpha value used for blending - not supported on all backends

`get_antialiased()`

Return true if the object should try to do antialiased rendering

`get_capstyle()`

Return the capstyle as a string in ('butt', 'round', 'projecting')

`get_clip_path()`

Return the clip path in the form (path, transform), where path is a Path instance, and transform is an affine transform to apply to the path before clipping.

`get_clip_rectangle()`

Return the clip rectangle as a Bbox instance

`get_dashes()`

Return the dash information as an offset dashlist tuple.

The dash list is a even size list that gives the ink on, ink off in pixels.

See p107 of to PostScript [BLUEBOOK](#) for more info.

Default value is None

`get_forced_alpha()`
Return whether the value given by `get_alpha()` should be used to override any other alpha-channel values.

`get_gid()`
Return the object identifier if one is set, `None` otherwise.

`get_hatch()`
Gets the current hatch style

`get_hatch_color()`
Gets the color to use for hatching.

`get_hatch_linewidth()`
Gets the linewidth to use for hatching.

`get_hatch_path(density=6.0)`
Returns a `Path` for the current hatch.

`get_joinstyle()`
Return the line join style as one of ('miter', 'round', 'bevel')

`get_linestyle()`
Deprecated since version 2.1: The `get_linestyle` function was deprecated in version 2.1.
Return the linestyle: one of ('solid', 'dashed', 'dashdot', 'dotted').

`get_linewidth()`
Return the line width in points as a scalar

`get_rgb()`
returns a tuple of three or four floats from 0-1.

`get_sketch_params()`
Returns the sketch parameters for the artist.

 Returns

 sketch_params [tuple or `None`]

 A 3-tuple with the following elements:

- scale: The amplitude of the wiggle perpendicular to the source line.
- length: The length of the wiggle along the line.
- randomness: The scale factor by which the length is shrunk or expanded.

 May return 'None' if no sketch parameters were set.

`get_snap()`
returns the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is

- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

`get_url()`

returns a url if one is set, None otherwise

`restore()`

Restore the graphics context from the stack - needed only for backends that save graphics contexts on a stack

`set_alpha(alpha)`

Set the alpha value used for blending - not supported on all backends. If `alpha=None` (the default), the alpha components of the foreground and fill colors will be used to set their respective transparencies (where applicable); otherwise, alpha will override them.

`set_antialiased(b)`

True if object should be drawn with antialiased rendering

`set_capstyle(cs)`

Set the capstyle as a string in ('butt', 'round', 'projecting')

`set_clip_path(path)`

Set the clip path and transformation. Path should be a `TransformedPath` instance.

`set_clip_rectangle(rectangle)`

Set the clip rectangle with sequence (left, bottom, width, height)

`set_dashes(dash_offset, dash_list)`

Set the dash style for the gc.

Parameters

`dash_offset` [float] is the offset (usually 0).

`dash_list` [array_like] specifies the on-off sequence as points. (None, None) specifies a solid line

`set_foreground(fg, isRGBA=False)`

Set the foreground color. `fg` can be a MATLAB format string, a html hex color string, an rgb or rgba unit tuple, or a float between 0 and 1. In the latter case, grayscale is used.

If you know `fg` is rgba, set `isRGBA=True` for efficiency.

`set_gid(id)`

Sets the id.

`set_hatch(hatch)`

Sets the hatch style for filling

`set_hatch_color(hatch_color)`

sets the color to use for hatching.

`set_joinstyle(js)`

Set the join style to be one of ('miter', 'round', 'bevel')

`set_linestyle(style)`

Deprecated since version 2.1: The `set_linestyle` function was deprecated in version 2.1.

Set the linestyle to be one of ('solid', 'dashed', 'dashdot', 'dotted'). These are defined in the rcParams `lines.dashed_pattern`, `lines.dashdot_pattern` and `lines.dotted_pattern`. One may also specify customized dash styles by providing a tuple of (offset, dash pairs).

`set_linewidth(w)`

Set the linewidth in points

`set_sketch_params(scale=None, length=None, randomness=None)`

Sets the sketch parameters.

Parameters

`scale` [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If `scale` is `None`, or not provided, no sketch filter will be provided.

`length` [float, optional] The length of the wiggle along the line, in pixels (default 128)

`randomness` [float, optional] The scale factor by which the length is shrunk or expanded (default 16)

`set_snap(snap)`

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

`set_url(url)`

Sets the url for links in compatible backends

`class matplotlib.backend_bases.IdleEvent(**kwargs)`

Bases: `matplotlib.backend_bases.Event`

Deprecated since version 2.1: The `IdleEvent` class was deprecated in version 2.1.

An event triggered by the GUI backend when it is idle – useful for passive animation

`class matplotlib.backend_bases.KeyEvent(name, canvas, key, x=0, y=0, guiEvent=None)`

Bases: `matplotlib.backend_bases.LocationEvent`

A key event (key press, key release).

Attach additional attributes as defined in `FigureCanvasBase.mpl_connect()`.

In addition to the `Event` and `LocationEvent` attributes, the following attributes are defined:

Notes

Modifier keys will be prefixed to the pressed key and will be in the order "ctrl", "alt", "super". The exception to this rule is when the pressed key is itself a modifier key, therefore "ctrl+alt" and "alt+control" can both be valid key values.

Examples

Usage:

```
def on_key(event):
    print('you pressed', event.key, event.xdata, event.ydata)

cid = fig.canvas.mpl_connect('key_press_event', on_key)
```

Attributes

key [None or str] the key(s) pressed. Could be None, a single case sensitive ascii character ("g", "G", "#", etc.), a special key ("control", "shift", "f1", "up", etc.) or a combination of the above (e.g., "ctrl+alt+g", "ctrl+alt+G").

class matplotlib.backend_bases.LocationEvent(name, canvas, x, y, guiEvent=None)

Bases: matplotlib.backend_bases.Event

An event that has a screen location

The following additional attributes are defined and shown with their default values.

In addition to the Event attributes, the following event attributes are defined:

Attributes

x [scalar] x position - pixels from left of canvas
y [scalar] y position - pixels from bottom of canvas
inaxes [bool] the Axes instance if mouse is over axes
xdata [scalar] x coord of mouse in data coords
ydata [scalar] y coord of mouse in data coords

x, y in figure coords, 0,0 = bottom, left

inaxes = None

lastevent = None

x = None

xdata = None

y = None

ydata = None

```
class matplotlib.backend_bases.MouseEvent(name, canvas, x, y, button=None, key=None,
                                           step=0, dblclick=False, guiEvent=None)
    Bases: matplotlib.backend_bases.LocationEvent
```

A mouse event ('button_press_event', 'button_release_event', 'scroll_event', 'motion_notify_event').

In addition to the Event and LocationEvent attributes, the following attributes are defined:

Examples

Usage:

```
def on_press(event):
    print('you pressed', event.button, event.xdata, event.ydata)

cid = fig.canvas.mpl_connect('button_press_event', on_press)
```

Attributes

button [None, scalar, or str] button pressed None, 1, 2, 3, 'up', 'down' (up and down are used for scroll events). Note that in the nbagg backend, both the middle and right clicks return 3 since right clicking will bring up the context menu in some browsers.

key [None, or str] the key depressed when the mouse event triggered (see KeyEvent)

step [scalar] number of scroll steps (positive for 'up', negative for 'down')

x, y in figure coords, 0,0 = bottom, left button pressed None, 1, 2, 3, 'up', 'down'

button = None

dblclick = None

inaxes = None

step = None

`x = None`

`xdata = None`

`y = None`

`ydata = None`

`class matplotlib.backend_bases.NavigationToolbar2(canvas)`

Bases: `object`

Base class for the navigation cursor, version 2

backends must implement a canvas that handles connections for 'button_press_event' and 'button_release_event'. See `FigureCanvasBase.mpl_connect()` for more information

They must also define

`save_figure()` save the current figure

`set_cursor()` if you want the pointer icon to change

`_init_toolbar()` create your toolbar widget

`draw_rubberband()` (optional) draw the zoom to rect "rubberband" rectangle

`press()` (optional) whenever a mouse button is pressed, you'll be notified with the event

`release()` (optional) whenever a mouse button is released, you'll be notified with the event

`set_message()` (optional) display message

`set_history_buttons()` (optional) you can change the history back / forward buttons to indicate disabled / enabled state.

That's it, we'll do the rest!

`back(*args)`
move back up the view lim stack

`drag_pan(event)`
Callback for dragging in pan/zoom mode.

`drag_zoom(event)`
Callback for dragging in zoom mode.

`draw()`
Redraw the canvases, update the locators.

`draw_rubberband(event, x0, y0, x1, y1)`
Draw a rectangle rubberband to indicate zoom limits.

Note that it is not guaranteed that $x0 \leq x1$ and $y0 \leq y1$.

`dynamic_update()`
Deprecated since version 2.1: The `dynamic_update` function was deprecated in version 2.1. Use `canvas.draw_idle` instead.

`forward(*args)`
Move forward in the view lim stack.

`home(*args)`
Restore the original view.

`mouse__move(event)`

`pan(*args)`
Activate the pan/zoom tool. pan with left button, zoom with right

`press(event)`
Called whenever a mouse button is pressed.

`press__pan(event)`
Callback for mouse button press in pan/zoom mode.

`press__zoom(event)`
Callback for mouse button press in zoom to rect mode.

`push_current()`
Push the current view limits and position onto the stack.

`release(event)`
Callback for mouse button release.

`release__pan(event)`
Callback for mouse button release in pan/zoom mode.

`release__zoom(event)`
Callback for mouse button release in zoom to rect mode.

`remove_rubberband()`
Remove the rubberband.

`save_figure(*args)`
Save the current figure.

`set_cursor(cursor)`
Set the current cursor to one of the `Cursors` enums values.

If required by the backend, this method should trigger an update in the backend event loop after the cursor is set, as this method may be called e.g. before a long-running task during which the GUI is not updated.

`set_history_buttons()`
Enable or disable the back/forward button.

`set_message(s)`
Display a message on toolbar or in status bar.

```
toolitems = (('Home', 'Reset original view', 'home', 'home'), ('Back', 'Back to previous view', 'back', 'back'))
```

```
update()
```

Reset the axes stack.

```
zoom(*args)
```

Activate zoom to rect mode.

```
exception matplotlib.backend_bases.NonGuiException
```

Bases: [Exception](#)

```
class matplotlib.backend_bases.PickEvent(name, canvas, mouseevent, artist,
                                         guiEvent=None, **kwargs)
```

Bases: [matplotlib.backend_bases.Event](#)

a pick event, fired when the user picks a location on the canvas sufficiently close to an artist.

Attrs: all the Event attributes plus

Examples

Usage:

```
ax.plot(np.rand(100), 'o', picker=5) # 5 points tolerance
```

```
def on_pick(event):
```

```
    line = event.artist
```

```
    xdata, ydata = line.get_data()
```

```
    ind = event.ind
```

```
    print('on pick line:', np.array([xdata[ind], ydata[ind]]).T)
```

```
cid = fig.canvas.mpl_connect('pick_event', on_pick)
```

Attributes

mouseevent [[MouseEvent](#)] the mouse event that generated the pick

artist [[matplotlib.artist.Artist](#)] the picked artist

other extra class dependent attrs – e.g., a [Line2D](#) pick may define different extra attributes than a [PatchCollection](#) pick event

```
class matplotlib.backend_bases.RendererBase
```

Bases: [object](#)

An abstract base class to handle drawing/rendering operations.

The following methods must be implemented in the backend for full functionality (though just implementing `draw_path()` alone would give a highly capable backend):

- `draw_path()`

- `draw_image()`
- `draw_gouraud_triangle()`

The following methods should be implemented in the backend for optimization reasons:

- `draw_text()`
- `draw_markers()`
- `draw_path_collection()`
- `draw_quad_mesh()`

`close_group(s)`

Close a grouping element with label `s` Is only currently used by `backend_svg`

`draw_gouraud_triangle(gc, points, colors, transform)`

Draw a Gouraud-shaded triangle.

Parameters

`points` [array_like, shape=(3, 2)] Array of (x, y) points for the triangle.

`colors` [array_like, shape=(3, 4)] RGBA colors for each point of the triangle.

`transform` [matplotlib.transforms.Transform] An affine transform to apply to the points.

`draw_gouraud_triangles(gc, triangles_array, colors_array, transform)`

Draws a series of Gouraud triangles.

Parameters

`points` [array_like, shape=(N, 3, 2)] Array of N (x, y) points for the triangles.

`colors` [array_like, shape=(N, 3, 4)] Array of N RGBA colors for each point of the triangles.

`transform` [matplotlib.transforms.Transform] An affine transform to apply to the points.

`draw_image(gc, x, y, im, transform=None)`

Draw an RGBA image.

Parameters

`gc` [GraphicsContextBase] a graphics context with clipping information.

`x` [scalar] the distance in physical units (i.e., dots or pixels) from the left hand side of the canvas.

`y` [scalar] the distance in physical units (i.e., dots or pixels) from the bottom side of the canvas.

`im` [array_like, shape=(N, M, 4), dtype=np.uint8] An array of RGBA pixels.

`transform` [matplotlib.transforms.Affine2DBase] If and only if the concrete backend is written such that `option_scale_image()` returns True, an affine transformation may be passed to `draw_image()`. It takes the form of a `Affine2DBase` instance. The translation vector of the transformation is given in physical units (i.e., dots or pixels). Note that the transformation does not override `x` and `y`, and has to be applied before translating the result by `x` and `y` (this can be accomplished by adding `x` and `y` to the translation vector defined by `transform`).

`draw_markers(gc, marker_path, marker_trans, path, trans, rgbFace=None)`

Draws a marker at each of the vertices in `path`. This includes all vertices, including control points on curves. To avoid that behavior, those vertices should be removed before calling this function.

This provides a fallback implementation of `draw_markers` that makes multiple calls to `draw_path()`. Some backends may want to override this method in order to draw the marker only once and reuse it multiple times.

Parameters

`gc` [GraphicsContextBase] The graphics context

`marker_trans` [matplotlib.transforms.Transform] An affine transform applied to the marker.

`trans` [matplotlib.transforms.Transform] An affine transform applied to the path.

`draw_path(gc, path, transform, rgbFace=None)`

Draws a `Path` instance using the given affine transform.

`draw_path_collection(gc, master_transform, paths, all_transforms, offsets, offsetTrans, facecolors, edgecolors, linewidths, linestyle, antialiaseds, urls, offset_position)`

Draws a collection of paths selecting drawing properties from the lists `facecolors`, `edgecolors`, `linewidths`, `linestyle` and `antialiaseds`. `offsets` is a list of offsets to apply to each of the paths. The offsets in `offsets` are first transformed by `offsetTrans` before being applied. `offset_position` may be either "screen" or "data" depending on the space that the offsets are in.

This provides a fallback implementation of `draw_path_collection()` that makes multiple calls to `draw_path()`. Some backends may want to override this in order to render each set of path data only once, and then reference that path multiple times with the different offsets, colors, styles etc. The generator methods `_iter_collection_raw_paths()` and `_iter_collection()` are provided to help with (and standardize) the implementation across backends. It is highly recommended to use those generators, so that changes to the behavior of `draw_path_collection()` can be made globally.

`draw_quad_mesh(gc, master_transform, meshWidth, meshHeight, coordinates, offsets, offsetTrans, facecolors, antialiased, edgecolors)`

This provides a fallback implementation of `draw_quad_mesh()` that generates paths and then calls `draw_path_collection()`.

`draw_tex(gc, x, y, s, prop, angle, ismath='TeX!', mtext=None)`

`draw_text(gc, x, y, s, prop, angle, ismath=False, mtext=None)`

Draw the text instance

Parameters

`gc` [GraphicsContextBase] the graphics context

`x` [scalar] the x location of the text in display coords

`y` [scalar] the y location of the text baseline in display coords

`s` [str] the text string

`prop` [matplotlib.font_manager.FontProperties] font properties

`angle` [scalar] the rotation angle in degrees

`mtext` [matplotlib.text.Text] the original text object to be rendered

Notes

backend implementers note

When you are trying to determine if you have gotten your bounding box right (which is what enables the text layout/alignment to work properly), it helps to change the line in `text.py`:

`if 0: bbox_artist(self, renderer)`

to if 1, and then the actual bounding box will be plotted along with your text.

`flipy()`

Return true if y small numbers are top for renderer Is used for drawing text (`matplotlib.text`) and images (`matplotlib.image`) only

`get_canvas_width_height()`

return the canvas width and height in display coords

`get_image_magnification()`

Get the factor by which to magnify images passed to `draw_image()`. Allows a backend to have images at a different resolution to other artists.

`get_texmanager()`

return the `matplotlib.texmanager.TextManager` instance

`get_text_width_height_descent(s, prop, ismath)`

Get the width, height, and descent (offset from the bottom to the baseline), in display coords, of the string `s` with `FontProperties` prop

`new_gc()`

Return an instance of a `GraphicsContextBase`

`open_group(s, gid=None)`

Open a grouping element with label `s`. If `gid` is given, use `gid` as the id of the group. Is only currently used by `backend_svg`.

`option_image_nocomposite()`

override this method for renderers that do not necessarily always want to rescale and composite raster images. (like SVG, PDF, or PS)

`option_scale_image()`

override this method for renderers that support arbitrary affine transformations in `draw_image()` (most vector backends).

`points_to_pixels(points)`

Convert points to display units

You need to override this function (unless your backend doesn't have a `dpi`, e.g., `postscript` or `svg`). Some imaging systems assume some value for pixels per inch:

$$\text{points to pixels} = \text{points} * \text{pixels_per_inch} / 72.0 * \text{dpi} / 72.0$$

Parameters

`points` [scalar or array_like] a float or a numpy array of float

Returns

Points converted to pixels

`start_filter()`

Used in `AggRenderer`. Switch to a temporary renderer for image filtering effects.

`start_rasterizing()`

Used in `MixedModeRenderer`. Switch to the raster renderer.

`stop_filter(filter_func)`

Used in `AggRenderer`. Switch back to the original renderer. The contents of the temporary renderer is processed with the `filter_func` and is drawn on the original renderer as an image.

`stop_rasterizing()`

Used in `MixedModeRenderer`. Switch back to the vector renderer and draw the contents of the raster renderer as an image on the vector renderer.

`strip_math(s)`

`class matplotlib.backend_bases.ResizeEvent(name, canvas)`

Bases: `matplotlib.backend_bases.Event`

An event triggered by a canvas resize

In addition to the Event attributes, the following event attributes are defined:

Attributes

width [scalar] width of the canvas in pixels

height [scalar] height of the canvas in pixels

class matplotlib.backend_bases.ShowBase

Bases: matplotlib.backend_bases._Backend

Simple base class to generate a show() callable in backends.

Subclass must override mainloop() method.

class matplotlib.backend_bases.StatusbarBase(toolmanager)

Bases: object

Base class for the statusbar

set_message(s)

Display a message on toolbar or in status bar

Parameters

s [str] Message text

class matplotlib.backend_bases.TimerBase(interval=None, callbacks=None)

Bases: object

A base class for providing timer events, useful for things animations. Backends need to implement a few specific methods in order to use their own timing mechanisms so that the timer events are integrated into their event loops.

Mandatory functions that must be implemented:

- `_timer_start`: Contains backend-specific code for starting the timer
- `_timer_stop`: Contains backend-specific code for stopping the timer

Optional overrides:

- `_timer_set_single_shot`: Code for setting the timer to single shot operating mode, if supported by the timer object. If not, the Timer class itself will store the flag and the `_on_timer` method should be overridden to support such behavior.
- `_timer_set_interval`: Code for setting the interval on the timer, if there is a method for doing so on the timer object.
- `_on_timer`: This is the internal function that any timer object should call, which will handle the task of running all callbacks that have been set.

Attributes

interval [scalar] The time between timer events in milliseconds. Default is 1000 ms.

single_shot [bool] Boolean flag indicating whether this timer should operate as single shot (run once and then stop). Defaults to `False`.

`callbacks` [List[Tuple[callable, Tuple, Dict]]] Stores list of (func, args, kwargs) tuples that will be called upon timer events. This list can be manipulated directly, or the functions `add_callback` and `remove_callback` can be used.

`add_callback(func, *args, **kwargs)`

Register func to be called by timer when the event fires. Any additional arguments provided will be passed to func.

`interval`

`remove_callback(func, *args, **kwargs)`

Remove func from list of callbacks. args and kwargs are optional and used to distinguish between copies of the same function registered to be called with different arguments.

`single_shot`

`start(interval=None)`

Start the timer object. interval is optional and will be used to reset the timer interval first if provided.

`stop()`

Stop the timer.

class `matplotlib.backend_bases.ToolContainerBase(toolmanager)`

Bases: `object`

Base class for all tool containers, e.g. toolbars.

Attributes

`toolmanager` [ToolManager] The tools with which this ToolContainer wants to communicate.

`add_tool(tool, group, position=-1)`

Adds a tool to this container

Parameters

`tool` [tool_like] The tool to add, see `ToolManager.get_tool`.

`group` [str] The name of the group to add this tool to.

`position` [int (optional)] The position within the group to place this tool. Defaults to end.

`add_toolitem(name, group, position, image, description, toggle)`

Add a toolitem to the container

This method must get implemented per backend

The callback associated with the button click event, must be EXACTLY self.
`trigger_tool(name)`

Parameters

name [string] Name of the tool to add, this gets used as the tool's ID and as the default label of the buttons

group [String] Name of the group that this tool belongs to

position [Int] Position of the tool within its group, if -1 it goes at the End

image_file [String] Filename of the image for the button or `None`

description [String] Description of the tool, used for the tooltips

toggle [Bool]

- `True` : The button is a toggle (change the pressed/unpressed state between consecutive clicks)
- `False` : The button is a normal button (returns to unpressed state after release)

`remove_toolitem(name)`

Remove a toolitem from the ToolContainer

This method must get implemented per backend

Called when ToolManager emits a `tool_removed_event`

Parameters

name [string] Name of the tool to remove

`toggle_toolitem(name, toggled)`

Toggle the toolitem without firing event

Parameters

name [String] Id of the tool to toggle

toggled [bool] Whether to set this tool as toggled or not.

`trigger_tool(name)`

Trigger the tool

Parameters

name [String] Name (id) of the tool triggered from within the container

`matplotlib.backend_bases.get_registered_canvas_class(format)`

Return the registered default canvas for given file format. Handles deferred import of required backend.

`matplotlib.backend_bases.key_press_handler(event, canvas, toolbar=None)`

Implement the default mpl key bindings for the canvas and toolbar described at [Navigation Keyboard Shortcuts](#)

Parameters

event [KeyEvent] a key press/release event

canvas [FigureCanvasBase] the backend-specific canvas instance

toolbar [NavigationToolbar2] the navigation cursor toolbar

matplotlib.backend_bases.register_backend(format, backend, description=None)

Register a backend for saving to a given file format.

Parameters

format [str] File extension

backend [module string or canvas class] Backend for handling file output

description [str, optional] Description of the file type. Defaults to an empty string

37.2 matplotlib.backend_managers

ToolManager Class that makes the bridge between user interaction (key press, toolbar clicks, ..) and the actions in response to the user inputs.

class matplotlib.backend_managers.ToolEvent(name, sender, tool, data=None)

Bases: `object`

Event for tool manipulation (add/remove)

class matplotlib.backend_managers.ToolManager(figure=None)

Bases: `object`

Helper class that groups all the user interactions for a Figure

Attributes

figure: 'Figure'

keypresslock: 'widgets.LockDraw' LockDraw object to know if the canvas key_press_event is locked

messagelock: 'widgets.LockDraw' LockDraw object to know if the message is available to write

active_toggle

Currently toggled tools

add_tool(name, tool, *args, **kwargs)

Add tool to ToolManager

If successful adds a new event tool_trigger_name where name is the name of the tool, this event is fired everytime the tool is triggered.

Parameters

name [str] Name of the tool, treated as the ID, has to be unique

tool [class_like, i.e. str or type] Reference to find the class of the Tool to added.

See also:

`matplotlib.backend_tools.ToolBase` The base class for tools.

Notes

`args` and `kwargs` get passed directly to the tools constructor.

`canvas`

Canvas managed by `FigureManager`

`figure`

Figure that holds the canvas

`get_tool(name, warn=True)`

Return the tool object, also accepts the actual tool for convenience

Parameters

`name` [`str`, `ToolBase`] Name of the tool, or the tool itself

`warn` [`bool`, optional] If this method should give warnings.

`get_tool_keymap(name)`

Get the keymap associated with the specified tool

Parameters

`name` [`string`] Name of the Tool

Returns

`list` [list of keys associated with the Tool]

`message_event(message, sender=None)`

Emit a `ToolManagerMessageEvent`

`remove_tool(name)`

Remove tool from `ToolManager`

Parameters

`name` [`string`] Name of the Tool

`set_figure(figure, update_tools=True)`

Sets the figure to interact with the tools

Parameters

`figure`: 'Figure'

`update_tools`: `bool` Force tools to update figure

`toolmanager_connect(s, func)`

Connect event with string `s` to `func`.

Parameters

s [String] Name of the event

The following events are recognized

- 'tool_message_event'
- 'tool_removed_event'
- 'tool_added_event'

For every tool added a new event is created

- 'tool_trigger_TOOLNAME' Where TOOLNAME is the id of the tool.

func [function] Function to be called with signature def func(event)

toolmanager_disconnect(cid)

Disconnect callback id cid

Example usage:

```
cid = toolmanager.toolmanager_connect('tool_trigger_zoom',
                                     on_press)
#...later
toolmanager.toolmanager_disconnect(cid)
```

tools

Return the tools controlled by ToolManager

trigger_tool(name, sender=None, canvasevent=None, data=None)

Trigger a tool and emit the tool_trigger_[name] event

Parameters

name [string] Name of the tool

sender: object Object that wishes to trigger the tool

canvasevent [Event] Original Canvas event or None

data [Object] Extra data to pass to the tool when triggering

update_keymap(name, *keys)

Set the keymap to associate with the specified tool

Parameters

name [string] Name of the Tool

keys [keys to associate with the Tool]

class matplotlib.backend_managers.ToolManagerMessageEvent(name, sender, message)

Bases: object

Event carrying messages from toolmanager

Messages usually get displayed to the user by the toolbar


```
class matplotlib.backend_managers.ToolTriggerEvent(name, sender, tool, canvasev-
                                                    ent=None, data=None)
    Bases: matplotlib.backend_managers.ToolEvent
    Event to inform that a tool has been triggered
```

37.3 matplotlib.backends.backend_mixed

```
class matplotlib.backends.backend_mixed.MixedModeRenderer(figure, width, height,
                                                            dpi, vector_renderer,
                                                            raster_renderer_class=None,
                                                            bbox_inches_restore=None)
    Bases: object
```

A helper class to implement a renderer that switches between vector and raster drawing. An example may be a PDF writer, where most things are drawn with PDF vector commands, but some very complex objects, such as quad meshes, are rasterised and then output as images.

Parameters

`figure` [matplotlib.figure.Figure] The figure instance.

`width` [scalar] The width of the canvas in logical units

`height` [scalar] The height of the canvas in logical units

`dpi` [scalar] The dpi of the canvas

`vector_renderer` [matplotlib.backend_bases.RendererBase] An instance of a subclass of `RendererBase` that will be used for the vector drawing.

`raster_renderer_class` [matplotlib.backend_bases.RendererBase] The renderer class to use for the raster drawing. If not provided, this will use the Agg backend (which is currently the only viable option anyway.)

`start_rasterizing()`

Enter "raster" mode. All subsequent drawing commands (until `stop_rasterizing` is called) will be drawn with the raster backend.

If `start_rasterizing` is called multiple times before `stop_rasterizing` is called, this method has no effect.

`stop_rasterizing()`

Exit "raster" mode. All of the drawing that was done since the last `start_rasterizing` command will be copied to the vector backend by calling `draw_image`.

If `stop_rasterizing` is called multiple times before `start_rasterizing` is called, this method has no effect.

37.4 matplotlib.backend_tools

Abstract base classes define the primitives for Tools. These tools are used by matplotlib.backend_managers.ToolManager

ToolBase Simple stateless tool

ToolToggleBase Tool that has two states, only one Toggle tool can be active at any given time for the same matplotlib.backend_managers.ToolManager

```
class matplotlib.backend_tools.AxisScaleBase(*args, **kwargs)
```

Bases: matplotlib.backend_tools.ToolToggleBase

Base Tool to toggle between linear and logarithmic

disable(event)

Disable the toggle tool

trigger call this method when toggled is True.

This can happen in different circumstances

- Click on the toolbar tool button
- Call to matplotlib.backend_managers.ToolManager.trigger_tool
- Another ToolToggleBase derived tool is triggered (from the same ToolManager)

enable(event)

Enable the toggle tool

trigger calls this method when toggled is False

trigger(sender, event, data=None)

Calls enable or disable based on toggled value

```
class matplotlib.backend_tools.ConfigureSubplotsBase(toolmanager, name)
```

Bases: matplotlib.backend_tools.ToolBase

Base tool for the configuration of subplots

description = 'Configure subplots'

image = 'subplots'

```
class matplotlib.backend_tools.Cursors
```

Bases: object

Simple namespace for cursor reference

HAND = 0

MOVE = 3

POINTER = 1

SELECT_REGION = 2

WAIT = 4

```
class matplotlib.backend_tools.RubberbandBase(toolmanager, name)
    Bases: matplotlib.backend_tools.ToolBase

    Draw and remove rubberband

    draw__rubberband(*data)
        Draw rubberband

        This method must get implemented per backend

    remove__rubberband()
        Remove rubberband

        This method should get implemented per backend

    trigger(sender, event, data)
        Call draw__rubberband or remove__rubberband based on data
```

```
class matplotlib.backend_tools.SaveFigureBase(toolmanager, name)
    Bases: matplotlib.backend_tools.ToolBase

    Base tool for figure saving

    default__keymap = ['s', 'ctrl+s']

    description = 'Save the figure'

    image = 'filesave'
```

```
class matplotlib.backend_tools.SetCursorBase(*args, **kwargs)
    Bases: matplotlib.backend_tools.ToolBase

    Change to the current cursor while inaxes

    This tool, keeps track of all ToolToggleBase derived tools, and calls set__cursor when a tool
    gets triggered

    set__cursor(cursor)
        Set the cursor

        This method has to be implemented per backend

    set__figure(figure)
        Assign a figure to the tool

        Parameters
```

figure: 'Figure'

class matplotlib.backend_tools.ToolBack(toolmanager, name)

Bases: matplotlib.backend_tools.ViewsPositionsBase

Move back up the view lim stack

default_keymap = ['left', 'c', 'backspace']

description = 'Back to previous view'

image = 'back'

class matplotlib.backend_tools.ToolBase(toolmanager, name)

Bases: object

Base tool class

A base tool, only implements trigger method or not method at all. The tool is instantiated by matplotlib.backend_managers.ToolManager

Attributes

toolmanager: 'matplotlib.backend_managers.ToolManager' ToolManager
that controls this Tool

figure: 'FigureCanvas' Figure instance that is affected by this Tool

name: String Used as Id of the tool, has to be unique among tools of the same
ToolManager

canvas

default_keymap = None

Keymap to associate with this tool

String: List of comma separated keys that will be used to call this tool when the keypress event of self.figure.canvas is emitted

description = None

Description of the Tool

String: If the Tool is included in the Toolbar this text is used as a Tooltip

destroy()

Destroy the tool

This method is called when the tool is removed by matplotlib.backend_managers.ToolManager.remove_tool

figure

image = None

Filename of the image

String: Filename of the image to use in the toolbar. If None, the name is used as a label in the toolbar button

name

Tool Id

set_figure(*figure*)

Assign a figure to the tool

Parameters

figure: 'Figure'

toolmanager

trigger(*sender*, *event*, *data*=None)

Called when this tool gets used

This method is called by `matplotlib.backend_managers.ToolManager.trigger_tool`

Parameters

event: 'Event' The Canvas event that caused this tool to be called

sender: object Object that requested the tool to be triggered

data: object Extra data

`class matplotlib.backend_tools.ToolCursorPosition(*args, **kwargs)`

Bases: `matplotlib.backend_tools.ToolBase`

Send message with the current pointer position

This tool runs in the background reporting the position of the cursor

send_message(*event*)

Call `matplotlib.backend_managers.ToolManager.message_event`

set_figure(*figure*)

Assign a figure to the tool

Parameters

figure: 'Figure'

`class matplotlib.backend_tools.ToolEnableAllNavigation(toolmanager, name)`

Bases: `matplotlib.backend_tools.ToolBase`

Tool to enable all axes for toolmanager interaction

default_keymap = ['a']

description = 'Enables all axes toolmanager'

trigger(sender, event, data=None)
Called when this tool gets used

This method is called by matplotlib.backend_managers.ToolManager.trigger_tool

Parameters

event: 'Event' The Canvas event that caused this tool to be called

sender: object Object that requested the tool to be triggered

data: object Extra data

class matplotlib.backend_tools.ToolEnableNavigation(toolmanager, name)

Bases: matplotlib.backend_tools.ToolBase

Tool to enable a specific axes for toolmanager interaction

default_keymap = (1, 2, 3, 4, 5, 6, 7, 8, 9)

description = 'Enables one axes toolmanager'

trigger(sender, event, data=None)
Called when this tool gets used

This method is called by matplotlib.backend_managers.ToolManager.trigger_tool

Parameters

event: 'Event' The Canvas event that caused this tool to be called

sender: object Object that requested the tool to be triggered

data: object Extra data

class matplotlib.backend_tools.ToolForward(toolmanager, name)

Bases: matplotlib.backend_tools.ViewsPositionsBase

Move forward in the view lim stack

default_keymap = ['right', 'v']

description = 'Forward to next view'

image = 'forward'

class matplotlib.backend_tools.ToolFullScreen(*args, **kwargs)

Bases: matplotlib.backend_tools.ToolToggleBase

Tool to toggle full screen

default_keymap = ['f', 'ctrl+f']

description = 'Toogle Fullscreen mode'

disable(event)

Disable the toggle tool

trigger call this method when toggled is True.

This can happen in different circumstances

- Click on the toolbar tool button
- Call to matplotlib.backend_managers.ToolManager.trigger_tool
- Another ToolToggleBase derived tool is triggered (from the same ToolManager)

enable(event)

Enable the toggle tool

trigger calls this method when toggled is False

class matplotlib.backend_tools.ToolGrid(toolmanager, name)

Bases: matplotlib.backend_tools._ToolGridBase

Tool to toggle the major grids of the figure

default_keymap = ['g']

description = 'Toogle major grids'

class matplotlib.backend_tools.ToolHome(toolmanager, name)

Bases: matplotlib.backend_tools.ViewsPositionsBase

Restore the original view lim

default_keymap = ['h', 'r', 'home']

description = 'Reset original view'

image = 'home'

class matplotlib.backend_tools.ToolMinorGrid(toolmanager, name)

Bases: matplotlib.backend_tools._ToolGridBase

Tool to toggle the major and minor grids of the figure

default_keymap = ['G']

description = 'Toogle major and minor grids'

class matplotlib.backend_tools.ToolPan(*args)

Bases: matplotlib.backend_tools.ZoomPanBase

Pan axes with left mouse, zoom with right

cursor = 3

default_keymap = ['p']

description = 'Pan axes with left mouse, zoom with right'

image = 'move'

radio_group = 'default'

class matplotlib.backend_tools.ToolQuit(toolmanager, name)

Bases: matplotlib.backend_tools.ToolBase

Tool to call the figure manager destroy method

default_keymap = ['ctrl+w', 'cmd+w', 'q']

description = 'Quit the figure'

trigger(sender, event, data=None)

Called when this tool gets used

This method is called by matplotlib.backend_managers.ToolManager.trigger_tool

Parameters

event: 'Event' The Canvas event that caused this tool to be called

sender: object Object that requested the tool to be triggered

data: object Extra data

class matplotlib.backend_tools.ToolQuitAll(toolmanager, name)

Bases: matplotlib.backend_tools.ToolBase

Tool to call the figure manager destroy method

default_keymap = ['W', 'cmd+W', 'Q']

description = 'Quit all figures'

trigger(sender, event, data=None)

Called when this tool gets used

This method is called by matplotlib.backend_managers.ToolManager.trigger_tool

Parameters

event: 'Event' The Canvas event that caused this tool to be called

sender: object Object that requested the tool to be triggered

data: object Extra data

class matplotlib.backend_tools.ToolToggleBase(*args, **kwargs)

Bases: matplotlib.backend_tools.ToolBase

Toggleable tool

Every time it is triggered, it switches between enable and disable

Parameters

“*args“ Variable length argument to be used by the Tool

“**kwargs“ toggled if present and True, sets the initial state of the Tool
Arbitrary keyword arguments to be consumed by the Tool

cursor = None

Cursor to use when the tool is active

default_toggled = False

Default of toggled state

disable(event=None)

Disable the toggle tool

trigger call this method when toggled is True.

This can happen in different circumstances

- Click on the toolbar tool button
- Call to matplotlib.backend_managers.ToolManager.trigger_tool
- Another ToolToggleBase derived tool is triggered (from the same ToolManager)

enable(event=None)

Enable the toggle tool

trigger calls this method when toggled is False

radio_group = None

Attribute to group 'radio' like tools (mutually exclusive)

String that identifies the group or None if not belonging to a group

set_figure(figure)

Assign a figure to the tool

Parameters

figure: 'Figure'

toggled

State of the toggled tool

`trigger(sender, event, data=None)`
 Calls enable or disable based on toggled value

`class matplotlib.backend_tools.ToolViewsPositions(*args, **kwargs)`
 Bases: `matplotlib.backend_tools.ToolBase`

Auxiliary Tool to handle changes in views and positions

Runs in the background and should get used by all the tools that need to access the figure's history of views and positions, e.g.

- `ToolZoom`
- `ToolPan`
- `ToolHome`
- `ToolBack`
- `ToolForward`

`add_figure(figure)`
 Add the current figure to the stack of views and positions

`back()`
 Back one step in the stack of views and positions

`clear(figure)`
 Reset the axes stack

`forward()`
 Forward one step in the stack of views and positions

`home()`
 Recall the first view and position from the stack

`push_current(figure=None)`
 Push the current view limits and position onto their respective stacks

`refresh_locators()`
 Redraw the canvases, update the locators

`update_home_views(figure=None)`
 Make sure that `self.home_views` has an entry for all axes present in the figure

`update_view()`
 Update the view limits and position for each axes from the current stack position. If any axes are present in the figure that aren't in the current stack position, use the home view limits for those axes and don't update any positions.

`class matplotlib.backend_tools.ToolXScale(*args, **kwargs)`
 Bases: `matplotlib.backend_tools.AxisScaleBase`

Tool to toggle between linear and logarithmic scales on the X axis

`default_keymap = ['k', 'L']`

```
description = 'Toogle Scale X axis'
```

```
set_scale(ax, scale)
```

```
class matplotlib.backend_tools.ToolYScale(*args, **kwargs)
```

```
    Bases: matplotlib.backend_tools.AxisScaleBase
```

```
    Tool to toggle between linear and logarithmic scales on the Y axis
```

```
    default_keymap = ['l']
```

```
description = 'Toogle Scale Y axis'
```

```
set_scale(ax, scale)
```

```
class matplotlib.backend_tools.ToolZoom(*args)
```

```
    Bases: matplotlib.backend_tools.ZoomPanBase
```

```
    Zoom to rectangle
```

```
    cursor = 2
```

```
    default_keymap = ['o']
```

```
description = 'Zoom to rectangle'
```

```
image = 'zoom_to_rect'
```

```
radio_group = 'default'
```

```
class matplotlib.backend_tools.ViewsPositionsBase(toolmanager, name)
```

```
    Bases: matplotlib.backend_tools.ToolBase
```

```
    Base class for ToolHome, ToolBack and ToolForward
```

```
    trigger(sender, event, data=None)
```

```
        Called when this tool gets used
```

```
        This method is called by matplotlib.backend_managers.ToolManager.trigger_tool
```

```
        Parameters
```

```
            event: 'Event' The Canvas event that caused this tool to be called
```

```
            sender: object Object that requested the tool to be triggered
```

```
            data: object Extra data
```

```
class matplotlib.backend_tools.ZoomPanBase(*args)
    Bases: matplotlib.backend_tools.ToolToggleBase

    Base class for ToolZoom and ToolPan

    disable(event)
        Release the canvas and disconnect press/release events

    enable(event)
        Connect press/release events and lock the canvas

    scroll_zoom(event)

    trigger(sender, event, data=None)
        Calls enable or disable based on toggled value

matplotlib.backend_tools.add_tools_to_container(container, tools=[['navigation', ['home',
                                                                    'back', 'forward']], ['zoompan', ['pan',
                                                                    'zoom', 'subplots']], ['io', ['save']]])

Add multiple tools to the container.

Parameters
    container: Container backend_bases.ToolContainerBase object that will get
        the tools added
    tools [list, optional] List in the form [[group1, [tool1, tool2 ...]], [group2, [...]]]
        Where the tools given by tool1, and tool2 will display in group1. See
        add_tool for details.
```

```
matplotlib.backend_tools.add_tools_to_manager(toolmanager, tools={'allnav':
    <class 'matplotlib.backend_tools.ToolEnableAllNavigation'>,
    'back': <class 'matplotlib.backend_tools.ToolBack'>,
    'cursor': 'ToolSetCursor',
    'forward': <class 'matplotlib.backend_tools.ToolForward'>,
    'fullscreen': <class 'matplotlib.backend_tools.ToolFullScreen'>,
    'grid': <class 'matplotlib.backend_tools.ToolGrid'>,
    'grid_minor': <class 'matplotlib.backend_tools.ToolMinorGrid'>,
    'home': <class 'matplotlib.backend_tools.ToolHome'>,
    'nav': <class 'matplotlib.backend_tools.ToolEnableNavigation'>,
    'pan': <class 'matplotlib.backend_tools.ToolPan'>,
    'position': <class 'matplotlib.backend_tools.ToolCursorPosition'>,
    'quit': <class 'matplotlib.backend_tools.ToolQuit'>,
    'quit_all': <class 'matplotlib.backend_tools.ToolQuitAll'>,
    'rubberband': 'ToolRubberband', 'save': 'ToolSaveFigure',
    'subplots': 'ToolConfigureSubplots', 'viewpos': <class 'matplotlib.backend_tools.ToolViewsPositions'>,
    'xscale': <class 'matplotlib.backend_tools.ToolXScale'>,
    'yscale': <class 'matplotlib.backend_tools.ToolYScale'>,
    'zoom': <class 'matplotlib.backend_tools.ToolZoom'>})
```

Add multiple tools to ToolManager

Parameters

`toolmanager`: `ToolManager` backend_managers.ToolManager object that will get the tools added

`tools` [{str: class_like}, optional] The tools to add in a {name: tool} dict, see `add_tool` for more info.

```
matplotlib.backend_tools.default_toolbar_tools = [['navigation', ['home', 'back', 'forward']], ['zoompan', ['pan', 'zoom']]]
```

Default tools in the toolbar

```
matplotlib.backend_tools.default_tools = {'allnav': <class 'matplotlib.backend_tools.ToolEnableAllNavigation'>, ...}
```

Default tools

37.5 matplotlib.backends.backend_agg

An agg <http://antigrain.com/> backend

Features that are implemented

- capstyles and join styles
- dashes
- linewidth
- lines, rectangles, ellipses
- clipping to a rectangle
- output to RGBA and PNG, optionally JPEG and TIFF
- alpha blending
- DPI scaling properly - everything scales properly (dashes, linewidths, etc)
- draw polygon
- freetype2 w/ ft2font

TODO:

- integrate screen dpi w/ ppi and text

`matplotlib.backends.backend_agg.FigureCanvas`

alias of `matplotlib.backends.backend_agg.FigureCanvasAgg`

`class matplotlib.backends.backend_agg.FigureCanvasAgg(figure)`

Bases: `matplotlib.backend_bases.FigureCanvasBase`

The canvas the figure renders into. Calls the draw and print fig methods, creates the renderers, etc...

Attributes

`figure` [`matplotlib.figure.Figure`] A high-level Figure instance

`buffer_rgba()`

Get the image as an RGBA byte string

draw must be called at least once before this function will work and to update the renderer for any subsequent changes to the Figure.

Returns

bytes

`copy_from_bbox(bbox)`

`draw()`

Draw the figure using the renderer

`get_renderer(cleared=False)`

`print_jpeg(filename_or_obj, *args, **kwargs)`

Other Parameters

`quality` [int] The image quality, on a scale from 1 (worst) to 95 (best). The default is 95, if not given in the `matplotlibrc` file in the `savefig.jpeg_quality` parameter. Values above 95 should be avoided; 100 completely disables the JPEG quantization stage.

`optimize` [bool] If present, indicates that the encoder should make an extra pass over the image in order to select optimal encoder settings.

`progressive` [bool] If present, indicates that this image should be stored as a progressive JPEG file.

`print_jpg(filename_or_obj, *args, **kwargs)`

Other Parameters

`quality` [int] The image quality, on a scale from 1 (worst) to 95 (best). The default is 95, if not given in the `matplotlibrc` file in the `savefig.jpeg_quality` parameter. Values above 95 should be avoided; 100 completely disables the JPEG quantization stage.

`optimize` [bool] If present, indicates that the encoder should make an extra pass over the image in order to select optimal encoder settings.

`progressive` [bool] If present, indicates that this image should be stored as a progressive JPEG file.

`print_png(filename_or_obj, *args, **kwargs)`

`print_raw(filename_or_obj, *args, **kwargs)`

`print_rgba(filename_or_obj, *args, **kwargs)`

`print_tif(filename_or_obj, *args, **kwargs)`

`print_tiff(filename_or_obj, *args, **kwargs)`

`print_to_buffer()`

`restore_region(region, bbox=None, xy=None)`

`tostring_argb()`

Get the image as an ARGB byte string

`draw` must be called at least once before this function will work and to update the renderer for any subsequent changes to the Figure.

Returns

bytes

`tostring_rgb()`

Get the image as an RGB byte string

`draw` must be called at least once before this function will work and to update the renderer for any subsequent changes to the Figure.

Returns

bytes

`class matplotlib.backends.backend_agg.RendererAgg(width, height, dpi)`

Bases: `matplotlib.backend_bases.RendererBase`

The renderer handles all the drawing primitives using a graphics context instance that controls the colors/styles

`buffer_rgba()`

`clear()`

`debug`

Deprecated since version 2.2: The `debug` function was deprecated in version 2.2.

`draw_markers(*kl, **kw)`

Draws a marker at each of the vertices in `path`. This includes all vertices, including control points on curves. To avoid that behavior, those vertices should be removed before calling this function.

This provides a fallback implementation of `draw_markers` that makes multiple calls to `draw_path()`. Some backends may want to override this method in order to draw the marker only once and reuse it multiple times.

Parameters

`gc` [`GraphicsContextBase`] The graphics context

`marker_trans` [`matplotlib.transforms.Transform`] An affine transform applied to the marker.

`trans` [`matplotlib.transforms.Transform`] An affine transform applied to the path.

`draw_mathtext(gc, x, y, s, prop, angle)`

Draw the math text using `matplotlib.mathtext`

`draw_path(gc, path, transform, rgbFace=None)`

Draw the path

`draw_path_collection(*kl, **kw)`

Draws a collection of paths selecting drawing properties from the lists `facecolors`, `edgecolors`, `linewidths`, `linestyles` and `antialiaseds`. `offsets` is a list of offsets to apply to each of the paths. The offsets in `offsets` are first transformed by `offsetTrans` before being applied. `offset_position` may be either "screen" or "data" depending on the space that the offsets are in.

This provides a fallback implementation of `draw_path_collection()` that makes multiple calls to `draw_path()`. Some backends may want to override this in order to render each set of path data only once, and then reference that path multiple times with the different offsets, colors, styles etc. The generator methods `_iter_collection_raw_paths()` and `_iter_collection()` are provided to help with (and standardize) the implementation across backends. It is highly recommended to use those generators, so that changes to the behavior of `draw_path_collection()` can be made globally.

`draw_tex(gc, x, y, s, prop, angle, ismath='TeX!', mtext=None)`

`draw_text(gc, x, y, s, prop, angle, ismath=False, mtext=None)`

Render the text

`get_canvas_width_height()`

return the canvas width and height in display coords

`get_text_width_height_descent(s, prop, ismath)`

Get the width, height, and descent (offset from the bottom to the baseline), in display coords, of the string `s` with `FontProperties` prop

`lock = <unlocked _thread.RLock object owner=0 count=0>`

`option_image_nocomposite()`

override this method for renderers that do not necessarily always want to rescale and composite raster images. (like SVG, PDF, or PS)

`option_scale_image()`

agg backend doesn't support arbitrary scaling of image.

`points_to_pixels(points)`

convert point measures to pixels using dpi and the pixels per inch of the display

`restore_region(region, bbox=None, xy=None)`

Restore the saved region. If `bbox` (instance of `BboxBase`, or its extents) is given, only the region specified by the `bbox` will be restored. `xy` (a tuple of two floats) optionally specifies the new position (the LLC of the original region, not the LLC of the `bbox`) where the region will be restored.

```

>>> region = renderer.copy_from_bbox()
>>> x1, y1, x2, y2 = region.get_extents()
>>> renderer.restore_region(region, bbox=(x1+dx, y1, x2, y2),
...                                   xy=(x1-dx, y1))

```

`start_filter()`

Start filtering. It simply create a new canvas (the old one is saved).

`stop_filter(post_processing)`

Save the plot in the current canvas as a image and apply the `post_processing` function.

```

def post_processing(image, dpi): # ny, nx, depth = image.shape # image
    (numpy array) has RGBA channels and has a depth of 4. ... # create
    a new_image (numpy array of 4 channels, size can be # different). The
    resulting image may have offsets from # lower-left corner of the original
    image return new_image, offset_x, offset_y

```

The saved renderer is restored and the returned image from `post_processing` is plotted (using `draw_image`) on it.

`tostring_argb()`

`tostring_rgb()`

`tostring_rgba_minimized()`

`matplotlib.backends.backend_agg.get_hinting_flag()`

37.6 matplotlib.backends.backend_cairo

37.6.1 A Cairo backend for matplotlib

Author Steve Chaplin and others

This backend depends on [cairo](#), and either on `cairocffi`, or (Python 2 only) on `pycairo`.

`class matplotlib.backends.backend_cairo.ArrayWrapper(myarray)`

Bases: `object`

Thin wrapper around `numpy.ndarray` to expose the interface expected by `cairocffi`. Basically replicates the `array.array` interface.

`buffer_info()`

`matplotlib.backends.backend_cairo.FigureCanvas`

alias of `matplotlib.backends.backend_cairo.FigureCanvasCairo`

```
class matplotlib.backends.backend_cairo.FigureCanvasCairo(figure)
    Bases: matplotlib.backend_bases.FigureCanvasBase

    print_pdf(fobj, *args, **kwargs)

    print_png(fobj, *args, **kwargs)

    print_ps(fobj, *args, **kwargs)

    print_svg(fobj, *args, **kwargs)

    print_svgz(fobj, *args, **kwargs)

    supports_blit = False

class matplotlib.backends.backend_cairo.GraphicsContextCairo(renderer)
    Bases: matplotlib.backend_bases.GraphicsContextBase

    get_rgb()
        returns a tuple of three or four floats from 0-1.

    restore()
        Restore the graphics context from the stack - needed only for backends that save graphics
        contexts on a stack

    set_alpha(alpha)
        Set the alpha value used for blending - not supported on all backends. If alpha=None
        (the default), the alpha components of the foreground and fill colors will be used to set
        their respective transparencies (where applicable); otherwise, alpha will override them.

    set_capstyle(cs)
        Set the capstyle as a string in ('butt', 'round', 'projecting')

    set_clip_path(path)
        Set the clip path and transformation. Path should be a TransformedPath instance.

    set_clip_rectangle(rectangle)
        Set the clip rectangle with sequence (left, bottom, width, height)

    set_dashes(offset, dashes)
        Set the dash style for the gc.

        Parameters
            dash_offset [float] is the offset (usually 0).
            dash_list [array_like] specifies the on-off sequence as points. (None, None)
            specifies a solid line

    set_foreground(fg, isRGBA=None)
        Set the foreground color. fg can be a MATLAB format string, a html hex color string,
```

an rgb or rgba unit tuple, or a float between 0 and 1. In the latter case, grayscale is used.

If you know fg is rgba, set isRGBA=True for efficiency.

`set_joinstyle(js)`

Set the join style to be one of ('miter', 'round', 'bevel')

`set_linewidth(w)`

Set the linewidth in points

`class matplotlib.backends.backend_cairo.RendererCairo(dpi)`

Bases: `matplotlib.backend_bases.RendererBase`

`static convert_path(ctx, path, transform, clip=None)`

`draw_image(gc, x, y, im)`

Draw an RGBA image.

Parameters

`gc` [`GraphicsContextBase`] a graphics context with clipping information.

`x` [scalar] the distance in physical units (i.e., dots or pixels) from the left hand side of the canvas.

`y` [scalar] the distance in physical units (i.e., dots or pixels) from the bottom side of the canvas.

`im` [array_like, shape=(N, M, 4), dtype=np.uint8] An array of RGBA pixels.

`transform` [`matplotlib.transforms.Affine2DBase`] If and only if the concrete backend is written such that `option_scale_image()` returns True, an affine transformation may be passed to `draw_image()`. It takes the form of a `Affine2DBase` instance. The translation vector of the transformation is given in physical units (i.e., dots or pixels). Note that the transformation does not override x and y, and has to be applied before translating the result by x and y (this can be accomplished by adding x and y to the translation vector defined by transform).

`draw_markers(gc, marker_path, marker_trans, path, transform, rgbFace=None)`

Draws a marker at each of the vertices in path. This includes all vertices, including control points on curves. To avoid that behavior, those vertices should be removed before calling this function.

This provides a fallback implementation of `draw_markers` that makes multiple calls to `draw_path()`. Some backends may want to override this method in order to draw the marker only once and reuse it multiple times.

Parameters

`gc` [`GraphicsContextBase`] The graphics context

`marker_trans` [matplotlib.transforms.Transform] An affine transform applied to the marker.

`trans` [matplotlib.transforms.Transform] An affine transform applied to the path.

`draw_path(gc, path, transform, rgbFace=None)`

Draws a Path instance using the given affine transform.

`draw_text(gc, x, y, s, prop, angle, ismath=False, mtext=None)`

Draw the text instance

Parameters

`gc` [GraphicsContextBase] the graphics context

`x` [scalar] the x location of the text in display coords

`y` [scalar] the y location of the text baseline in display coords

`s` [str] the text string

`prop` [matplotlib.font_manager.FontProperties] font properties

`angle` [scalar] the rotation angle in degrees

`mtext` [matplotlib.text.Text] the original text object to be rendered

Notes

backend implementers note

When you are trying to determine if you have gotten your bounding box right (which is what enables the text layout/alignment to work properly), it helps to change the line in `text.py`:

```
if 0: bbox_artist(self, renderer)
```

to if 1, and then the actual bounding box will be plotted along with your text.

```
fontangles = {'italic': <MyCairoCffi name='mock.FONT_SLANT_ITALIC' id='140677916054864'>, 'normal': <MyCairoCffi name='mock.FONT_WEIGHT_NORMAL' id='140677916025744'>, 2
```

```
fontweights = {100: <MyCairoCffi name='mock.FONT_WEIGHT_NORMAL' id='140677916025744'>, 2
```

`get_canvas_width_height()`

return the canvas width and height in display coords

`get_text_width_height_descent(s, prop, ismath)`

Get the width, height, and descent (offset from the bottom to the baseline), in display coords, of the string `s` with FontProperties `prop`

`new_gc()`

Return an instance of a GraphicsContextBase

`points_to_pixels(points)`

Convert points to display units

You need to override this function (unless your backend doesn't have a `dpi`, e.g., `postscript` or `svg`). Some imaging systems assume some value for pixels per inch:

$$\text{points to pixels} = \text{points} * \text{pixels_per_inch} / 72.0 * \text{dpi} / 72.0$$

Parameters

`points` [scalar or array_like] a float or a numpy array of float

Returns

Points converted to pixels

`set_ctx_from_surface(surface)`

`set_width_height(width, height)`

37.7 matplotlib.backends.backend_gtkagg

TODO We'll add this later, importing the gtk backends requires an active X-session, which is not compatible with cron jobs.

37.8 matplotlib.backends.backend_gtkcairo

TODO We'll add this later, importing the gtk backends requires an active X-session, which is not compatible with cron jobs.

37.9 matplotlib.backends.backend_gtk3agg

TODO We'll add this later, importing the gtk3 backends requires an active X-session, which is not compatible with cron jobs.

37.10 matplotlib.backends.backend_gtk3cairo

TODO We'll add this later, importing the gtk3 backends requires an active X-session, which is not compatible with cron jobs.

37.11 matplotlib.backends.backend_nbagg

Interactive figures in the IPython notebook

class matplotlib.backends.backend_nbagg.CommSocket(manager)

Bases: `object`

Manages the Comm connection between IPython and the browser (client).

Comms are 2 way, with the CommSocket being able to publish a message via the `send_json` method, and handle a message with `on_message`. On the JS side `figure.send_message` and `figure.ws.onmessage` do the sending and receiving respectively.

`is_open()`

`on_close()`

`on_message(message)`

`send_binary(blob)`

`send_json(content)`

matplotlib.backends.backend_nbagg.FigureCanvas

alias of matplotlib.backends.backend_nbagg.FigureCanvasNbAgg

class matplotlib.backends.backend_nbagg.FigureCanvasNbAgg(*args, **kwargs)

Bases: matplotlib.backends.backend_webagg_core.FigureCanvasWebAggCore

`new_timer(*args, **kwargs)`

Creates a new backend-specific subclass of `backend_bases.Timer`. This is useful for getting periodic events through the backend's native event loop. Implemented only for backends with GUIs.

Other Parameters

`interval` [scalar] Timer interval in milliseconds

`callbacks` [List[Tuple[callable, Tuple, Dict]]] Sequence of (func, args, kwargs) where `func(*args, **kwargs)` will be executed by the timer every interval.

callbacks which return `False` or `0` will be removed from the timer.

Examples

```
>>> timer = fig.canvas.new_timer(callbacks=[(f1, (1, ), {'a': 3}),])
```

`matplotlib.backends.backend_nbagg.FigureManager`
alias of `matplotlib.backends.backend_nbagg.FigureManagerNbAgg`

class `matplotlib.backends.backend_nbagg.FigureManagerNbAgg(canvas, num)`
Bases: `matplotlib.backends.backend_webagg_core.FigureManagerWebAgg`

`ToolbarCls`
alias of `NavigationIPy`

`cleanup_closed()`
Clear up any closed Comms.

`connected`

`destroy()`

`display_js()`

classmethod `get_javascript(stream=None)`

`remove_comm(comm_id)`

`reshow()`
A special method to re-show the figure in the notebook.

`show()`
For GUI backends, show the figure window and redraw. For non-GUI backends, raise an exception to be caught by `show()`, for an optional warning.

class `matplotlib.backends.backend_nbagg.NavigationIPy(canvas)`
Bases: `matplotlib.backends.backend_webagg_core.NavigationToolbar2WebAgg`

`toolitems` = `[('Home', 'Reset original view', 'fa fa-home icon-home', 'home'), ('Back', 'Back to previous view', 'fa fa-undo icon-undo', 'back')]`

`matplotlib.backends.backend_nbagg.connection_info()`
Return a string showing the figure and connection status for the backend. This is intended as a diagnostic tool, and not for general use.

`matplotlib.backends.backend_nbagg.new_figure_manager_given_figure(num, figure)`
Create a new figure manager instance for the given figure.

`matplotlib.backends.backend_nbagg.show(*args, **kwargs)`
Show all figures.

`show` blocks by calling `mainloop` if `block` is `True`, or if it is `None` and we are neither in IPython's `%pylab` mode, nor in interactive mode.

37.12 matplotlib.backends.backend_pdf

A PDF matplotlib backend Author: Jouni K Seppänen <jks@iki.fi>

matplotlib.backends.backend_pdf.FigureCanvas

alias of matplotlib.backends.backend_pdf.FigureCanvasPdf

class matplotlib.backends.backend_pdf.FigureCanvasPdf(*figure*)

Bases: matplotlib.backend_bases.FigureCanvasBase

The canvas the figure renders into. Calls the draw and print fig methods, creates the renderers, etc...

Attributes

figure [matplotlib.figure.Figure] A high-level Figure instance

draw()

Render the Figure.

filetypes = {'pdf': 'Portable Document Format'}

fixed_dpi = 72

get_default_filetype()

Get the default savefig file format as specified in rcParam savefig.format. Returned string excludes period. Overridden in backends that only support a single file type.

print_pdf(filename, **kwargs)

matplotlib.backends.backend_pdf.FigureManager

alias of matplotlib.backends.backend_pdf.FigureManagerPdf

class matplotlib.backends.backend_pdf.FigureManagerPdf(*canvas*, *num*)

Bases: matplotlib.backend_bases.FigureManagerBase

class matplotlib.backends.backend_pdf.GraphicsContextPdf(*file*)

Bases: matplotlib.backend_bases.GraphicsContextBase

alpha_cmd(alpha, forced, effective_alphas)

capstyle_cmd(style)

capstyles = {'butt': 0, 'projecting': 2, 'round': 1}

clip_cmd(cliprect, clippath)

Set clip rectangle. Calls self.pop() and self.push().

commands = (((('_cliprect', '_clippath'), <function GraphicsContextPdf.clip_cmd>), (('alpha', '_forced',

`copy_properties(other)`

Copy properties of other into self.

`dash_cmd(dashes)`

`delta(other)`

Copy properties of other into self and return PDF commands needed to transform self into other.

`fill(*args)`

Predicate: does the path need to be filled?

An optional argument can be used to specify an alternative `_fillcolor`, as needed by `RendererPdf.draw_markers`.

`fillcolor_cmd(rgb)`

`finalize()`

Make sure every pushed graphics state is popped.

`hatch_cmd(hatch, hatch_color)`

`joinstyle_cmd(style)`

`joinstyles = {'bevel': 2, 'miter': 0, 'round': 1}`

`linewidth_cmd(width)`

`paint()`

Return the appropriate pdf operator to cause the path to be stroked, filled, or both.

`pop()`

`push()`

`rgb_cmd(rgb)`

`stroke()`

Predicate: does the path need to be stroked (its outline drawn)? This tests for the various conditions that disable stroking the path, in which case it would presumably be filled.

`class matplotlib.backends.backend_pdf.Name(name)`

Bases: `object`

PDF name object.

static hexify(match)

name

pdfRepr()

class matplotlib.backends.backend_pdf.Operator(op)

Bases: `object`

PDF operator object.

op

pdfRepr()

class matplotlib.backends.backend_pdf.PdfFile(filename, metadata=None)

Bases: `object`

PDF file object.

addGouraudTriangles(points, colors)

alphaState(alpha)

Return name of an ExtGState that sets alpha to the given value.

beginStream(id, len, extra=None, png=None)

close()

Flush all buffers and free all resources.

createType1Descriptor(t1font, fontfile)

dviFontName(dvifont)

Given a dvi font object, return a name suitable for `Op.selectfont`. This registers the font information in `self.dviFontInfo` if not yet registered.

embedTTF(filename, characters)

Embed the TTF font from the named file into the document.

endStream()

finalize()

Write out the various deferred objects and the pdf end matter.

fontName(fontprop)

Select a font based on fontprop and return a name suitable for `Op.selectfont`. If fontprop is a string, it will be interpreted as the filename of the font.

`hatchPattern(hatch_style)`

`imageObject(image)`

Return name of an image XObject representing the given image.

`markerObject(path, trans, fill, stroke, lw, joinstyle, capstyle)`

Return name of a marker XObject representing the given path.

`newPage(width, height)`

`newTextnote(text, positionRect=[-100, -100, 0, 0])`

`output(*data)`

`pathCollectionObject(gc, path, trans, padding, filled, stroked)`

`static pathOperations(path, transform, clip=None, simplify=None, sketch=None)`

`recordXref(id)`

`reserveObject(name=)`

Reserve an ID for an indirect object. The name is used for debugging in case we forget to print out the object with `writeObject`.

`texFontMap`

`write(data)`

`writeFonts()`

`writeGouraudTriangles()`

`writeHatches()`

`writeImages()`

`writeInfoDict()`

Write out the info dictionary, checking it for good form

`writeMarkers()`

`writeObject(object, contents)`

`writePath(path, transform, clip=False, sketch=None)`

`writePathCollectionTemplates()`

`writeTrailer()`

Write out the PDF trailer.

`writeXref()`

Write out the xref table.

```
class matplotlib.backends.backend_pdf.PdfPages(filename, keep_empty=True, meta-
                                              data=None)
```

Bases: `object`

A multi-page PDF file.

Notes

In reality PdfPages is a thin wrapper around PdfFile, in order to avoid confusion when using `savefig()` and forgetting the format argument.

Examples

```
>>> import matplotlib.pyplot as plt
>>> # Initialize:
>>> with PdfPages('foo.pdf') as pdf:
...     # As many times as you like, create a figure and save it:
...     fig = plt.figure()
...     pdf.savefig(fig)
...     # When no figure is specified the current figure is saved
...     pdf.savefig()
```

Create a new PdfPages object.

Parameters

`filename` [str] Plots using PdfPages.savefig() will be written to a file at this location. The file is opened at once and any older file with the same name is overwritten.

`keep_empty` [bool, optional] If set to False, then empty pdf files will be deleted automatically when closed.

`metadata` [dictionary, optional] Information dictionary object (see PDF reference section 10.2.1 'Document Information Dictionary'), e.g.: {'Creator': 'My software', 'Author': 'Me', 'Title': 'Awesome fig'}

The standard keys are 'Title', 'Author', 'Subject', 'Keywords', 'Creator', 'Producer', 'CreationDate', 'ModDate', and 'Trapped'. Values have been

predefined for 'Creator', 'Producer' and 'CreationDate'. They can be removed by setting them to `None`.

`attach_note(text, positionRect=[-100, -100, 0, 0])`

Add a new text note to the page to be saved next. The optional `positionRect` specifies the position of the new note on the page. It is outside the page per default to make sure it is invisible on printouts.

`close()`

Finalize this object, making the underlying file a complete PDF file.

`get_pagecount()`

Returns the current number of pages in the multipage pdf file.

`infodict()`

Return a modifiable information dictionary object (see PDF reference section 10.2.1 'Document Information Dictionary').

`keep_empty`

`savefig(figure=None, **kwargs)`

Saves a Figure to this file as a new page.

Any other keyword arguments are passed to `savefig()`.

Parameters

`figure` [Figure or int, optional] Specifies what figure is saved to file. If not specified, the active figure is saved. If a Figure instance is provided, this figure is saved. If an int is specified, the figure instance to save is looked up by number.

`class matplotlib.backends.backend_pdf.Reference(id)`

Bases: `object`

PDF reference object. Use `PdfFile.reserveObject()` to create References.

`pdfRepr()`

`write(contents, file)`

`class matplotlib.backends.backend_pdf.RendererPdf(file, image_dpi, height, width)`

Bases: `matplotlib.backends.backend_bases.RendererBase`

`afm_font_cache = {}`

`check_gc(gc, fillcolor=None)`

`draw_gouraud_triangle(gc, points, colors, trans)`

Draw a Gouraud-shaded triangle.

Parameters

`points` [array_like, shape=(3, 2)] Array of (x, y) points for the triangle.

`colors` [array_like, shape=(3, 4)] RGBA colors for each point of the triangle.

`transform` [matplotlib.transforms.Transform] An affine transform to apply to the points.

`draw_gouraud_triangles(gc, points, colors, trans)`

Draws a series of Gouraud triangles.

Parameters

`points` [array_like, shape=(N, 3, 2)] Array of N (x, y) points for the triangles.

`colors` [array_like, shape=(N, 3, 4)] Array of N RGBA colors for each point of the triangles.

`transform` [matplotlib.transforms.Transform] An affine transform to apply to the points.

`draw_image(gc, x, y, im, transform=None)`

Draw an RGBA image.

Parameters

`gc` [GraphicsContextBase] a graphics context with clipping information.

`x` [scalar] the distance in physical units (i.e., dots or pixels) from the left hand side of the canvas.

`y` [scalar] the distance in physical units (i.e., dots or pixels) from the bottom side of the canvas.

`im` [array_like, shape=(N, M, 4), dtype=np.uint8] An array of RGBA pixels.

`transform` [matplotlib.transforms.Affine2DBase] If and only if the concrete backend is written such that `option_scale_image()` returns True, an affine transformation may be passed to `draw_image()`. It takes the form of a `Affine2DBase` instance. The translation vector of the transformation is given in physical units (i.e., dots or pixels). Note that the transformation does not override `x` and `y`, and has to be applied before translating the result by `x` and `y` (this can be accomplished by adding `x` and `y` to the translation vector defined by `transform`).

`draw_markers(gc, marker_path, marker_trans, path, trans, rgbFace=None)`

Draws a marker at each of the vertices in `path`. This includes all vertices, including control points on curves. To avoid that behavior, those vertices should be removed before calling this function.

This provides a fallback implementation of `draw_markers` that makes multiple calls to `draw_path()`. Some backends may want to override this method in order to draw the marker only once and reuse it multiple times.

Parameters

- `gc` [GraphicsContextBase] The graphics context
- `marker_trans` [matplotlib.transforms.Transform] An affine transform applied to the marker.
- `trans` [matplotlib.transforms.Transform] An affine transform applied to the path.

`draw_mathtext(gc, x, y, s, prop, angle)`

`draw_path(gc, path, transform, rgbFace=None)`

Draws a Path instance using the given affine transform.

`draw_path_collection(gc, master_transform, paths, all_transforms, offsets, offsetTrans, facecolors, edgecolors, linewidths, linestyle, antialiaseds, urls, offset_position)`

Draws a collection of paths selecting drawing properties from the lists facecolors, edgecolors, linewidths, linestyle and antialiaseds. offsets is a list of offsets to apply to each of the paths. The offsets in offsets are first transformed by offsetTrans before being applied. offset_position may be either "screen" or "data" depending on the space that the offsets are in.

This provides a fallback implementation of `draw_path_collection()` that makes multiple calls to `draw_path()`. Some backends may want to override this in order to render each set of path data only once, and then reference that path multiple times with the different offsets, colors, styles etc. The generator methods `_iter_collection_raw_paths()` and `_iter_collection()` are provided to help with (and standardize) the implementation across backends. It is highly recommended to use those generators, so that changes to the behavior of `draw_path_collection()` can be made globally.

`draw_tex(gc, x, y, s, prop, angle, ismath='TeX!', mtext=None)`

`draw_text(gc, x, y, s, prop, angle, ismath=False, mtext=None)`

Draw the text instance

Parameters

- `gc` [GraphicsContextBase] the graphics context
- `x` [scalar] the x location of the text in display coords
- `y` [scalar] the y location of the text baseline in display coords
- `s` [str] the text string
- `prop` [matplotlib.font_manager.FontProperties] font properties
- `angle` [scalar] the rotation angle in degrees
- `mtext` [matplotlib.text.Text] the original text object to be rendered

Notes

backend implementers note

When you are trying to determine if you have gotten your bounding box right (which is what enables the text layout/alignment to work properly), it helps to change the line in `text.py`:

```
if 0: bbox_artist(self, renderer)
```

to if 1, and then the actual bounding box will be plotted along with your text.

`encode_string(s, fonttype)`

`finalize()`

`flipy()`

Return true if y small numbers are top for renderer Is used for drawing text (`matplotlib.text`) and images (`matplotlib.image`) only

`get_canvas_width_height()`

return the canvas width and height in display coords

`get_image_magnification()`

Get the factor by which to magnify images passed to `draw_image()`. Allows a backend to have images at a different resolution to other artists.

`get_text_width_height_descent(s, prop, ismath)`

Get the width, height, and descent (offset from the bottom to the baseline), in display coords, of the string `s` with `FontProperties` `prop`

`merge_used_characters(other)`

`new_gc()`

Return an instance of a `GraphicsContextBase`

`option_image_nocomposite()`

return whether to generate a composite image from multiple images on a set of axes

`option_scale_image()`

pdf backend support arbitrary scaling of image.

`track_characters(font, s)`

Keeps track of which characters are required from each font.

`class matplotlib.backends.backend_pdf.Stream(id, len, file, extra=None, png=None)`

Bases: `object`

PDF stream object.

This has no `pdfRepr` method. Instead, call `begin()`, then output the contents of the stream by calling `write()`, and finally call `end()`.

id: object id of stream; len: an unused Reference object for the length of the stream, or None (to use a memory buffer); file: a PdfFile; extra: a dictionary of extra key-value pairs to include in the stream header; png: if the data is already png compressed, the decode parameters

compressobj

end()

Finalize stream.

extra

file

id

len

pdfFile

pos

write(data)

Write some data on the stream.

class matplotlib.backends.backend_pdf.Verbatim(x)

Bases: [object](#)

Store verbatim PDF command content for later inclusion in the stream.

pdfRepr()

matplotlib.backends.backend_pdf.fill(strings, linelen=75)

Make one string from sequence of strings, with whitespace in between. The whitespace is chosen to form lines of at most linelen characters, if possible.

matplotlib.backends.backend_pdf.pdfRepr(obj)

Map Python objects to PDF syntax.

37.13 matplotlib.backends.backend_pgf

matplotlib.backends.backend_pgf.FigureCanvas

alias of matplotlib.backends.backend_pgf.FigureCanvasPgf

class matplotlib.backends.backend_pgf.FigureCanvasPgf(figure)

Bases: matplotlib.backend_bases.FigureCanvasBase

```
filetypes = {'pdf': 'LaTeX compiled PGF picture', 'pgf': 'LaTeX PGF picture', 'png': 'Portable Network G

get_default_filetype()
    Get the default savefig file format as specified in rcParam savefig.format. Returned
    string excludes period. Overridden in backends that only support a single file type.

get_renderer()

print_pdf(fname_or_fh, *args, **kwargs)
    Use LaTeX to compile a Pgf generated figure to PDF.

print_pgf(fname_or_fh, *args, **kwargs)
    Output pgf commands for drawing the figure so it can be included and rendered in
    latex documents.

print_png(fname_or_fh, *args, **kwargs)
    Use LaTeX to compile a pgf figure to pdf and convert it to png.

matplotlib.backends.backend_pgf.FigureManager
    alias of matplotlib.backends.backend_pgf.FigureManagerPgf

class matplotlib.backends.backend_pgf.FigureManagerPgf(*args)
    Bases: matplotlib.backend_bases.FigureManagerBase

class matplotlib.backends.backend_pgf.GraphicsContextPgf
    Bases: matplotlib.backend_bases.GraphicsContextBase

exception matplotlib.backends.backend_pgf.LatexError(message, latex_output=")
    Bases: Exception

class matplotlib.backends.backend_pgf.LatexManager
    Bases: object

    The LatexManager opens an instance of the LaTeX application for determining the metrics
    of text elements. The LaTeX environment can be modified by setting fonts and/or a custom
    preamble in the rc parameters.

    get_width_height_descent(text, prop)
        Get the width, total height and descent for a text typesetted by the current LaTeX
        environment.

class matplotlib.backends.backend_pgf.LatexManagerFactory
    Bases: object

    static get_latex_manager()

    previous_instance = None

class matplotlib.backends.backend_pgf.RendererPgf(figure, fh, dummy=False)
    Bases: matplotlib.backend_bases.RendererBase
```

Creates a new PGF renderer that translates any drawing instruction into text commands to be interpreted in a latex pgfpicture environment.

Attributes

`figure` [matplotlib.figure.Figure] Matplotlib figure to initialize height, width and dpi from.

`fh` [file-like] File handle for the output of the drawing commands.

`draw_image(gc, x, y, im, transform=None)`
Draw an RGBA image.

Parameters

`gc` [GraphicsContextBase] a graphics context with clipping information.

`x` [scalar] the distance in physical units (i.e., dots or pixels) from the left hand side of the canvas.

`y` [scalar] the distance in physical units (i.e., dots or pixels) from the bottom side of the canvas.

`im` [array_like, shape=(N, M, 4), dtype=np.uint8] An array of RGBA pixels.

`transform` [matplotlib.transforms.Affine2DBase] If and only if the concrete backend is written such that `option_scale_image()` returns True, an affine transformation may be passed to `draw_image()`. It takes the form of a `Affine2DBase` instance. The translation vector of the transformation is given in physical units (i.e., dots or pixels). Note that the transformation does not override `x` and `y`, and has to be applied before translating the result by `x` and `y` (this can be accomplished by adding `x` and `y` to the translation vector defined by `transform`).

`draw_markers(gc, marker_path, marker_trans, path, trans, rgbFace=None)`

Draws a marker at each of the vertices in `path`. This includes all vertices, including control points on curves. To avoid that behavior, those vertices should be removed before calling this function.

This provides a fallback implementation of `draw_markers` that makes multiple calls to `draw_path()`. Some backends may want to override this method in order to draw the marker only once and reuse it multiple times.

Parameters

`gc` [GraphicsContextBase] The graphics context

`marker_trans` [matplotlib.transforms.Transform] An affine transform applied to the marker.

`trans` [matplotlib.transforms.Transform] An affine transform applied to the path.

`draw_path(gc, path, transform, rgbFace=None)`

Draws a Path instance using the given affine transform.

`draw_tex(gc, x, y, s, prop, angle, ismath='TeX!', mtext=None)`

`draw_text(gc, x, y, s, prop, angle, ismath=False, mtext=None)`

Draw the text instance

Parameters

`gc` [GraphicsContextBase] the graphics context

`x` [scalar] the x location of the text in display coords

`y` [scalar] the y location of the text baseline in display coords

`s` [str] the text string

`prop` [matplotlib.font_manager.FontProperties] font properties

`angle` [scalar] the rotation angle in degrees

`mtext` [matplotlib.text.Text] the original text object to be rendered

Notes

backend implementers note

When you are trying to determine if you have gotten your bounding box right (which is what enables the text layout/alignment to work properly), it helps to change the line in `text.py`:

```
if 0: bbox_artist(self, renderer)
```

to if 1, and then the actual bounding box will be plotted along with your text.

`flipy()`

Return true if y small numbers are top for renderer Is used for drawing text (`matplotlib.text`) and images (`matplotlib.image`) only

`get_canvas_width_height()`

return the canvas width and height in display coords

`get_text_width_height_descent(s, prop, ismath)`

Get the width, height, and descent (offset from the bottom to the baseline), in display coords, of the string `s` with `FontProperties` `prop`

`new_gc()`

Return an instance of a `GraphicsContextBase`

`option_image_nocomposite()`

return whether to generate a composite image from multiple images on a set of axes

`option_scale_image()`

pgf backend supports affine transform of image.

`points_to_pixels(points)`

Convert points to display units

You need to override this function (unless your backend doesn't have a `dpi`, e.g., `postscript` or `svg`). Some imaging systems assume some value for pixels per inch:

$$\text{points to pixels} = \text{points} * \text{pixels_per_inch} / 72.0 * \text{dpi} / 72.0$$

Parameters

`points` [scalar or array_like] a float or a numpy array of float

Returns

Points converted to pixels

`class matplotlib.backends.backend_pgf.TmpDirCleaner`

Bases: `object`

`static add(tmpdir)`

`static cleanup_remaining_tmpdirs()`

`remaining_tmpdirs = {}`

`matplotlib.backends.backend_pgf.common_texification(text)`

Do some necessary and/or useful substitutions for texts to be included in LaTeX documents.

`matplotlib.backends.backend_pgf.get_fontspec()`

Build fontspec preamble from rc.

`matplotlib.backends.backend_pgf.get_preamble()`

Get LaTeX preamble from rc.

`matplotlib.backends.backend_pgf.get_texcommand()`

Get chosen TeX system from rc.

`matplotlib.backends.backend_pgf.make_pdf_to_png_converter()`

Returns a function that converts a pdf file to a png file.

`matplotlib.backends.backend_pgf.repl_escapetext(m)`

`matplotlib.backends.backend_pgf.repl_mathdefault(m)`

`matplotlib.backends.backend_pgf.writeln(fh, line)`

37.14 matplotlib.backends.backend_ps

A PostScript backend, which can produce both PostScript .ps and .eps

matplotlib.backends.backend_ps.FigureCanvas

alias of matplotlib.backends.backend_ps.FigureCanvasPS

class matplotlib.backends.backend_ps.FigureCanvasPS(*figure*)

Bases: matplotlib.backend_bases.FigureCanvasBase

draw()

Render the Figure.

filetypes = {'eps': 'Encapsulated Postscript', 'ps': 'Postscript'}

fixed_dpi = 72

get_default_filetype()

Get the default savefig file format as specified in rcParam savefig.format. Returned string excludes period. Overridden in backends that only support a single file type.

print_eps(*outfile*, *args, **kwargs)

print_ps(*outfile*, *args, **kwargs)

matplotlib.backends.backend_ps.FigureManager

alias of matplotlib.backends.backend_ps.FigureManagerPS

class matplotlib.backends.backend_ps.FigureManagerPS(*canvas*, *num*)

Bases: matplotlib.backend_bases.FigureManagerBase

class matplotlib.backends.backend_ps.GraphicsContextPS

Bases: matplotlib.backend_bases.GraphicsContextBase

get_capstyle()

Return the capstyle as a string in ('butt', 'round', 'projecting')

get_joinstyle()

Return the line join style as one of ('miter', 'round', 'bevel')

shouldstroke()

class matplotlib.backends.backend_ps.PsBackendHelper

Bases: `object`

gs_exe

executable name of ghostscript.

gs_version

version of ghostscript.

`supports_ps2write`

True if the installed ghostscript supports ps2write device.

`class matplotlib.backends.backend_ps.RendererPS(width, height, pswriter, imagedpi=72)`

Bases: `matplotlib.backend_bases.RendererBase`

The renderer handles all the drawing primitives using a graphics context instance that controls the colors/styles.

Although postscript itself is dpi independent, we need to inform the image code about a requested dpi to generate high res images and then scale them before embedding them

`afmfontd = {}`

`create_hatch(hatch)`

`draw_gouraud_triangle(gc, points, colors, trans)`

Draw a Gouraud-shaded triangle.

Parameters

`points` [array_like, shape=(3, 2)] Array of (x, y) points for the triangle.

`colors` [array_like, shape=(3, 4)] RGBA colors for each point of the triangle.

`transform` [matplotlib.transforms.Transform] An affine transform to apply to the points.

`draw_gouraud_triangles(gc, points, colors, trans)`

Draws a series of Gouraud triangles.

Parameters

`points` [array_like, shape=(N, 3, 2)] Array of N (x, y) points for the triangles.

`colors` [array_like, shape=(N, 3, 4)] Array of N RGBA colors for each point of the triangles.

`transform` [matplotlib.transforms.Transform] An affine transform to apply to the points.

`draw_image(gc, x, y, im, transform=None)`

Draw the Image instance into the current axes; x is the distance in pixels from the left hand side of the canvas and y is the distance from bottom

`draw_markers(gc, marker_path, marker_trans, path, trans, rgbFace=None)`

Draw the markers defined by path at each of the positions in x and y. path coordinates are points, x and y coords will be transformed by the transform

`draw_mathtext(gc, x, y, s, prop, angle)`

Draw the math text using `matplotlib.mathtext`

`draw_path(gc, path, transform, rgbFace=None)`

Draws a Path instance using the given affine transform.

`draw_path_collection(gc, master_transform, paths, all_transforms, offsets, offset-
Trans, facecolors, edgecolors, linewidths, linestyle, antialiaseds,
urls, offset_position)`

Draws a collection of paths selecting drawing properties from the lists facecolors, edgecolors, linewidths, linestyle and antialiaseds. offsets is a list of offsets to apply to each of the paths. The offsets in offsets are first transformed by offsetTrans before being applied. offset_position may be either "screen" or "data" depending on the space that the offsets are in.

This provides a fallback implementation of `draw_path_collection()` that makes multiple calls to `draw_path()`. Some backends may want to override this in order to render each set of path data only once, and then reference that path multiple times with the different offsets, colors, styles etc. The generator methods `_iter_collection_raw_paths()` and `_iter_collection()` are provided to help with (and standardize) the implementation across backends. It is highly recommended to use those generators, so that changes to the behavior of `draw_path_collection()` can be made globally.

`draw_tex(gc, x, y, s, prop, angle, ismath='TeX!', mtext=None)`

draw a Text instance

`draw_text(gc, x, y, s, prop, angle, ismath=False, mtext=None)`

Draw a Text instance.

`flipy()`

return true if small y numbers are top for renderer

`get_canvas_width_height()`

return the canvas width and height in display coords

`get_image_magnification()`

Get the factor by which to magnify images passed to `draw_image`. Allows a backend to have images at a different resolution to other artists.

`get_text_width_height_descent(s, prop, ismath)`

get the width and height in display coords of the string s with FontProperty prop

`merge_used_characters(other)`

`new_gc()`

Return an instance of a GraphicsContextBase

`option_image_nocomposite()`

return whether to generate a composite image from multiple images on a set of axes

`option_scale_image()`

ps backend support arbitrary scaling of image.

`set_color(r, g, b, store=1)`

`set_font(fontname, fontsize, store=1)`

`set_linecap(linecap, store=1)`

`set_linedash(offset, seq, store=1)`

`set_linejoin(linejoin, store=1)`

`set_linewidth(linewidth, store=1)`

`track_characters(font, s)`

Keeps track of which characters are required from each font.

`matplotlib.backends.backend_ps.convert_psfrags(tmpfile, psfrags, font_preamble, custom_preamble, paperWidth, paperHeight, orientation)`

When we want to use the LaTeX backend with postscript, we write PSFrag tags to a temporary postscript file, each one marking a position for LaTeX to render some text. `convert_psfrags` generates a LaTeX document containing the commands to convert those tags to text. LaTeX/dvips produces the postscript file that includes the actual text.

`matplotlib.backends.backend_ps.get_bbox(tmpfile, bbox)`

Use ghostscript's `bbbox` device to find the center of the bounding box. Return an appropriately sized `bbbox` centered around that point. A bit of a hack.

`matplotlib.backends.backend_ps.get_bbox_header(lbrt, rotated=False)`

return a postscript header string for the given `bbbox` `lbrt`=(l, b, r, t). Optionally, return rotate command.

`matplotlib.backends.backend_ps.gs_distill(tmpfile, eps=False, ptype='letter', bbox=None, rotated=False)`

Use ghostscript's `pswrite` or `epswrite` device to distill a file. This yields smaller files without illegal encapsulated postscript operators. The output is low-level, converting text to outlines.

`matplotlib.backends.backend_ps.pstoeps(tmpfile, bbox=None, rotated=False)`

Convert the postscript to encapsulated postscript. The `bbbox` of the `eps` file will be replaced with the given `bbbox` argument. If `None`, original `bbbox` will be used.

`matplotlib.backends.backend_ps.quote_ps_string(s)`

Quote dangerous characters of `S` for use in a PostScript string constant.

`matplotlib.backends.backend_ps.xpdf_distill(tmpfile, eps=False, ptype='letter', bbox=None, rotated=False)`

Use ghostscript's `ps2pdf` and `xpdf`'s/`poppler`'s `pdftops` to distill a file. This yields smaller files without illegal encapsulated postscript operators. This distiller is preferred, generating high-level postscript output that treats text as text.

37.15 matplotlib.backends.backend_qt4agg

NOTE Not included, to avoid adding a dependency to building the docs.

37.16 matplotlib.backends.backend_qt4cairo

NOTE Not included, to avoid adding a dependency to building the docs.

37.17 matplotlib.backends.backend_qt5agg

NOTE Not included, to avoid adding a dependency to building the docs.

37.18 matplotlib.backends.backend_qt5cairo

NOTE Not included, to avoid adding a dependency to building the docs.

37.19 matplotlib.backends.backend_svg

matplotlib.backends.backend_svg.FigureCanvas

alias of matplotlib.backends.backend_svg.FigureCanvasSVG

class matplotlib.backends.backend_svg.FigureCanvasSVG(figure)

Bases: matplotlib.backend_bases.FigureCanvasBase

filetypes = {'svg': 'Scalable Vector Graphics', 'svgz': 'Scalable Vector Graphics'}

fixed_dpi = 72

get_default_filetype()

Get the default savefig file format as specified in rcParam savefig.format. Returned string excludes period. Overridden in backends that only support a single file type.

print_svg(filename, *args, **kwargs)

print_svgz(filename, *args, **kwargs)

matplotlib.backends.backend_svg.FigureManager

alias of matplotlib.backends.backend_svg.FigureManagerSVG

class matplotlib.backends.backend_svg.FigureManagerSVG(canvas, num)

Bases: matplotlib.backend_bases.FigureManagerBase

```
class matplotlib.backends.backend_svg.RendererSVG(width, height, svgwriter, base-
                                                name=None, image_dpi=72)
    Bases: matplotlib.backends.RendererBase
    FONT_SCALE = 100.0
```

`close_group(s)`
 Close a grouping element with label `s` Is only currently used by `backend_svg`

`draw_gouraud_triangle(gc, points, colors, trans)`
 Draw a Gouraud-shaded triangle.

Parameters

- `points` [array_like, shape=(3, 2)] Array of (x, y) points for the triangle.
- `colors` [array_like, shape=(3, 4)] RGBA colors for each point of the triangle.
- `transform` [matplotlib.transforms.Transform] An affine transform to apply to the points.

`draw_gouraud_triangles(gc, triangles_array, colors_array, transform)`
 Draws a series of Gouraud triangles.

Parameters

- `points` [array_like, shape=(N, 3, 2)] Array of N (x, y) points for the triangles.
- `colors` [array_like, shape=(N, 3, 4)] Array of N RGBA colors for each point of the triangles.
- `transform` [matplotlib.transforms.Transform] An affine transform to apply to the points.

`draw_image(gc, x, y, im, transform=None)`
 Draw an RGBA image.

Parameters

- `gc` [GraphicsContextBase] a graphics context with clipping information.
- `x` [scalar] the distance in physical units (i.e., dots or pixels) from the left hand side of the canvas.
- `y` [scalar] the distance in physical units (i.e., dots or pixels) from the bottom side of the canvas.
- `im` [array_like, shape=(N, M, 4), dtype=np.uint8] An array of RGBA pixels.
- `transform` [matplotlib.transforms.Affine2DBase] If and only if the concrete backend is written such that `option_scale_image()` returns True, an affine transformation may be passed to `draw_image()`. It takes the form of a `Affine2DBase` instance. The translation vector of the transformation is

given in physical units (i.e., dots or pixels). Note that the transformation does not override `x` and `y`, and has to be applied before translating the result by `x` and `y` (this can be accomplished by adding `x` and `y` to the translation vector defined by `transform`).

`draw_markers(gc, marker_path, marker_trans, path, trans, rgbFace=None)`

Draws a marker at each of the vertices in `path`. This includes all vertices, including control points on curves. To avoid that behavior, those vertices should be removed before calling this function.

This provides a fallback implementation of `draw_markers` that makes multiple calls to `draw_path()`. Some backends may want to override this method in order to draw the marker only once and reuse it multiple times.

Parameters

`gc` [GraphicsContextBase] The graphics context

`marker_trans` [matplotlib.transforms.Transform] An affine transform applied to the marker.

`trans` [matplotlib.transforms.Transform] An affine transform applied to the path.

`draw_path(gc, path, transform, rgbFace=None)`

Draws a Path instance using the given affine transform.

`draw_path_collection(gc, master_transform, paths, all_transforms, offsets, offsetTrans, facecolors, edgecolors, linewidths, linestyle, antialiaseds, urls, offset_position)`

Draws a collection of paths selecting drawing properties from the lists `facecolors`, `edgecolors`, `linewidths`, `linestyle` and `antialiaseds`. `offsets` is a list of offsets to apply to each of the paths. The offsets in `offsets` are first transformed by `offsetTrans` before being applied. `offset_position` may be either "screen" or "data" depending on the space that the offsets are in.

This provides a fallback implementation of `draw_path_collection()` that makes multiple calls to `draw_path()`. Some backends may want to override this in order to render each set of path data only once, and then reference that path multiple times with the different offsets, colors, styles etc. The generator methods `_iter_collection_raw_paths()` and `_iter_collection()` are provided to help with (and standardize) the implementation across backends. It is highly recommended to use those generators, so that changes to the behavior of `draw_path_collection()` can be made globally.

`draw_tex(gc, x, y, s, prop, angle, ismath='TeX!', mtext=None)`

`draw_text(gc, x, y, s, prop, angle, ismath=False, mtext=None)`

Draw the text instance

Parameters

`gc` [GraphicsContextBase] the graphics context

`x` [scalar] the x location of the text in display coords

`y` [scalar] the y location of the text baseline in display coords
`s` [str] the text string
`prop` [matplotlib.font_manager.FontProperties] font properties
`angle` [scalar] the rotation angle in degrees
`mtext` [matplotlib.text.Text] the original text object to be rendered

Notes

backend implementers note

When you are trying to determine if you have gotten your bounding box right (which is what enables the text layout/alignment to work properly), it helps to change the line in `text.py`:

```
if 0: bbox_artist(self, renderer)
```

to if 1, and then the actual bounding box will be plotted along with your text.

`finalize()`

`flipy()`

Return true if y small numbers are top for renderer Is used for drawing text (matplotlib.text) and images (matplotlib.image) only

`fontd = {}`

`get_canvas_width_height()`

return the canvas width and height in display coords

`get_image_magnification()`

Get the factor by which to magnify images passed to `draw_image()`. Allows a backend to have images at a different resolution to other artists.

`get_text_width_height_descent(s, prop, ismath)`

Get the width, height, and descent (offset from the bottom to the baseline), in display coords, of the string `s` with `FontProperties` prop

`open_group(s, gid=None)`

Open a grouping element with label `s`. If `gid` is given, use `gid` as the id of the group.

`option_image_nocomposite()`

return whether to generate a composite image from multiple images on a set of axes

`option_scale_image()`

override this method for renderers that support arbitrary affine transformations in `draw_image()` (most vector backends).

class matplotlib.backends.backend_svg.XMLWriter(file)

Bases: `object`

`close(id)`

`comment(comment)`

`data(text)`

`element(tag, text=None, attrib={}, **extra)`

`end(tag=None, indent=True)`

`flush()`

`start(tag, attrib={}, **extra)`

`matplotlib.backends.backend_svg.escape_attr(s)`

`matplotlib.backends.backend_svg.escape_cdata(s)`

`matplotlib.backends.backend_svg.escape_comment(s)`

`matplotlib.backends.backend_svg.generate_css(attrib={})`

`matplotlib.backends.backend_svg.generate_transform(transform_list=[])`

`matplotlib.backends.backend_svg.short_float_fmt(x)`

Create a short string representation of a float, which is %f formatting with trailing zeros and the decimal point removed.

37.20 matplotlib.backends.backend_tkagg

`matplotlib.backends.backend_tkagg.FigureCanvas`

alias of `matplotlib.backends.backend_tkagg.FigureCanvasTkAgg`

`class matplotlib.backends.backend_tkagg.FigureCanvasTkAgg(figure, master=None, re-size_callback=None)`

Bases: `matplotlib.backends.backend_agg.FigureCanvasAgg`, `matplotlib.backends._backend_tk.FigureCanvasTk`

`blit(bbox=None)`

Blit the canvas in bbox (default entire canvas).

`draw()`

Draw the figure using the renderer

`class matplotlib.backends.backend_tkagg.FigureManagerTkAgg(**kwargs)`

Bases: `matplotlib.backends._backend_tk.FigureManagerTk`

Deprecated since version 2.2: The `FigureManagerTkAgg` class was deprecated in version 2.2.

`class matplotlib.backends.backend_tkagg.NavigationToolbar2TkAgg(**kwargs)`

Bases: `matplotlib.backends._backend_tk.NavigationToolbar2Tk`

Deprecated since version 2.2: The `NavigationToolbar2TkAgg` class was deprecated in version 2.2.

37.21 `matplotlib.backends.backend_webagg`

Note: The WebAgg backend is not documented here, in order to avoid adding Tornado to the doc build requirements.

37.22 `matplotlib.backends.backend_wxagg`

`matplotlib.backends.backend_wxagg.FigureCanvas`

alias of `matplotlib.backends.backend_wxagg.FigureCanvasWxAgg`

`class matplotlib.backends.backend_wxagg.FigureCanvasWxAgg(parent, id, figure)`

Bases: `matplotlib.backends.backend_agg.FigureCanvasAgg`, `matplotlib.backends.backend_wx._FigureCanvasWxBase`

The `FigureCanvas` contains the figure and does event handling.

In the wxPython backend, it is derived from `wxPanel`, and (usually) lives inside a frame instantiated by a `FigureManagerWx`. The parent window probably implements a `wxSizer` to control the displayed control size - but we give a hint as to our preferred minimum size.

Initialise a `FigureWx` instance.

- Initialise the `FigureCanvasBase` and `wxPanel` parents.
- Set event handlers for: `EVT_SIZE` (Resize event) `EVT_PAINT` (Paint event)

`blit(bbox=None)`

Transfer the region of the agg buffer defined by `bbox` to the display. If `bbox` is `None`, the entire buffer is transferred.

`draw(drawDC=None)`

Render the figure using agg.

`filetypes = {'eps': 'Encapsulated Postscript', 'jpeg': 'Joint Photographic Experts Group', 'jpg': 'Joint Photographic Experts Group', 'png': 'Portable Network Graphics', 'tif': 'Tagged Image File Format', 'tiff': 'Tagged Image File Format'}`


```
class matplotlib.backends.backend_wxagg.FigureFrameWxAgg(num, fig)
    Bases: matplotlib.backends.backend_wx.FigureFrameWx
    get_canvas(fig)
```

```
class matplotlib.backends.backend_wxagg.Toolbar(**kwargs)
    Bases: matplotlib.backends.backend_wx.NavigationToolbar2Wx

    Deprecated since version 2.2: The Toolbar class was deprecated in version 2.2. Use NavigationToolbar2WxAgg instead.
```

38.1 matplotlib.cbook

A collection of utility functions and classes. Originally, many (but not all) were from the Python Cookbook – hence the name cbook.

This module is safe to import from anywhere within matplotlib; it imports matplotlib only at runtime.

```
class matplotlib.cbook.Bunch(**kwds)
```

Bases: `object`

Often we want to just collect a bunch of stuff together, naming each item of the bunch; a dictionary's OK for that, but a small do-nothing class is even handier, and prettier to use. Whenever you want to group a few variables:

```
>>> point = Bunch(datum=2, squared=4, coord=12)
>>> point.datum
```

By: Alex Martelli

From: <https://code.activestate.com/recipes/121294/>

```
class matplotlib.cbook.CallbackRegistry(exception_handler=<function          __except-
                                         tion_printer>)
```

Bases: `object`

Handle registering and disconnecting for a set of signals and callbacks:

```
>>> def oneat(x):
...     print('eat', x)
>>> def ondrink(x):
...     print('drink', x)
```

```
>>> from matplotlib.cbook import CallbackRegistry
>>> callbacks = CallbackRegistry()
```

```
>>> id_eat = callbacks.connect('eat', oneat)
>>> id_drink = callbacks.connect('drink', ondrink)
```

```
>>> callbacks.process('drink', 123)
drink 123
>>> callbacks.process('eat', 456)
eat 456
>>> callbacks.process('be merry', 456) # nothing will be called
>>> callbacks.disconnect(id_eat)
>>> callbacks.process('eat', 456)      # nothing will be called
```

In practice, one should always disconnect all callbacks when they are no longer needed to avoid dangling references (and thus memory leaks). However, real code in matplotlib rarely does so, and due to its design, it is rather difficult to place this kind of code. To get around this, and prevent this class of memory leaks, we instead store weak references to bound methods only, so when the destination object needs to die, the CallbackRegistry won't keep it alive. The Python stdlib weakref module can not create weak references to bound methods directly, so we need to create a proxy object to handle weak references to bound methods (or regular free functions). This technique was shared by Peter Parente on his "[Mindtrove](#)" blog.

Parameters

`exception_handler` [callable, optional] If provided must have signature

```
def handler(exc: Exception) -> None:
```

If not None this function will be called with any `Exception` subclass raised by the callbacks in `CallbackRegistry.process`. The handler may either consume the exception or re-raise.

The callable must be pickle-able.

The default handler is

```
def h(exc):
    traceback.print_exc()
```

`connect(s, func)`

Register func to be called when signal s is generated.

`disconnect(cid)`

Disconnect the callback registered with callback id cid.

`process(s, *args, **kwargs)`

Process signal s.

All of the functions registered to receive callbacks on s will be called with *args and **kwargs.

`class matplotlib.cbook.GetRealpathAndStat`

Bases: `object`

`class matplotlib.cbook.Grouper(init=())`

Bases: `object`

This class provides a lightweight way to group arbitrary objects together into disjoint sets when a full-blown graph data structure would be overkill.

Objects can be joined using `join()`, tested for connectedness using `joined()`, and all disjoint sets can be retrieved by using the object as an iterator.

The objects being joined must be hashable and weak-referenceable.

For example:

```
>>> from matplotlib.cbook import Grouper
>>> class Foo(object):
...     def __init__(self, s):
...         self.s = s
...     def __repr__(self):
...         return self.s
...
>>> a, b, c, d, e, f = [Foo(x) for x in 'abcdef']
>>> grp = Grouper()
>>> grp.join(a, b)
>>> grp.join(b, c)
>>> grp.join(d, e)
>>> sorted(map(tuple, grp))
[(a, b, c), (d, e)]
>>> grp.joined(a, b)
True
>>> grp.joined(a, c)
True
>>> grp.joined(a, d)
False
```

`clean()`

Clean dead weak references from the dictionary

`get_siblings(a)`

Returns all of the items joined with a, including itself.

`join(a, *args)`

Join given arguments into the same set. Accepts one or more arguments.

`joined(a, b)`

Returns True if a and b are members of the same set.

`remove(a)`

exception `matplotlib.cbook.IgnoredKeywordWarning`

Bases: `UserWarning`

A class for issuing warnings about keyword arguments that will be ignored by matplotlib

class `matplotlib.cbook.Locked(path)`

Bases: `object`

Context manager to handle locks.

Based on code from conda.

(c) 2012-2013 Continuum Analytics, Inc. / <https://www.continuum.io/> All Rights Reserved

conda is distributed under the terms of the BSD 3-clause license. Consult LICENSE_CONDA or <https://opensource.org/licenses/BSD-3-Clause>.

LOCKFN = '.matplotlib_lock'

exception TimeoutError

Bases: RuntimeError

class matplotlib.cbook.Null(**kwargs)

Bases: object

Deprecated since version 2.1: The Null class was deprecated in version 2.1.

Null objects always and reliably "do nothing."

class matplotlib.cbook.RingBuffer(**kwargs)

Bases: object

Deprecated since version 2.1: The RingBuffer class was deprecated in version 2.1.

class that implements a not-yet-full buffer

append(x)

append an element at the end of the buffer

get()

Return a list of elements from the oldest to the newest.

class matplotlib.cbook.Sorter(**kwargs)

Bases: object

Deprecated since version 2.1: sorted(..., key=itemgetter(...))

Sort by attribute or item

Example usage:

```
sort = Sorter()

list = [(1, 2), (4, 8), (0, 3)]
dict = [{'a': 3, 'b': 4}, {'a': 5, 'b': 2}, {'a': 0, 'b': 0},
        {'a': 9, 'b': 9}]

sort(list)      # default sort
sort(list, 1)   # sort by index 1
sort(dict, 'a') # sort a list of dicts by key 'a'
```

byAttribute(data, attributename, inplace=1)

```
byItem(data, itemindex=None, inplace=1)
```

```
sort(data, itemindex=None, inplace=1)
```

```
class matplotlib.cbook.Stack(default=None)
```

Bases: `object`

Implement a stack where elements can be pushed on and you can move back and forth. But no pop. Should mimic home / back / forward in a browser

```
back()
```

move the position back and return the current element

```
bubble(o)
```

raise o to the top of the stack and return o. o must be in the stack

```
clear()
```

empty the stack

```
empty()
```

```
forward()
```

move the position forward and return the current element

```
home()
```

push the first element onto the top of the stack

```
push(o)
```

push object onto stack at current position - all elements occurring later than the current position are discarded

```
remove(o)
```

remove element o from the stack

```
class matplotlib.cbook.Xlator(**kwargs)
```

Bases: `dict`

Deprecated since version 2.1: The Xlator class was deprecated in version 2.1.

All-in-one multiple-string-substitution class

Example usage:

```
text = "Larry Wall is the creator of Perl"
adict = {
    "Larry Wall" : "Guido van Rossum",
    "creator" : "Benevolent Dictator for Life",
    "Perl" : "Python",
}

print(multiple_replace(adict, text))
```

(continues on next page)

(continued from previous page)

```
xlat = Xlator(adict)
print(xlat.xlat(text))
```

```
xlat(text)
```

Translate text, returns the modified text.

`matplotlib.cbook.align_iterators(func, *iterables)`

Deprecated since version 2.2: The `align_iterators` function was deprecated in version 2.2.

This generator takes a bunch of iterables that are ordered by `func`. It sends out ordered tuples:

```
(func(row), [rows from all iterators matching func(row)])
```

It is used by `matplotlib.mlab.recs_join()` to join record arrays

`matplotlib.cbook.allequal(seq)`

Deprecated since version 2.1: The `allequal` function was deprecated in version 2.1.

Return True if all elements of `seq` compare equal. If `seq` is 0 or 1 length, return True

`matplotlib.cbook.allpairs(x)`

Deprecated since version 2.1: The `allpairs` function was deprecated in version 2.1.

return all possible pairs in sequence `x`

`matplotlib.cbook.alltrue(seq)`

Deprecated since version 2.1: The `alltrue` function was deprecated in version 2.1.

Return True if all elements of `seq` evaluate to True. If `seq` is empty, return False.

`matplotlib.cbook.boxplot_stats(X, whis=1.5, bootstrap=None, labels=None, autorange=False)`

Returns list of dictionaries of statistics used to draw a series of box and whisker plots. The Returns section enumerates the required keys of the dictionary. Users can skip this function and pass a user-defined set of dictionaries to the new axes.bxp method instead of relying on MPL to do the calculations.

Parameters

X [array-like] Data that will be represented in the boxplots. Should have 2 or fewer dimensions.

whis [float, string, or sequence (default = 1.5)] As a float, determines the reach of the whiskers to the beyond the first and third quartiles. In other words, where IQR is the interquartile range (Q3-Q1), the upper whisker will extend to last datum less than $Q3 + whis * IQR$. Similarly, the lower whisker will extend to the first datum greater than $Q1 - whis * IQR$. Beyond the whiskers, data are considered outliers and are plotted as individual points. This can be set this to an ascending sequence of percentile (e.g., [5, 95]) to set the whiskers at specific percentiles of the data. Finally, `whis` can be the string 'range' to force the whiskers to the minimum and maximum of the data. In

the edge case that the 25th and 75th percentiles are equivalent, `whis` can be automatically set to 'range' via the `autorange` option.

`bootstrap` [int, optional] Number of times the confidence intervals around the median should be bootstrapped (percentile method).

`labels` [array-like, optional] Labels for each dataset. Length must be compatible with dimensions of `X`.

`autorange` [bool, optional (False)] When `True` and the data are distributed such that the 25th and 75th percentiles are equal, `whis` is set to 'range' such that the whisker ends are at the minimum and maximum of the data.

Returns

`bxpstats` [list of dict] A list of dictionaries containing the results for each column of data. Keys of each dictionary are the following:

Key	Value Description
<code>label</code>	tick label for the boxplot
<code>mean</code>	arithmetic mean value
<code>med</code>	50th percentile
<code>q1</code>	first quartile (25th percentile)
<code>q3</code>	third quartile (75th percentile)
<code>cilo</code>	lower notch around the median
<code>cihi</code>	upper notch around the median
<code>whislo</code>	end of the lower whisker
<code>whishi</code>	end of the upper whisker
<code>fliers</code>	outliers

Notes

Non-bootstrapping approach to confidence interval uses Gaussian- based asymptotic approximation:

$$\text{med} \pm 1.57 \times \frac{\text{iqr}}{\sqrt{N}}$$

General approach from: McGill, R., Tukey, J.W., and Larsen, W.A. (1978) "Variations of Boxplots", The American Statistician, 32:12-16.

`matplotlib.cbook.contiguous_regions(mask)`

Return a list of (ind0, ind1) such that `mask[ind0:ind1].all()` is `True` and we cover all such regions

`class matplotlib.cbook.converter(**kwargs)`

Bases: `object`

Deprecated since version 2.1: The converter class was deprecated in version 2.1.

Base class for handling string -> python type with support for missing values

`is_missing(s)`

`matplotlib.cbook.dedent(s)`

Remove excess indentation from docstring `s`.

Discards any leading blank lines, then removes up to `n` whitespace characters from each line, where `n` is the number of leading whitespace characters in the first line. It differs from `textwrap.dedent` in its deletion of leading blank lines and its use of the first non-blank line to determine the indentation.

It is also faster in most cases.

`matplotlib.cbook.delete_masked_points(*args)`

Find all masked and/or non-finite points in a set of arguments, and return the arguments with only the unmasked points remaining.

Arguments can be in any of 5 categories:

- 1) 1-D masked arrays
- 2) 1-D ndarrays
- 3) ndarrays with more than one dimension
- 4) other non-string iterables
- 5) anything else

The first argument must be in one of the first four categories; any argument with a length differing from that of the first argument (and hence anything in category 5) then will be passed through unchanged.

Masks are obtained from all arguments of the correct length in categories 1, 2, and 4; a point is bad if masked in a masked array or if it is a `nan` or `inf`. No attempt is made to extract a mask from categories 2, 3, and 4 if `np.isfinite()` does not yield a Boolean array.

All input arguments that are not passed unchanged are returned as ndarrays after removing the points or rows corresponding to masks in any of the arguments.

A vastly simpler version of this function was originally written as a helper for `Axes.scatter()`.

`matplotlib.cbook.dict_delall(d, keys)`

Deprecated since version 2.1: The `dict_delall` function was deprecated in version 2.1.

delete all of the keys from the `dict` `d`

`matplotlib.cbook.exception_to_str(s=None)`

Deprecated since version 2.1: The `exception_to_str` function was deprecated in version 2.1.

`matplotlib.cbook.file_requires_unicode(x)`

Returns `True` if the given writable file-like object requires Unicode to be written to it.

`matplotlib.cbook.finddir(o, match, case=False)`

Deprecated since version 2.1: The `finddir` function was deprecated in version 2.1.

return all attributes of `o` which match string in `match`. if `case` is `True` require an exact case match.

`matplotlib.cbook.flatten(seq, scalarp=<function is_scalar_or_string>)`

Returns a generator of flattened nested containers

For example:

```
>>> from matplotlib.cbook import flatten
>>> l = (('John', ['Hunter']), (1, 23), [[[42, (5, 23)], ]])
>>> print(list(flatten(l)))
['John', 'Hunter', 1, 23, 42, 5, 23]
```

By: Composite of Holger Krekel and Luther Blissett From: <https://code.activestate.com/recipes/121294/> and Recipe 1.12 in cookbook

`matplotlib.cbook.get_label(y, default_name)`

`matplotlib.cbook.get_recursive_filelist(args)`

Deprecated since version 2.1: The `get_recursive_filelist` function was deprecated in version 2.1.

Recurse all the files and dirs in `args` ignoring symbolic links and return the files as a list of strings

`matplotlib.cbook.get_sample_data(fname, asfileobj=True)`

Return a sample data file. `fname` is a path relative to the `mpl-data/sample_data` directory. If `asfileobj` is `True` return a file object, otherwise just a file path.

Set the `rc` parameter `examples.directory` to the directory where we should look, if `sample_data` files are stored in a location different than default (which is `'mpl-data/sample_data'` at the same level of `'matplotlib'` Python module files).

If the filename ends in `.gz`, the file is implicitly unzipped.

`matplotlib.cbook.get_split_ind(seq, N)`

Deprecated since version 2.1: The `get_split_ind` function was deprecated in version 2.1.

`seq` is a list of words. Return the index into `seq` such that:

```
len(' '.join(seq[:ind]))<=N
```

.

`matplotlib.cbook.index_of(y)`

A helper function to get the index of an input to plot against if `x` values are not explicitly given.

Tries to get `y.index` (works if this is a `pd.Series`), if that fails, return `np.arange(y.shape[0])`.

This will be extended in the future to deal with more types of labeled data.

Parameters

`y` [scalar or array-like] The proposed `y`-value

Returns

`x, y` [ndarray] The x and y values to plot.

`matplotlib.cbook.is_hashable(obj)`

Returns true if obj can be hashed

`matplotlib.cbook.is_math_text(s)`

`matplotlib.cbook.is_numlike(obj)`

return true if obj looks like a number

`matplotlib.cbook.is_scalar(obj)`

Deprecated since version 2.1: The `is_scalar` function was deprecated in version 2.1.

return true if obj is not string like and is not iterable

`matplotlib.cbook.is_scalar_or_string(val)`

Return whether the given object is a scalar or string like.

`matplotlib.cbook.is_sequence_of_strings(obj)`

Deprecated since version 2.1: The `is_sequence_of_strings` function was deprecated in version 2.1.

Returns true if obj is iterable and contains strings

`matplotlib.cbook.is_string_like(obj)`

Deprecated since version 2.1: The `is_string_like` function was deprecated in version 2.1.

Return True if obj looks like a string

`matplotlib.cbook.is_writable_file_like(obj)`

return true if obj looks like a file object with a write method

`matplotlib.cbook.issubclass_safe(x, klass)`

Deprecated since version 2.1: The `issubclass_safe` function was deprecated in version 2.1.

return `issubclass(x, klass)` and return False on a `TypeError`

`matplotlib.cbook.iterable(obj)`

return true if obj is iterable

`matplotlib.cbook.listFiles(root, patterns='*', recurse=1, return_folders=0)`

Recursively list files

from Parmar and Martelli in the Python Cookbook

`matplotlib.cbook.local_over_kwdict(local_var, kwargs, *keys)`

Enforces the priority of a local variable over potentially conflicting argument(s) from a kwargs dict. The following possible output values are considered in order of priority:

`local_var > kwargs[keys[0]] > ... > kwargs[keys[-1]]`

The first of these whose value is not None will be returned. If all are None then None will be returned. Each key in keys will be removed from the kwargs dict in place.

Parameters

`local_var`: any object

The local variable (highest priority)

kwargs: dict Dictionary of keyword arguments; modified in place

keys: str(s) Name(s) of keyword arguments to process, in descending order of priority

Returns

out: any object Either local_var or one of kwargs[key] for key in keys

Raises

IgnoredKeywordWarning For each key in keys that is removed from kwargs but not used as the output value

class matplotlib.cbook.maxdict(maxsize)

Bases: dict

A dictionary with a maximum size; this doesn't override all the relevant methods to constrain the size, just setitem, so use with caution

matplotlib.cbook.mkdirs(newdir, mode=511)

make directory newdir recursively, and set mode. Equivalent to

```
> mkdir -p NEWDIR
> chmod MODE NEWDIR
```

matplotlib.cbook.normalize_kwargs(kw, alias_mapping=None, required=(), forbidden=(), allowed=None)

Helper function to normalize kwarg inputs

The order they are resolved are:

1. aliasing
2. required
3. forbidden
4. allowed

This order means that only the canonical names need appear in allowed, forbidden, required

Parameters

alias_mapping, dict, optional A mapping between a canonical name to a list of aliases, in order of precedence from lowest to highest.

If the canonical value is not in the list it is assumed to have the highest priority.

required [iterable, optional] A tuple of fields that must be in kwargs.

forbidden [iterable, optional] A list of keys which may not be in kwargs

allowed [tuple, optional] A tuple of allowed fields. If this not None, then raise if kw contains any keys not in the union of required and allowed. To allow only the required fields pass in () for allowed

Raises

TypeError To match what python raises if invalid args/kwargs are passed to a callable.

matplotlib.cbook.onetrue(seq)

Deprecated since version 2.1: The onetrue function was deprecated in version 2.1.

Return True if one element of seq is True. If seq is empty, return False.

matplotlib.cbook.open_file_cm(path_or_file, mode='r', encoding=None)

Pass through file objects and context-manage PathLikes.

matplotlib.cbook.pieces(seq, num=2)

Deprecated since version 2.1: The pieces function was deprecated in version 2.1.

Break up the seq into num tuples

matplotlib.cbook.print_cycles(objects, outstream=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>, show_progress=False)

objects A list of objects to find cycles in. It is often useful to pass in gc.garbage to find the cycles that are preventing some objects from being garbage collected.

outstream The stream for output.

show_progress If True, print the number of objects reached as they are found.

matplotlib.cbook.pts_to_midstep(x, *args)

Convert continuous line to mid-steps.

Given a set of N points convert to 2N points which when connected linearly give a step function which changes values at the middle of the intervals.

Parameters

x [array] The x location of the steps. May be empty.

y1, ..., yp [array] y arrays to be turned into steps; all must be the same length as x.

Returns

out [array] The x and y values converted to steps in the same order as the input; can be unpacked as x_out, y1_out, ..., yp_out. If the input is length N, each of these arrays will be length 2N.

Examples

```
>>> x_s, y1_s, y2_s = pts_to_midstep(x, y1, y2)
```

`matplotlib.cbook.pts_to_poststep(x, *args)`

Convert continuous line to post-steps.

Given a set of N points convert to $2N + 1$ points, which when connected linearly give a step function which changes values at the end of the intervals.

Parameters

`x` [array] The x location of the steps. May be empty.

`y1, ..., yp` [array] y arrays to be turned into steps; all must be the same length as `x`.

Returns

`out` [array] The x and y values converted to steps in the same order as the input; can be unpacked as `x_out, y1_out, ..., yp_out`. If the input is length N , each of these arrays will be length $2N + 1$. For $N=0$, the length will be 0.

Examples

```
>> x_s, y1_s, y2_s = pts_to_poststep(x, y1, y2)
```

`matplotlib.cbook.pts_to_prestep(x, *args)`

Convert continuous line to pre-steps.

Given a set of N points, convert to $2N - 1$ points, which when connected linearly give a step function which changes values at the beginning of the intervals.

Parameters

`x` [array] The x location of the steps. May be empty.

`y1, ..., yp` [array] y arrays to be turned into steps; all must be the same length as `x`.

Returns

`out` [array] The x and y values converted to steps in the same order as the input; can be unpacked as `x_out, y1_out, ..., yp_out`. If the input is length N , each of these arrays will be length $2N + 1$. For $N=0$, the length will be 0.

Examples

```
>> x_s, y1_s, y2_s = pts_to_prestep(x, y1, y2)
```

`matplotlib.cbook.recursive_remove(path)`

Deprecated since version 2.1: The `recursive_remove` function was deprecated in version 2.1. Use `shutil.rmtree` instead.

`matplotlib.cbook.report_memory(i=0)`

return the memory consumed by process

`matplotlib.cbook.restrict_dict(d, keys)`

Deprecated since version 2.1: The `restrict_dict` function was deprecated in version 2.1.

Return a dictionary that contains those keys that appear in both `d` and `keys`, with values from `d`.

`matplotlib.cbook.reverse_dict(d)`

Deprecated since version 2.1: The `reverse_dict` function was deprecated in version 2.1.

reverse the dictionary – may lose data if values are not unique!

`matplotlib.cbook.safe_first_element(obj)`

`matplotlib.cbook.safe_masked_invalid(x, copy=False)`

`matplotlib.cbook.safezip(*args)`

make sure args are equal len before zipping

`matplotlib.cbook.sanitize_sequence(data)`

Converts dictview object to list

`class matplotlib.cbook.silent_list(type, seq=None)`

Bases: `list`

override repr when returning a list of matplotlib artists to prevent long, meaningless output. This is meant to be used for a homogeneous list of a given type

`matplotlib.cbook.simple_linear_interpolation(a, steps)`

Resample an array with steps - 1 points between original point pairs.

Parameters

`a` [array, shape (n, ...)]

`steps` [int]

Returns

array, shape “((n - 1) * steps + 1, ...)”

Along each column of `*a*`, “(steps - 1)” points are introduced between each original values; the values are linearly interpolated.

`matplotlib.cbook.soundex(name, len=4)`

Deprecated since version 2.1: The `soundex` function was deprecated in version 2.1.

soundex module conforming to Odell-Russell algorithm

`matplotlib.cbook.strip_math(s)`

remove latex formatting from mathtext

`matplotlib.cbook.to_filehandle(fname, flag='rU', return_opened=False, encoding=None)`

`fname` can be an `os.PathLike` or a file handle. Support for gzipped files is automatic, if the filename ends in `.gz`. `flag` is a read/write flag for `file()`

`class matplotlib.cbook.todate(**kwargs)`

Bases: `matplotlib.cbook.converter`

Deprecated since version 2.1: The `todate` class was deprecated in version 2.1.

convert to a date or None

`class matplotlib.cbook.todatetime(**kwargs)`

Bases: `matplotlib.cbook.converter`

Deprecated since version 2.1: The `todatetime` class was deprecated in version 2.1.

convert to a datetime or None

`class matplotlib.cbook.tofloat(**kwargs)`

Bases: `matplotlib.cbook.converter`

Deprecated since version 2.1: The `tofloat` class was deprecated in version 2.1.

convert to a float or None

`class matplotlib.cbook.toint(**kwargs)`

Bases: `matplotlib.cbook.converter`

Deprecated since version 2.1: The `toint` class was deprecated in version 2.1.

convert to an int or None

`class matplotlib.cbook.tostr(**kwargs)`

Bases: `matplotlib.cbook.converter`

Deprecated since version 2.1: The `tostr` class was deprecated in version 2.1.

convert to string or None

`matplotlib.cbook.unicode_safe(s)`

`matplotlib.cbook.unique(x)`

Deprecated since version 2.1: The `unique` function was deprecated in version 2.1.

Return a list of unique elements of x

`matplotlib.cbook.unmasked_index_ranges(mask, compressed=True)`

Deprecated since version 2.1: The `unmasked_index_ranges` function was deprecated in version 2.1.

Find index ranges where mask is False.

mask will be flattened if it is not already 1-D.

Returns Nx2 [numpy.ndarray](#) with each row the start and stop indices for slices of the compressed [numpy.ndarray](#) corresponding to each of N uninterrupted runs of unmasked values. If optional argument `compressed` is False, it returns the start and stop indices into the original [numpy.ndarray](#), not the compressed [numpy.ndarray](#). Returns None if there are no unmasked values.

Example:


```
y = ma.array(np.arange(5), mask = [0,0,1,0,0])
ii = unmasked_index_ranges(ma.getmaskarray(y))
# returns array [[0,2,] [2,4,]]

y.compressed()[ii[1,0]:ii[1,1]]
# returns array [3,4,]

ii = unmasked_index_ranges(ma.getmaskarray(y), compressed=False)
# returns array [[0, 2], [3, 5]]

y.filled()[ii[1,0]:ii[1,1]]
# returns array [3,4,]
```

Prior to the transforms refactoring, this was used to support masked arrays in Line2D.

`matplotlib.cbook.violin_stats(X, method, points=100)`

Returns a list of dictionaries of data which can be used to draw a series of violin plots. See the Returns section below to view the required keys of the dictionary. Users can skip this function and pass a user-defined set of dictionaries to the `axes.vplot` method instead of using MPL to do the calculations.

Parameters

- `X` [array-like] Sample data that will be used to produce the gaussian kernel density estimates. Must have 2 or fewer dimensions.
- `method` [callable] The method used to calculate the kernel density estimate for each column of data. When called via `method(v, coords)`, it should return a vector of the values of the KDE evaluated at the values specified in `coords`.
- `points` [scalar, default = 100] Defines the number of points to evaluate each of the gaussian kernel density estimates at.

Returns

A list of dictionaries containing the results for each column of data.

The dictionaries contain at least the following:

- `coords`: A list of scalars containing the coordinates this particular kernel density estimate was evaluated at.
- `vals`: A list of scalars containing the values of the kernel density estimate at each of the coordinates given in `coords`.
- `mean`: The mean value for this column of data.
- `median`: The median value for this column of data.
- `min`: The minimum value for this column of data.
- `max`: The maximum value for this column of data.

`matplotlib.cbook.wrap(prefix, text, cols)`

Deprecated since version 2.1: The `wrap` function was deprecated in version 2.1. Use `textwrap.TextWrapper` instead.

wrap text with prefix at length cols

CM (COLORMAP)

39.1 matplotlib.cm

Builtin colormaps, colormap handling utilities, and the ScalarMappable mixin.

See [/gallery/color/colormap_reference](#) for a list of builtin colormaps. See [Colormaps in Matplotlib](#) for an in-depth discussion of colormaps.

`class matplotlib.cm.ScalarMappable(norm=None, cmap=None)`

Bases: `object`

This is a mixin class to support scalar data to RGBA mapping. The `ScalarMappable` makes use of data normalization before returning RGBA colors from the given colormap.

Parameters

`norm` [`matplotlib.colors.Normalize` instance] The normalizing object which scales data, typically into the interval `[0, 1]`. If `None`, `norm` defaults to a `colors.Normalize` object which initializes its scaling based on the first data processed.

`cmap` [`str` or `Colormap` instance] The colormap used to map normalized data values to RGBA colors.

`add_checker(checker)`

Add an entry to a dictionary of boolean flags that are set to `True` when the mappable is changed.

`autoscale()`

Autoscale the scalar limits on the `norm` instance using the current array

`autoscale__None()`

Autoscale the scalar limits on the `norm` instance using the current array, changing only limits that are `None`

`changed()`

Call this whenever the mappable is changed to notify all the `callbackSM` listeners to the 'changed' signal

`check_update(checker)`

If mappable has changed since the last check, return `True`; else return `False`

`cmap = None`

The Colormap instance of this ScalarMappable.

`colorbar = None`

The last colorbar associated with this ScalarMappable. May be None.

`get_array()`

Return the array

`get_clim()`

return the min, max of the color limits for image scaling

`get_cmap()`

return the colormap

`norm = None`

The Normalization instance of this ScalarMappable.

`set_array(A)`

Set the image array from numpy array A.

Parameters

A [ndarray]

`set_clim(vmin=None, vmax=None)`

set the norm limits for image scaling; if `vmin` is a length2 sequence, interpret it as (`vmin`, `vmax`) which is used to support `setp`

ACCEPTS: a length 2 sequence of floats; may be overridden in methods that have `vmin` and `vmax` kwargs.

`set_cmap(cmap)`

set the colormap for luminance data

ACCEPTS: a colormap or registered colormap name

`set_norm(norm)`

Set the normalization instance.

Parameters

norm [Normalize]

`to_rgba(x, alpha=None, bytes=False, norm=True)`

Return a normalized rgba array corresponding to `x`.

In the normal case, `x` is a 1-D or 2-D sequence of scalars, and the corresponding ndarray of rgba values will be returned, based on the norm and colormap set for this ScalarMappable.

There is one special case, for handling images that are already rgb or rgba, such as might have been read from an image file. If `x` is an ndarray with 3 dimensions, and the last dimension is either 3 or 4, then it will be treated as an rgb or rgba array, and no mapping will be done. The array can be uint8, or it can be floating point with values in the 0-1 range; otherwise a `ValueError` will be raised. If it is a masked array, the mask will be ignored. If the last dimension is 3, the `alpha` kwarg (defaulting to 1) will be used

to fill in the transparency. If the last dimension is 4, the alpha kwarg is ignored; it does not replace the pre-existing alpha. A `ValueError` will be raised if the third dimension is other than 3 or 4.

In either case, if `bytes` is `False` (default), the `rgba` array will be floats in the 0-1 range; if it is `True`, the returned `rgba` array will be `uint8` in the 0 to 255 range.

If `norm` is `False`, no normalization of the input data is performed, and it is assumed to be in the range (0-1).

`matplotlib.cm.get_cmap(name=None, lut=None)`

Get a colormap instance, defaulting to `rc` values if `name` is `None`.

Colormaps added with `register_cmap()` take precedence over built-in colormaps.

If `name` is a `matplotlib.colors.Colormap` instance, it will be returned.

If `lut` is not `None` it must be an integer giving the number of entries desired in the lookup table, and `name` must be a standard `mpl` colormap name.

`matplotlib.cm.register_cmap(name=None, cmap=None, data=None, lut=None)`

Add a colormap to the set recognized by `get_cmap()`.

It can be used in two ways:

```
register_cmap(name='swirly', cmap=swirly_cmap)

register_cmap(name='choppy', data=choppydata, lut=128)
```

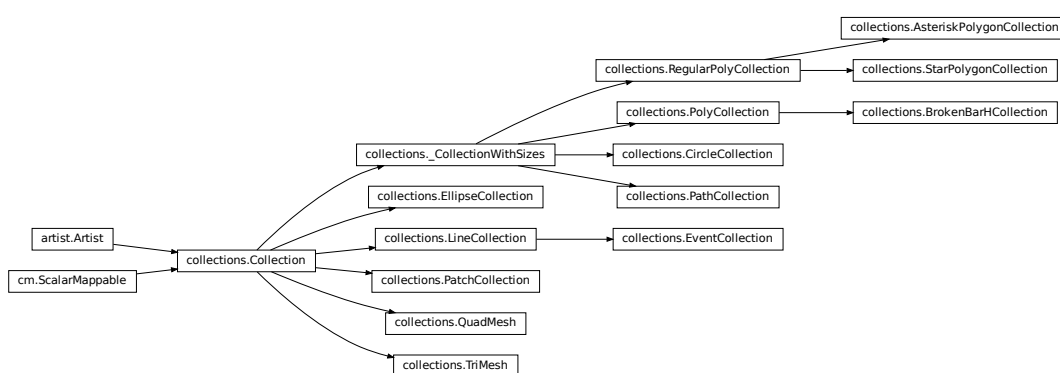
In the first case, `cmap` must be a `matplotlib.colors.Colormap` instance. The `name` is optional; if absent, the `name` will be the `name` attribute of the `cmap`.

In the second case, the three arguments are passed to the `LinearSegmentedColormap` initializer, and the resulting colormap is registered.

`matplotlib.cm.revcmmap(data)`

Can only handle specification data in dictionary format.

COLLECTIONS



40.1 matplotlib.collections

Classes for the efficient drawing of large collections of objects that share most properties, e.g., a large number of line segments or polygons.

The classes are not meant to be as flexible as their single element counterparts (e.g., you may not be able to select all line styles) but they are meant to be fast for common use cases (e.g., a large set of solid line segments)

```
class matplotlib.collections.AsteriskPolygonCollection(numsides, rotation=0, sizes=(1, ),
                                                    **kwargs)
```

Bases: matplotlib.collections.RegularPolyCollection

Draw a collection of regular asterisks with numsides points.

numsides the number of sides of the polygon

rotation the rotation of the polygon in radians

sizes gives the area of the circle circumscribing the regular polygon in points²

Valid Collection keyword arguments:

- `edgecolors`: None
- `facecolors`: None
- `linewidths`: None
- `antialiaseds`: None
- `offsets`: None
- `transOffset`: `transforms.IdentityTransform()`
- `norm`: None (optional for `matplotlib.cm.ScalarMappable`)
- `cmap`: None (optional for `matplotlib.cm.ScalarMappable`)

`offsets` and `transOffset` are used to translate the patch after rendering (default no offsets)

If any of `edgecolors`, `facecolors`, `linewidths`, `antialiaseds` are None, they default to their `matplotlib.rcParams` patch setting, in sequence form.

Example: see `examples/dynamic_collection.py` for complete example:

```
offsets = np.random.rand(20,2)
facecolors = [cm.jet(x) for x in np.random.rand(20)]
black = (0,0,0,1)

collection = RegularPolyCollection(
    numsides=5, # a pentagon
    rotation=0, sizes=(50,),
    facecolors = facecolors,
    edgecolors = (black,),
    linewidths = (1,),
    offsets = offsets,
    transOffset = ax.transData,
)
```

`add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`add_checker(checker)`

Add an entry to a dictionary of boolean flags that are set to True when the mappable is changed.

`aname = 'Artist'`

`autoscale()`

Autoscale the scalar limits on the norm instance using the current array

`autoscale_None()`

Autoscale the scalar limits on the norm instance using the current array, changing only

limits that are None

`axes`

The Axes instance the artist resides in, or None.

`changed()`

Call this whenever the mappable is changed to notify all the callbackSM listeners to the 'changed' signal

`check_update(checker)`

If mappable has changed since the last check, return True; else return False

`contains(mouseevent)`

Test whether the mouse event occurred in the collection.

Returns True | False, dict(ind=itemlist), where every item in itemlist contains the event.

`convert_xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`convert_yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`draw(renderer)`

Derived classes drawing method

`findobj(match=None, include_self=True)`

Find artist objects.

Recursively find all Artist instances contained in self.

match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is True (default), include self in the list to be checked for a match.

`format_cursor_data(data)`

Return cursor data string formatted.

`get_agg_filter()`

Return filter function to be used for agg filter.

`get_alpha()`

Return the alpha value used for blending - not supported on all backends

`get_animated()`

Return the artist's animated state

`get_array()`

Return the array

`get_capstyle()`

`get_children()`
Return a list of the child Artist's this :class:`Artist` contains.

`get_clim()`
return the min, max of the color limits for image scaling

`get_clip_box()`
Return artist clipbox

`get_clip_on()`
Return whether artist uses clipping

`get_clip_path()`
Return artist clip path

`get_cmap()`
return the colormap

`get_contains()`
Return the `_contains` test used by the artist, or `None` for default.

`get_cursor_data(event)`
Get the cursor data for a given event.

`get_dashes()`

`get_datalim(transData)`

`get_edgecolor()`

`get_edgecolors()`

`get_facecolor()`

`get_facecolors()`

`get_figure()`
Return the Figure instance the artist belongs to.

`get_fill()`
return whether fill is set

`get_gid()`
Returns the group id.

`get_hatch()`
Return the current hatching pattern.

`get_joinstyle()`

`get_label()`

Get the label used for this artist in the legend.

`get_linestyle()`

`get_linestyles()`

`get_linewidth()`

`get_linewidths()`

`get_numsides()`

`get_offset_position()`

Returns how offsets are applied for the collection. If `offset_position` is 'screen', the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`get_offset_transform()`

`get_offsets()`

Return the offsets for the collection.

`get_path_effects()`

`get_paths()`

`get_picker()`

Return the picker object used by this artist.

`get_pickradius()`

`get_rasterized()`

Return whether the artist is to be rasterized.

`get_rotation()`

`get_sizes()`

Returns the sizes of the elements in the collection. The value represents the 'area' of the element.

Returns

sizes [array] The 'area' of each element.

`get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

`sketch_params` [tuple or `None`]

A 3-tuple with the following elements:

- `scale`: The amplitude of the wiggle perpendicular to the source line.
- `length`: The length of the wiggle along the line.
- `randomness`: The scale factor by which the length is shrunk or expanded.

May return `'None'` if no sketch parameters were set.

`get_snap()`

Returns the snap setting which may be:

- `True`: snap vertices to the nearest pixel center
- `False`: leave vertices as-is
- `None`: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`get_transform()`

Return the Transform instance used by this artist.

`get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`get_transforms()`

`get_url()`

Returns the url.

`get_urls()`

`get_visible()`

Return the artist's visibility

`get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done

when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`get_zorder()`

Return the artist's zorder.

`have_units()`

Return True if units are set on the x or y axes

`hitlist(event)`

Deprecated since version 2.2: The hitlist function was deprecated in version 2.2.

List the children of the artist which contain the mouse event event.

`is_figure_set()`

Deprecated since version 2.2: `artist.figure` is not None

Returns whether the artist is assigned to a Figure.

`is_transform_set()`

Returns True if Artist has a transform explicitly set.

`mouseover`

`pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`pick(mouseevent)`

Process pick event

each child artist will fire a pick event if `mouseevent` is over the artist and the artist has picker set

`pickable()`

Return True if Artist is pickable.

`properties()`

return a dictionary mapping property name -> value for all Artist props

`remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`set(**kwargs)`

A property batch setter. Pass kwargs to set properties.

`set_agg_filter(filter_func)`

Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`set_alpha(alpha)`

Set the alpha transparencies of the collection. alpha must be a float or None.

ACCEPTS: float or None

`set_animated(b)`

Set the artist's animation state.

Parameters

`b` [bool]

`set_antialiased(aa)`

Set the antialiasing state for rendering.

ACCEPTS: Boolean or sequence of booleans

`set_antialiaseds(aa)`

alias for `set_antialiased`

`set_array(A)`

Set the image array from numpy array A.

Parameters

`A` [ndarray]

`set_capstyle(cs)`

Set the capstyle for the collection. The capstyle can only be set globally for all elements in the collection

Parameters

`cs` [['butt' | 'round' | 'projecting']] The capstyle

`set_clim(vmin=None, vmax=None)`

set the norm limits for image scaling; if `vmin` is a length2 sequence, interpret it as (`vmin`, `vmax`) which is used to support `setp`

ACCEPTS: a length 2 sequence of floats; may be overridden in methods that have `vmin` and `vmax` kwargs.

`set_clip_box(clipbox)`

Set the artist's clip Bbox.

Parameters

`clipbox` [Bbox]

`set_clip_on(b)`

Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

`b` [bool]

`set_clip_path(path, transform=None)`

Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

`set_cmap(cmap)`

set the colormap for luminance data

ACCEPTS: a colormap or registered colormap name

`set_color(c)`

Set both the edgecolor and the facecolor.

ACCEPTS: matplotlib color arg or sequence of rgba tuples

See also:

`set_facecolor()`, `set_edgecolor()` For setting the edge or face color individually.

`set_contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

`picker` [callable]

`set_dashes(ls)`

alias for `set_linestyle`

`set_edgecolor(c)`

Set the `edgecolor(s)` of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'face', the edge color will always be the same as the face color. If it is 'none', the patch boundary will not be drawn.

ACCEPTS: matplotlib color spec or sequence of specs

`set_edgecolors(c)`

alias for `set_edgecolor`

`set_facecolor(c)`

Set the `facecolor(s)` of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'none', the patch will not be filled.

ACCEPTS: matplotlib color spec or sequence of specs

`set_facecolors(c)`

alias for `set_facecolor`

`set_figure(fig)`

Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`set_gid(gid)`

Sets the (group) id for the artist.

Parameters

`gid` [str]

`set_hatch(hatch)`

Set the hatching pattern

`hatch` can be one of:

/	- diagonal hatching
\	- back diagonal
	- vertical
-	- horizontal
+	- crossed
x	- crossed diagonal
o	- small circle
O	- large circle
.	- dots
*	- stars

Letters can be combined, in which case all the specified hatchings are done. If same letter repeats, it increases the density of hatching of that pattern.

Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Unlike other properties such as linewidth and colors, hatching can only be specified for the collection as a whole, not separately for each member.

ACCEPTS: ['/' | '\ ' | '|' | '-' | '+' | 'x' | 'o' | 'O' | '?' | '*']

`set_joinstyle(js)`

Set the joinstyle for the collection. The joinstyle can only be set globally for all elements in the collection.

Parameters

`js` [['miter' | 'round' | 'bevel']] The joinstyle

`set_label(s)`

Set the label to `s` for auto legend.

Parameters

`s` [object] `s` will be converted to a string by calling `str` (unicode on Py2).

`set_linestyle(ls)`

Set the linestyle(`s`) for the collection.

linestyle	description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line

Alternatively a dash tuple of the following form can be provided:

(offset, onoffseq),

where onoffseq is an even length tuple of on and off ink in points.

ACCEPTS: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '--' | '-.' | ':' | 'None' | '' | '']

Parameters

`ls` [{ '-', '--', '-.', ':'} and more see description] The line style.

`set_linestyles(ls)`

alias for `set_linestyle`

`set_linewidth(lw)`

Set the linewidth(`s`) for the collection. `lw` can be a scalar or a sequence; if it is a sequence the patches will cycle through the sequence

ACCEPTS: float or sequence of floats

`set_linewidths(lw)`
alias for `set_linewidth`

`set_lw(lw)`
alias for `set_linewidth`

`set_norm(norm)`
Set the normalization instance.

Parameters

`norm` [Normalize]

`set_offset_position(offset_position)`
Set how offsets are applied. If `offset_position` is 'screen' (default) the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`set_offsets(offsets)`
Set the offsets for the collection. `offsets` can be a scalar or a sequence.

ACCEPTS: float or sequence of floats

`set_path_effects(path_effects)`
Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`set_paths()`

`set_picker(picker)`
Set the epsilon for picking used by this artist

`picker` can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if `picker` is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if `picker` is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

`hit, props = picker(artist, mouseevent)`

to determine the hit test. if the mouse event is over the artist, return `hit=True` and `props` is a dictionary of properties you want added to the `PickEvent` attributes.

Parameters

`picker` [None or bool or float or callable]

`set_pickradius(pr)`

Set the pick radius used for containment tests.

Parameters

`d` [float] Pick radius, in points.

`set_rasterized(rasterized)`

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

`rasterized` [bool or None]

`set_sizes(sizes, dpi=72.0)`

Set the sizes of each member of the collection.

Parameters

`sizes` [ndarray or None] The size to set for each element of the collection.
The value is the 'area' of the element.

`dpi` [float] The dpi of the canvas. Defaults to 72.0.

`set_sketch_params(scale=None, length=None, randomness=None)`

Sets the sketch parameters.

Parameters

`scale` [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If `scale` is `None`, or not provided, no sketch filter will be provided.

`length` [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

`randomness` [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

`set_snap(snap)`

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

 snap [bool or None]

set_transform(t)

Set the artist transform.

Parameters

 t [Transform]

set_url(url)

Sets the url for the artist.

Parameters

 url [str]

set_urls(urls)

Parameters

 urls [List[str] or None]

set_visible(b)

Set the artist's visibility.

Parameters

 b [bool]

set_zorder(level)

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

 level [float]

stale

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

sticky_edges

x and y sticky edge lists.

When performing autoscaling, if a data limit coincides with a value in the corresponding sticky_edges list, then no margin will be added—the view limit "sticks" to the edge. A typical usecase is histograms, where one usually expects no margin on the bottom edge (0) of the histogram.

This attribute cannot be assigned to; however, the x and y lists can be modified in place as needed.

Examples

```
>>> artist.sticky_edges.x[:] = (xmin, xmax)
>>> artist.sticky_edges.y[:] = (ymin, ymax)
```

`to_rgba(x, alpha=None, bytes=False, norm=True)`

Return a normalized rgba array corresponding to x.

In the normal case, x is a 1-D or 2-D sequence of scalars, and the corresponding ndarray of rgba values will be returned, based on the norm and colormap set for this ScalarMappable.

There is one special case, for handling images that are already rgb or rgba, such as might have been read from an image file. If x is an ndarray with 3 dimensions, and the last dimension is either 3 or 4, then it will be treated as an rgb or rgba array, and no mapping will be done. The array can be uint8, or it can be floating point with values in the 0-1 range; otherwise a `ValueError` will be raised. If it is a masked array, the mask will be ignored. If the last dimension is 3, the alpha kwarg (defaulting to 1) will be used to fill in the transparency. If the last dimension is 4, the alpha kwarg is ignored; it does not replace the pre-existing alpha. A `ValueError` will be raised if the third dimension is other than 3 or 4.

In either case, if bytes is `False` (default), the rgba array will be floats in the 0-1 range; if it is `True`, the returned rgba array will be uint8 in the 0 to 255 range.

If norm is `False`, no normalization of the input data is performed, and it is assumed to be in the range (0-1).

`update(props)`

Update this artist's properties from the dictionary prop.

`update_from(other)`

copy properties from other to self

`update_scalarmappable()`

If the scalar mappable array is not none, update colors from scalar data

`zorder = 0`

`class matplotlib.collections.BrokenBarHCollection(xranges, yrange, **kwargs)`

Bases: `matplotlib.collections.PolyCollection`

A collection of horizontal bars spanning yrange with a sequence of xrange.

xranges sequence of (xmin, xwidth)

yrange ymin, ywidth

Valid Collection keyword arguments:

- edgecolors: None
- facecolors: None

- linewidths: None
- antialiaseds: None
- offsets: None
- transOffset: transforms.IdentityTransform()
- norm: None (optional for matplotlib.cm.ScalarMappable)
- cmap: None (optional for matplotlib.cm.ScalarMappable)

offsets and transOffset are used to translate the patch after rendering (default no offsets)

If any of edgcolors, facecolors, linewidths, antialiaseds are None, they default to their matplotlib.rcParams patch setting, in sequence form.

`add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`add_checker(checker)`

Add an entry to a dictionary of boolean flags that are set to True when the mappable is changed.

`aname = 'Artist'`

`autoscale()`

Autoscale the scalar limits on the norm instance using the current array

`autoscale_None()`

Autoscale the scalar limits on the norm instance using the current array, changing only limits that are None

`axes`

The Axes instance the artist resides in, or None.

`changed()`

Call this whenever the mappable is changed to notify all the callbackSM listeners to the 'changed' signal

`check_update(checker)`

If mappable has changed since the last check, return True; else return False

`contains(mouseevent)`

Test whether the mouse event occurred in the collection.

Returns True | False, dict(ind=itemlist), where every item in itemlist contains the event.

`convert_xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`convert_yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`draw(renderer)`
Derived classes drawing method

`findobj(match=None, include_self=True)`
Find artist objects.
Recursively find all Artist instances contained in self.
match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is `True` (default), include self in the list to be checked for a match.

`format_cursor_data(data)`
Return cursor data string formatted.

`get_agg_filter()`
Return filter function to be used for agg filter.

`get_alpha()`
Return the alpha value used for blending - not supported on all backends

`get_animated()`
Return the artist's animated state

`get_array()`
Return the array

`get_capstyle()`

`get_children()`
Return a list of the child Artist's this :class:`Artist` contains.

`get_clim()`
return the min, max of the color limits for image scaling

`get_clip_box()`
Return artist clipbox

`get_clip_on()`
Return whether artist uses clipping

`get_clip_path()`
Return artist clip path

`get_cmap()`
return the colormap

`get_contains()`
Return the `_contains` test used by the artist, or `None` for default.

`get_cursor_data(event)`

Get the cursor data for a given event.

`get_dashes()`

`get_datalim(transData)`

`get_edgecolor()`

`get_edgecolors()`

`get_facecolor()`

`get_facecolors()`

`get_figure()`

Return the Figure instance the artist belongs to.

`get_fill()`

return whether fill is set

`get_gid()`

Returns the group id.

`get_hatch()`

Return the current hatching pattern.

`get_joinstyle()`

`get_label()`

Get the label used for this artist in the legend.

`get_linestyle()`

`get_linestyles()`

`get_linewidth()`

`get_linewidths()`

`get_offset_position()`

Returns how offsets are applied for the collection. If `offset_position` is 'screen', the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`get_offset_transform()`

`get_offsets()`

Return the offsets for the collection.

`get_path_effects()`

`get_paths()`

`get_picker()`

Return the picker object used by this artist.

`get_pickradius()`

`get_rasterized()`

Return whether the artist is to be rasterized.

`get_sizes()`

Returns the sizes of the elements in the collection. The value represents the 'area' of the element.

Returns

sizes [array] The 'area' of each element.

`get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

sketch_params [tuple or `None`]

A 3-tuple with the following elements:

- scale: The amplitude of the wiggle perpendicular to the source line.
- length: The length of the wiggle along the line.
- randomness: The scale factor by which the length is shrunk or expanded.

May return 'None' if no sketch parameters were set.

`get_snap()`

Returns the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`get_transform()`
Return the Transform instance used by this artist.

`get_transformed_clip_path_and_affine()`
Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`get_transforms()`

`get_url()`
Returns the url.

`get_urls()`

`get_visible()`
Return the artist's visibility

`get_window_extent(renderer)`
Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`get_zorder()`
Return the artist's zorder.

`have_units()`
Return True if units are set on the x or y axes

`hitlist(event)`
Deprecated since version 2.2: The hitlist function was deprecated in version 2.2.

List the children of the artist which contain the mouse event event.

`is_figure_set()`
Deprecated since version 2.2: `artist.figure` is not None

Returns whether the artist is assigned to a Figure.

`is_transform_set()`
Returns True if Artist has a transform explicitly set.

`mouseover`

`pchanged()`
Fire an event when property changed, calling all of the registered callbacks.

`pick(mouseevent)`
Process pick event

each child artist will fire a pick event if `mouseevent` is over the artist and the artist has picker set

`pickable()`
Return True if Artist is pickable.

`properties()`
return a dictionary mapping property name -> value for all Artist props

`remove()`
Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`remove_callback(oid)`
Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`set(**kwargs)`
A property batch setter. Pass kwargs to set properties.

`set_agg_filter(filter_func)`
Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`set_alpha(alpha)`
Set the alpha transparencies of the collection. `alpha` must be a float or None.

ACCEPTS: float or None

`set_animated(b)`
Set the artist's animation state.

Parameters

`b` [bool]

`set_antialiased(aa)`
Set the antialiasing state for rendering.

ACCEPTS: Boolean or sequence of booleans

`set_antialiaseds(aa)`
alias for `set_antialiased`

`set_array(A)`
Set the image array from numpy array A.

Parameters

A [ndarray]

`set_capstyle(cs)`
Set the capstyle for the collection. The capstyle can only be set globally for all elements in the collection

Parameters

cs [['butt' | 'round' | 'projecting']] The capstyle

`set_clim(vmin=None, vmax=None)`
set the norm limits for image scaling; if vmin is a length2 sequence, interpret it as (vmin, vmax) which is used to support setp

ACCEPTS: a length 2 sequence of floats; may be overridden in methods that have vmin and vmax kwargs.

`set_clip_box(clipbox)`
Set the artist's clip Bbox.

Parameters

clipbox [Bbox]

`set_clip_on(b)`
Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

b [bool]

`set_clip_path(path, transform=None)`
Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

`set_cmap(cmap)`

set the colormap for luminance data

ACCEPTS: a colormap or registered colormap name

`set_color(c)`

Set both the edgecolor and the facecolor.

ACCEPTS: matplotlib color arg or sequence of rgba tuples

See also:

`set_facecolor()`, `set_edgecolor()` For setting the edge or face color individually.

`set_contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

`picker` [callable]

`set_dashes(ls)`

alias for `set_linestyle`

`set_edgecolor(c)`

Set the edgecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'face', the edge color will always be the same as the face color. If it is 'none', the patch boundary will not be drawn.

ACCEPTS: matplotlib color spec or sequence of specs

`set_edgecolors(c)`

alias for `set_edgecolor`

`set_facecolor(c)`

Set the facecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'none', the patch will not be filled.

ACCEPTS: matplotlib color spec or sequence of specs

`set_facecolors(c)`

alias for `set_facecolor`

`set_figure(fig)`
Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`set_gid(gid)`
Sets the (group) id for the artist.

Parameters

`gid` [str]

`set_hatch(hatch)`
Set the hatching pattern

`hatch` can be one of:

```
/ - diagonal hatching
\ - back diagonal
| - vertical
- - horizontal
+ - crossed
x - crossed diagonal
o - small circle
O - large circle
. - dots
* - stars
```

Letters can be combined, in which case all the specified hatchings are done. If same letter repeats, it increases the density of hatching of that pattern.

Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Unlike other properties such as linewidth and colors, hatching can only be specified for the collection as a whole, not separately for each member.

ACCEPTS: ['/' | '\ ' | '|' | '- ' | '+ ' | 'x' | 'o' | 'O' | '?' | '*']

`set_joinstyle(js)`
Set the joinstyle for the collection. The joinstyle can only be set globally for all elements in the collection.

Parameters

`js` [['miter' | 'round' | 'bevel']] The joinstyle

`set_label(s)`
Set the label to `s` for auto legend.

Parameters

`s` [object] `s` will be converted to a string by calling `str` (unicode on Py2).

`set_linestyle(ls)`

Set the linestyle(s) for the collection.

linestyle	description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line

Alternatively a dash tuple of the following form can be provided:

`(offset, onoffseq),`

where onoffseq is an even length tuple of on and off ink in points.

ACCEPTS: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '--' |
'-.' | ':' | 'None' | ' ' | '']

Parameters

ls [{ '-', '--', '-.', ':'}] and more see description] The line style.

`set_linestyles(ls)`

alias for `set_linestyle`

`set_linewidth(lw)`

Set the linewidth(s) for the collection. lw can be a scalar or a sequence; if it is a sequence the patches will cycle through the sequence

ACCEPTS: float or sequence of floats

`set_linewidths(lw)`

alias for `set_linewidth`

`set_lw(lw)`

alias for `set_linewidth`

`set_norm(norm)`

Set the normalization instance.

Parameters

norm [Normalize]

`set_offset_position(offset_position)`

Set how offsets are applied. If `offset_position` is 'screen' (default) the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`set_offsets(offsets)`

Set the offsets for the collection. offsets can be a scalar or a sequence.

ACCEPTS: float or sequence of floats

`set_path_effects(path_effects)`

Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`set_paths(verts, closed=True)`

This allows one to delay initialization of the vertices.

`set_picker(picker)`

Set the epsilon for picking used by this artist

picker can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if picker is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if picker is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

`hit, props = picker(artist, mouseevent)`

to determine the hit test. if the mouse event is over the artist, return `hit=True` and `props` is a dictionary of properties you want added to the `PickEvent` attributes.

Parameters

`picker` [None or bool or float or callable]

`set_pickradius(pr)`

Set the pick radius used for containment tests.

Parameters

`d` [float] Pick radius, in points.

`set_rasterized(rasterized)`

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

`rasterized` [bool or None]

`set_sizes(sizes, dpi=72.0)`

Set the sizes of each member of the collection.

Parameters

`sizes` [ndarray or None] The size to set for each element of the collection.
The value is the 'area' of the element.

`dpi` [float] The dpi of the canvas. Defaults to 72.0.

`set_sketch_params(scale=None, length=None, randomness=None)`

Sets the sketch parameters.

Parameters

`scale` [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If scale is `None`, or not provided, no sketch filter will be provided.

`length` [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

`randomness` [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

`set_snap(snap)`

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

`snap` [bool or None]

`set_transform(t)`

Set the artist transform.

Parameters

`t` [Transform]

`set_url(url)`

Sets the url for the artist.

Parameters

`url` [str]

`set_urls(urls)`

Parameters

urls [List[str] or None]

set_verts(verts, closed=True)

This allows one to delay initialization of the vertices.

set_verts_and_codes(verts, codes)

This allows one to initialize vertices with path codes.

set_visible(b)

Set the artist's visibility.

Parameters

b [bool]

set_zorder(level)

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

level [float]

static span_where(x, ymin, ymax, where, **kwargs)

Create a BrokenBarHCollection to plot horizontal bars from over the regions in x where where is True. The bars range on the y-axis from ymin to ymax

A BrokenBarHCollection is returned. kwargs are passed on to the collection.

stale

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

sticky_edges

x and y sticky edge lists.

When performing autoscaling, if a data limit coincides with a value in the corresponding sticky_edges list, then no margin will be added—the view limit "sticks" to the edge. A typical usecase is histograms, where one usually expects no margin on the bottom edge (0) of the histogram.

This attribute cannot be assigned to; however, the x and y lists can be modified in place as needed.

Examples

```
>>> artist.sticky_edges.x[:] = (xmin, xmax)
>>> artist.sticky_edges.y[:] = (ymin, ymax)
```

to_rgba(x, alpha=None, bytes=False, norm=True)

Return a normalized rgba array corresponding to x.

In the normal case, x is a 1-D or 2-D sequence of scalars, and the corresponding ndarray of rgba values will be returned, based on the norm and colormap set for this ScalarMappable.

There is one special case, for handling images that are already rgb or rgba, such as might have been read from an image file. If `x` is an ndarray with 3 dimensions, and the last dimension is either 3 or 4, then it will be treated as an rgb or rgba array, and no mapping will be done. The array can be uint8, or it can be floating point with values in the 0-1 range; otherwise a `ValueError` will be raised. If it is a masked array, the mask will be ignored. If the last dimension is 3, the `alpha` kwarg (defaulting to 1) will be used to fill in the transparency. If the last dimension is 4, the `alpha` kwarg is ignored; it does not replace the pre-existing alpha. A `ValueError` will be raised if the third dimension is other than 3 or 4.

In either case, if `bytes` is `False` (default), the rgba array will be floats in the 0-1 range; if it is `True`, the returned rgba array will be uint8 in the 0 to 255 range.

If `norm` is `False`, no normalization of the input data is performed, and it is assumed to be in the range (0-1).

`update(props)`

Update this artist's properties from the dictionary `prop`.

`update_from(other)`

copy properties from `other` to self

`update_scalarmappable()`

If the scalar mappable array is not none, update colors from scalar data

`zorder = 0`

`class matplotlib.collections.CircleCollection(sizes, **kwargs)`

Bases: `matplotlib.collections._CollectionWithSizes`

A collection of circles, drawn using splines.

`sizes` Gives the area of the circle in points^2

Valid Collection keyword arguments:

- `edgecolors`: None
- `facecolors`: None
- `linewidths`: None
- `antialiaseds`: None
- `offsets`: None
- `transOffset`: `transforms.IdentityTransform()`
- `norm`: None (optional for `matplotlib.cm.ScalarMappable`)
- `cmap`: None (optional for `matplotlib.cm.ScalarMappable`)

`offsets` and `transOffset` are used to translate the patch after rendering (default no offsets)

If any of `edgecolors`, `facecolors`, `linewidths`, `antialiaseds` are None, they default to their `matplotlib.rcParams` patch setting, in sequence form.

`add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`add_checker(checker)`

Add an entry to a dictionary of boolean flags that are set to True when the mappable is changed.

`aname = 'Artist'`

`autoscale()`

Autoscale the scalar limits on the norm instance using the current array

`autoscale_None()`

Autoscale the scalar limits on the norm instance using the current array, changing only limits that are None

`axes`

The Axes instance the artist resides in, or None.

`changed()`

Call this whenever the mappable is changed to notify all the callbackSM listeners to the 'changed' signal

`check_update(checker)`

If mappable has changed since the last check, return True; else return False

`contains(mouseevent)`

Test whether the mouse event occurred in the collection.

Returns True | False, dict(ind=itemlist), where every item in itemlist contains the event.

`convert_xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`convert_yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`draw(renderer)`

Derived classes drawing method

`findobj(match=None, include_self=True)`

Find artist objects.

Recursively find all Artist instances contained in self.

match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is True (default), include self in the list to be checked for a match.

`format_cursor_data(data)`
Return cursor data string formatted.

`get_agg_filter()`
Return filter function to be used for agg filter.

`get_alpha()`
Return the alpha value used for blending - not supported on all backends

`get_animated()`
Return the artist's animated state

`get_array()`
Return the array

`get_capstyle()`

`get_children()`
Return a list of the child Artist's this :class:`Artist` contains.

`get_clim()`
return the min, max of the color limits for image scaling

`get_clip_box()`
Return artist clipbox

`get_clip_on()`
Return whether artist uses clipping

`get_clip_path()`
Return artist clip path

`get_cmap()`
return the colormap

`get_contains()`
Return the `__contains` test used by the artist, or `None` for default.

`get_cursor_data(event)`
Get the cursor data for a given event.

`get_dashes()`

`get_datalim(transData)`

`get_edgecolor()`

`get_edgecolors()`

`get_facecolor()`

`get_facecolors()`

`get_figure()`

Return the Figure instance the artist belongs to.

`get_fill()`

return whether fill is set

`get_gid()`

Returns the group id.

`get_hatch()`

Return the current hatching pattern.

`get_joinstyle()`

`get_label()`

Get the label used for this artist in the legend.

`get_linestyle()`

`get_linestyles()`

`get_linewidth()`

`get_linewidths()`

`get_offset_position()`

Returns how offsets are applied for the collection. If `offset_position` is 'screen', the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`get_offset_transform()`

`get_offsets()`

Return the offsets for the collection.

`get_path_effects()`

`get_paths()`

`get_picker()`

Return the picker object used by this artist.

`get_pickradius()`

`get_rasterized()`

Return whether the artist is to be rasterized.

`get_sizes()`

Returns the sizes of the elements in the collection. The value represents the 'area' of the element.

Returns

sizes [array] The 'area' of each element.

`get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

sketch_params [tuple or `None`]

A 3-tuple with the following elements:

- scale: The amplitude of the wiggle perpendicular to the source line.
- length: The length of the wiggle along the line.
- randomness: The scale factor by which the length is shrunk or expanded.

May return 'None' if no sketch parameters were set.

`get_snap()`

Returns the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`get_transform()`

Return the Transform instance used by this artist.

`get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`get_transforms()`

`get_url()`

Returns the url.

`get_urls()`

`get_visible()`

Return the artist's visibility

`get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`get_zorder()`

Return the artist's zorder.

`have_units()`

Return True if units are set on the x or y axes

`hitlist(event)`

Deprecated since version 2.2: The hitlist function was deprecated in version 2.2.

List the children of the artist which contain the mouse event event.

`is_figure_set()`

Deprecated since version 2.2: `artist.figure` is not None

Returns whether the artist is assigned to a Figure.

`is_transform_set()`

Returns True if Artist has a transform explicitly set.

`mouseover`

`pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`pick(mouseevent)`

Process pick event

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

`pickable()`

Return True if Artist is pickable.

`properties()`

return a dictionary mapping property name -> value for all Artist props

`remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`set(**kwargs)`

A property batch setter. Pass kwargs to set properties.

`set_agg_filter(filter_func)`

Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`set_alpha(alpha)`

Set the alpha transparencies of the collection. alpha must be a float or None.

ACCEPTS: float or None

`set_animated(b)`

Set the artist's animation state.

Parameters

`b` [bool]

`set_antialiased(aa)`

Set the antialiasing state for rendering.

ACCEPTS: Boolean or sequence of booleans

`set_antialiaseds(aa)`

alias for `set_antialiased`

`set_array(A)`

Set the image array from numpy array A.

Parameters

`A` [ndarray]

`set_capstyle(cs)`

Set the capstyle for the collection. The capstyle can only be set globally for all elements in the collection

Parameters

`cs` [['butt' | 'round' | 'projecting']] The capstyle

`set_clim(vmin=None, vmax=None)`

set the norm limits for image scaling; if `vmin` is a length2 sequence, interpret it as (`vmin`, `vmax`) which is used to support `setp`

ACCEPTS: a length 2 sequence of floats; may be overridden in methods that have vmin and vmax kwargs.

`set_clip_box(clipbox)`

Set the artist's clip Bbox.

Parameters

`clipbox` [Bbox]

`set_clip_on(b)`

Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

`b` [bool]

`set_clip_path(path, transform=None)`

Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

`set_cmap(cmap)`

set the colormap for luminance data

ACCEPTS: a colormap or registered colormap name

`set_color(c)`

Set both the edgecolor and the facecolor.

ACCEPTS: matplotlib color arg or sequence of rgba tuples

See also:

`set_facecolor()`, `set_edgecolor()` For setting the edge or face color individually.

`set_contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

picker [callable]

set__dashes(ls)

alias for set__linestyle

set__edgecolor(c)

Set the edgecolor(s) of the collection. c can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If c is 'face', the edge color will always be the same as the face color. If it is 'none', the patch boundary will not be drawn.

ACCEPTS: matplotlib color spec or sequence of specs

set__edgecolors(c)

alias for set__edgecolor

set__facecolor(c)

Set the facecolor(s) of the collection. c can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If c is 'none', the patch will not be filled.

ACCEPTS: matplotlib color spec or sequence of specs

set__facecolors(c)

alias for set__facecolor

set__figure(fig)

Set the Figure instance the artist belongs to.

Parameters

fig [Figure]

set__gid(gid)

Sets the (group) id for the artist.

Parameters

gid [str]

set__hatch(hatch)

Set the hatching pattern

hatch can be one of:

/	- diagonal hatching
\	- back diagonal
	- vertical
-	- horizontal
+	- crossed

(continues on next page)

(continued from previous page)

x	-	crossed diagonal
o	-	small circle
O	-	large circle
.	-	dots
*	-	stars

Letters can be combined, in which case all the specified hatchings are done. If same letter repeats, it increases the density of hatching of that pattern.

Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Unlike other properties such as linewidth and colors, hatching can only be specified for the collection as a whole, not separately for each member.

ACCEPTS: ['/' | '\ ' | ' ' | ' ' | ' ' | ' ' | 'x' | 'o' | 'O' | '.' | '*']

`set_joinstyle(js)`
Set the joinstyle for the collection. The joinstyle can only be set globally for all elements in the collection.

Parameters

js [['miter' | 'round' | 'bevel']] The joinstyle

`set_label(s)`
Set the label to s for auto legend.

Parameters

s [object] s will be converted to a string by calling `str` (unicode on Py2).

`set_linestyle(ls)`
Set the linestyle(s) for the collection.

linestyle	description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line

Alternatively a dash tuple of the following form can be provided:

(offset, onoffseq),

where onoffseq is an even length tuple of on and off ink in points.

ACCEPTS: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '--' | '-.' | ':' | 'None' | ' ' | '']

Parameters

ls [{ ' ', ' ', ' ', ' ' } and more see description] The line style.

`set_linestyles(ls)`
 alias for `set_linestyle`

`set_linewidth(lw)`
 Set the linewidth(s) for the collection. `lw` can be a scalar or a sequence; if it is a sequence the patches will cycle through the sequence
 ACCEPTS: float or sequence of floats

`set_linewidths(lw)`
 alias for `set_linewidth`

`set_lw(lw)`
 alias for `set_linewidth`

`set_norm(norm)`
 Set the normalization instance.

Parameters

`norm` [Normalize]

`set_offset_position(offset_position)`
 Set how offsets are applied. If `offset_position` is 'screen' (default) the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`set_offsets(offsets)`
 Set the offsets for the collection. `offsets` can be a scalar or a sequence.

ACCEPTS: float or sequence of floats

`set_path_effects(path_effects)`
 Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`set_paths()`

`set_picker(picker)`
 Set the epsilon for picking used by this artist

`picker` can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if `picker` is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data

to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event

- A function: if picker is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

to determine the hit test. if the mouse event is over the artist, return hit=True and props is a dictionary of properties you want added to the PickEvent attributes.

Parameters

picker [None or bool or float or callable]

set_pickradius(pr)

Set the pick radius used for containment tests.

Parameters

d [float] Pick radius, in points.

set_rasterized(rasterized)

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

rasterized [bool or None]

set_sizes(sizes, dpi=72.0)

Set the sizes of each member of the collection.

Parameters

sizes [ndarray or None] The size to set for each element of the collection.
The value is the 'area' of the element.

dpi [float] The dpi of the canvas. Defaults to 72.0.

set_sketch_params(scale=None, length=None, randomness=None)

Sets the sketch parameters.

Parameters

scale [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If scale is `None`, or not provided, no sketch filter will be provided.

length [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

randomness [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

`set_snap(snap)`

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

`snap` [bool or None]

`set_transform(t)`

Set the artist transform.

Parameters

`t` [Transform]

`set_url(url)`

Sets the url for the artist.

Parameters

`url` [str]

`set_urls(urls)`

Parameters

`urls` [List[str] or None]

`set_visible(b)`

Set the artist's visibility.

Parameters

`b` [bool]

`set_zorder(level)`

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

`level` [float]

`stale`

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

`sticky_edges`

x and y sticky edge lists.

When performing autoscaling, if a data limit coincides with a value in the corresponding `sticky_edges` list, then no margin will be added—the view limit "sticks" to the edge. A

typical usecase is histograms, where one usually expects no margin on the bottom edge (0) of the histogram.

This attribute cannot be assigned to; however, the x and y lists can be modified in place as needed.

Examples

```
>>> artist.sticky_edges.x[:] = (xmin, xmax)
>>> artist.sticky_edges.y[:] = (ymin, ymax)
```

`to_rgba(x, alpha=None, bytes=False, norm=True)`

Return a normalized rgba array corresponding to x.

In the normal case, x is a 1-D or 2-D sequence of scalars, and the corresponding ndarray of rgba values will be returned, based on the norm and colormap set for this ScalarMappable.

There is one special case, for handling images that are already rgb or rgba, such as might have been read from an image file. If x is an ndarray with 3 dimensions, and the last dimension is either 3 or 4, then it will be treated as an rgb or rgba array, and no mapping will be done. The array can be uint8, or it can be floating point with values in the 0-1 range; otherwise a `ValueError` will be raised. If it is a masked array, the mask will be ignored. If the last dimension is 3, the alpha kwarg (defaulting to 1) will be used to fill in the transparency. If the last dimension is 4, the alpha kwarg is ignored; it does not replace the pre-existing alpha. A `ValueError` will be raised if the third dimension is other than 3 or 4.

In either case, if bytes is `False` (default), the rgba array will be floats in the 0-1 range; if it is `True`, the returned rgba array will be uint8 in the 0 to 255 range.

If norm is `False`, no normalization of the input data is performed, and it is assumed to be in the range (0-1).

`update(props)`

Update this artist's properties from the dictionary prop.

`update_from(other)`

copy properties from other to self

`update_scalarmappable()`

If the scalar mappable array is not none, update colors from scalar data

`zorder = 0`

```
class matplotlib.collections.Collection(edgecolors=None, facecolors=None, linewidths=None,
                                       linestyle='solid', capstyle=None, joinstyle=None,
                                       antialiaseds=None, offsets=None, transOffset=None, norm=None, cmap=None, pickradius=5.0,
                                       hatch=None, urls=None, offset_position='screen',
                                       zorder=1, **kwargs)
```

Bases: matplotlib.artist.Artist, matplotlib.cm.ScalarMappable

Base class for Collections. Must be subclassed to be usable.

All properties in a collection must be sequences or scalars; if scalars, they will be converted to sequences. The property of the *i*th element of the collection is:

```
prop[i % len(props)]
```

Exceptions are capstyle and joinstyle properties, these can only be set globally for the whole collection.

Keyword arguments and default values:

- edgecolors: None
- facecolors: None
- linewidths: None
- capstyle: None
- joinstyle: None
- antialiaseds: None
- offsets: None
- transOffset: transforms.IdentityTransform()
- offset_position: 'screen' (default) or 'data'
- norm: None (optional for matplotlib.cm.ScalarMappable)
- cmap: None (optional for matplotlib.cm.ScalarMappable)
- hatch: None
- zorder: 1

offsets and transOffset are used to translate the patch after rendering (default no offsets). If offset_position is 'screen' (default) the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If offset_position is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

If any of edgecolors, facecolors, linewidths, antialiaseds are None, they default to their matplotlib.rcParams patch setting, in sequence form.

The use of ScalarMappable is optional. If the ScalarMappable matrix `_A` is not None (i.e., a call to `set_array` has been made), at draw time a call to scalar mappable will be made to set the face colors.

Create a Collection

```
%(Collection)s
```

```
add_callback(func)
```

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`add_checker(checker)`
Add an entry to a dictionary of boolean flags that are set to True when the mappable is changed.

`aname = 'Artist'`

`autoscale()`
Autoscale the scalar limits on the norm instance using the current array

`autoscale_None()`
Autoscale the scalar limits on the norm instance using the current array, changing only limits that are None

`axes`
The Axes instance the artist resides in, or None.

`changed()`
Call this whenever the mappable is changed to notify all the callbackSM listeners to the 'changed' signal

`check_update(checker)`
If mappable has changed since the last check, return True; else return False

`contains(mouseevent)`
Test whether the mouse event occurred in the collection.

Returns True | False, dict(ind=itemlist), where every item in itemlist contains the event.

`convert_xunits(x)`
For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`convert_yunits(y)`
For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`draw(renderer)`
Derived classes drawing method

`findobj(match=None, include_self=True)`
Find artist objects.

Recursively find all Artist instances contained in self.

`match` can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is True (default), include self in the list to be checked for a match.

`format_cursor_data(data)`
Return cursor data string formatted.

`get_agg_filter()`
Return filter function to be used for agg filter.

`get_alpha()`
Return the alpha value used for blending - not supported on all backends

`get_animated()`
Return the artist's animated state

`get_array()`
Return the array

`get_capstyle()`

`get_children()`
Return a list of the child Artist's this :class:`Artist` contains.

`get_clim()`
return the min, max of the color limits for image scaling

`get_clip_box()`
Return artist clipbox

`get_clip_on()`
Return whether artist uses clipping

`get_clip_path()`
Return artist clip path

`get_cmap()`
return the colormap

`get_contains()`
Return the `__contains` test used by the artist, or `None` for default.

`get_cursor_data(event)`
Get the cursor data for a given event.

`get_dashes()`

`get_datalim(transData)`

`get_edgecolor()`

`get_edgecolors()`

`get_facecolor()`

`get_facecolors()`

`get_figure()`
Return the Figure instance the artist belongs to.

`get_fill()`
return whether fill is set

`get_gid()`
Returns the group id.

`get_hatch()`
Return the current hatching pattern.

`get_joinstyle()`

`get_label()`
Get the label used for this artist in the legend.

`get_linestyle()`

`get_linestyles()`

`get_linewidth()`

`get_linewidths()`

`get_offset_position()`
Returns how offsets are applied for the collection. If `offset_position` is 'screen', the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`get_offset_transform()`

`get_offsets()`
Return the offsets for the collection.

`get_path_effects()`

`get_paths()`

`get_picker()`
Return the picker object used by this artist.

`get_pickradius()`

`get_rasterized()`
Return whether the artist is to be rasterized.

`get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

`sketch_params` [tuple or `None`]

A 3-tuple with the following elements:

- `scale`: The amplitude of the wiggle perpendicular to the source line.
- `length`: The length of the wiggle along the line.
- `randomness`: The scale factor by which the length is shrunk or expanded.

May return `'None'` if no sketch parameters were set.

`get_snap()`

Returns the snap setting which may be:

- `True`: snap vertices to the nearest pixel center
- `False`: leave vertices as-is
- `None`: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`get_transform()`

Return the Transform instance used by this artist.

`get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`get_transforms()`

`get_url()`

Returns the url.

`get_urls()`

`get_visible()`

Return the artist's visibility

`get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done

when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`get_zorder()`

Return the artist's zorder.

`have_units()`

Return True if units are set on the x or y axes

`hitlist(event)`

Deprecated since version 2.2: The hitlist function was deprecated in version 2.2.

List the children of the artist which contain the mouse event event.

`is_figure_set()`

Deprecated since version 2.2: `artist.figure` is not None

Returns whether the artist is assigned to a Figure.

`is_transform_set()`

Returns True if Artist has a transform explicitly set.

`mouseover`

`pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`pick(mouseevent)`

Process pick event

each child artist will fire a pick event if `mouseevent` is over the artist and the artist has picker set

`pickable()`

Return True if Artist is pickable.

`properties()`

return a dictionary mapping property name -> value for all Artist props

`remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`set(**kwargs)`
 A property batch setter. Pass kwargs to set properties.

`set_agg_filter(filter_func)`
 Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`set_alpha(alpha)`
 Set the alpha transparencies of the collection. alpha must be a float or None.

ACCEPTS: float or None

`set_animated(b)`
 Set the artist's animation state.

Parameters

`b` [bool]

`set_antialiased(aa)`
 Set the antialiasing state for rendering.

ACCEPTS: Boolean or sequence of booleans

`set_antialiaseds(aa)`
 alias for `set_antialiased`

`set_array(A)`
 Set the image array from numpy array A.

Parameters

`A` [ndarray]

`set_capstyle(cs)`
 Set the capstyle for the collection. The capstyle can only be set globally for all elements in the collection

Parameters

`cs` [['butt' | 'round' | 'projecting']] The capstyle

`set_clim(vmin=None, vmax=None)`
 set the norm limits for image scaling; if `vmin` is a length2 sequence, interpret it as (`vmin`, `vmax`) which is used to support `setp`

ACCEPTS: a length 2 sequence of floats; may be overridden in methods that have `vmin` and `vmax` kwargs.

`set_clip_box(clipbox)`
 Set the artist's clip Bbox.

Parameters

`clipbox` [Bbox]

`set_clip_on(b)`

Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

b [bool]

`set_clip_path(path, transform=None)`

Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

`set_cmap(cmap)`

set the colormap for luminance data

ACCEPTS: a colormap or registered colormap name

`set_color(c)`

Set both the edgecolor and the facecolor.

ACCEPTS: matplotlib color arg or sequence of rgba tuples

See also:

`set_facecolor()`, `set_edgecolor()` For setting the edge or face color individually.

`set_contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

picker [callable]

`set_dashes(ls)`

alias for `set_linestyle`

`set_edgecolor(c)`

Set the `edgecolor(s)` of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'face', the edge color will always be the same as the face color. If it is 'none', the patch boundary will not be drawn.

ACCEPTS: matplotlib color spec or sequence of specs

`set_edgecolors(c)`

alias for `set_edgecolor`

`set_facecolor(c)`

Set the `facecolor(s)` of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'none', the patch will not be filled.

ACCEPTS: matplotlib color spec or sequence of specs

`set_facecolors(c)`

alias for `set_facecolor`

`set_figure(fig)`

Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`set_gid(gid)`

Sets the (group) id for the artist.

Parameters

`gid` [str]

`set_hatch(hatch)`

Set the hatching pattern

`hatch` can be one of:

```
/ - diagonal hatching
\ - back diagonal
| - vertical
- - horizontal
+ - crossed
x - crossed diagonal
o - small circle
O - large circle
. - dots
* - stars
```

Letters can be combined, in which case all the specified hatchings are done. If same letter repeats, it increases the density of hatching of that pattern.

Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Unlike other properties such as linewidth and colors, hatching can only be specified for the collection as a whole, not separately for each member.

ACCEPTS: ['/' | '\ ' | '|' | '-' | '+' | 'x' | 'o' | 'O' | '?' | '*']

`set_joinstyle(js)`

Set the joinstyle for the collection. The joinstyle can only be set globally for all elements in the collection.

Parameters

`js` [['miter' | 'round' | 'bevel']] The joinstyle

`set_label(s)`

Set the label to `s` for auto legend.

Parameters

`s` [object] `s` will be converted to a string by calling `str` (unicode on Py2).

`set_linestyle(ls)`

Set the linestyle(`s`) for the collection.

linestyle	description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line

Alternatively a dash tuple of the following form can be provided:

(offset, onoffseq),

where `onoffseq` is an even length tuple of on and off ink in points.

ACCEPTS: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '--' | '-.' | ':' | 'None' | ' ' | '']

Parameters

`ls` [{ '-', '--', '-.', ':'} and more see description] The line style.

`set_linestyles(ls)`

alias for `set_linestyle`

`set_linewidth(lw)`

Set the linewidth(`s`) for the collection. `lw` can be a scalar or a sequence; if it is a sequence the patches will cycle through the sequence

ACCEPTS: float or sequence of floats

`set_linewidths(lw)`
alias for `set_linewidth`

`set_lw(lw)`
alias for `set_linewidth`

`set_norm(norm)`
Set the normalization instance.

Parameters

`norm` [Normalize]

`set_offset_position(offset_position)`
Set how offsets are applied. If `offset_position` is 'screen' (default) the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`set_offsets(offsets)`
Set the offsets for the collection. `offsets` can be a scalar or a sequence.

ACCEPTS: float or sequence of floats

`set_path_effects(path_effects)`
Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`set_paths()`

`set_picker(picker)`
Set the epsilon for picking used by this artist

`picker` can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if `picker` is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if `picker` is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

`hit, props = picker(artist, mouseevent)`

to determine the hit test. if the mouse event is over the artist, return `hit=True` and `props` is a dictionary of properties you want added to the `PickEvent` attributes.

Parameters

`picker` [None or bool or float or callable]

`set_pickradius(pr)`

Set the pick radius used for containment tests.

Parameters

`d` [float] Pick radius, in points.

`set_rasterized(rasterized)`

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

`rasterized` [bool or None]

`set_sketch_params(scale=None, length=None, randomness=None)`

Sets the sketch parameters.

Parameters

`scale` [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If `scale` is `None`, or not provided, no sketch filter will be provided.

`length` [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

`randomness` [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

`set_snap(snap)`

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

`snap` [bool or None]

`set_transform(t)`

Set the artist transform.

Parameters

`t` [Transform]

`set_url(url)`

Sets the url for the artist.

Parameters

`url` [str]

`set_urls(urls)`

Parameters

`urls` [List[str] or None]

`set_visible(b)`

Set the artist's visibility.

Parameters

`b` [bool]

`set_zorder(level)`

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

`level` [float]

`stale`

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

`sticky_edges`

x and y sticky edge lists.

When performing autoscaling, if a data limit coincides with a value in the corresponding `sticky_edges` list, then no margin will be added—the view limit "sticks" to the edge. A typical usecase is histograms, where one usually expects no margin on the bottom edge (0) of the histogram.

This attribute cannot be assigned to; however, the x and y lists can be modified in place as needed.

Examples

```
>>> artist.sticky_edges.x[:] = (xmin, xmax)
>>> artist.sticky_edges.y[:] = (ymin, ymax)
```

`to_rgba(x, alpha=None, bytes=False, norm=True)`

Return a normalized rgba array corresponding to x.

In the normal case, `x` is a 1-D or 2-D sequence of scalars, and the corresponding ndarray of `rgba` values will be returned, based on the `norm` and `colormap` set for this `ScalarMappable`.

There is one special case, for handling images that are already `rgb` or `rgba`, such as might have been read from an image file. If `x` is an ndarray with 3 dimensions, and the last dimension is either 3 or 4, then it will be treated as an `rgb` or `rgba` array, and no mapping will be done. The array can be `uint8`, or it can be floating point with values in the 0-1 range; otherwise a `ValueError` will be raised. If it is a masked array, the mask will be ignored. If the last dimension is 3, the `alpha` kwarg (defaulting to 1) will be used to fill in the transparency. If the last dimension is 4, the `alpha` kwarg is ignored; it does not replace the pre-existing `alpha`. A `ValueError` will be raised if the third dimension is other than 3 or 4.

In either case, if `bytes` is `False` (default), the `rgba` array will be floats in the 0-1 range; if it is `True`, the returned `rgba` array will be `uint8` in the 0 to 255 range.

If `norm` is `False`, no normalization of the input data is performed, and it is assumed to be in the range (0-1).

`update(props)`

Update this artist's properties from the dictionary `prop`.

`update_from(other)`

copy properties from `other` to self

`update_scalarmappable()`

If the scalar mappable array is not none, update colors from scalar data

`zorder = 0`

```
class matplotlib.collections.EllipseCollection(widths, heights, angles, units='points',
                                              **kwargs)
```

Bases: `matplotlib.collections.Collection`

A collection of ellipses, drawn using splines.

`widths`: sequence lengths of first axes (e.g., major axis lengths)

`heights`: sequence lengths of second axes

`angles`: sequence angles of first axes, degrees CCW from the X-axis

`units`: ['points' | 'inches' | 'dots' | 'width' | 'height' | 'x' | 'y' | 'xy']

units in which majors and minors are given; 'width' and 'height' refer to the dimensions of the axes, while 'x' and 'y' refer to the offsets data units. 'xy' differs from all others in that the angle as plotted varies with the aspect ratio, and equals the specified angle only when the aspect ratio is unity. Hence it behaves the same as the `Ellipse` with `axes.transData` as its transform.

Additional kwargs inherited from the base `Collection`:

Valid `Collection` keyword arguments:

- `edgecolors`: None
- `facecolors`: None
- `linewidths`: None
- `antialiaseds`: None
- `offsets`: None
- `transOffset`: `transforms.IdentityTransform()`
- `norm`: None (optional for `matplotlib.cm.ScalarMappable`)
- `cmap`: None (optional for `matplotlib.cm.ScalarMappable`)

`offsets` and `transOffset` are used to translate the patch after rendering (default no offsets)

If any of `edgecolors`, `facecolors`, `linewidths`, `antialiaseds` are None, they default to their `matplotlib.rcParams` patch setting, in sequence form.

`add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`add_checker(checker)`

Add an entry to a dictionary of boolean flags that are set to True when the mappable is changed.

`aname = 'Artist'`

`autoscale()`

Autoscale the scalar limits on the norm instance using the current array

`autoscale__None()`

Autoscale the scalar limits on the norm instance using the current array, changing only limits that are None

`axes`

The Axes instance the artist resides in, or None.

`changed()`

Call this whenever the mappable is changed to notify all the callbackSM listeners to the 'changed' signal

`check_update(checker)`

If mappable has changed since the last check, return True; else return False

`contains(mouseevent)`

Test whether the mouse event occurred in the collection.

Returns True | False, `dict(ind=itemlist)`, where every item in `itemlist` contains the event.

`convert_xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`convert_yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`draw(renderer)`

Derived classes drawing method

`findobj(match=None, include_self=True)`

Find artist objects.

Recursively find all Artist instances contained in self.

match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is `True` (default), include self in the list to be checked for a match.

`format_cursor_data(data)`

Return cursor data string formatted.

`get_agg_filter()`

Return filter function to be used for agg filter.

`get_alpha()`

Return the alpha value used for blending - not supported on all backends

`get_animated()`

Return the artist's animated state

`get_array()`

Return the array

`get_capstyle()`

`get_children()`

Return a list of the child Artist's this :class:`Artist` contains.

`get_clim()`

return the min, max of the color limits for image scaling

`get_clip_box()`

Return artist clipbox

`get_clip_on()`

Return whether artist uses clipping

`get_clip_path()`

Return artist clip path

`get_cmap()`
return the colormap

`get_contains()`
Return the `__contains` test used by the artist, or `None` for default.

`get_cursor_data(event)`
Get the cursor data for a given event.

`get_dashes()`

`get_datalim(transData)`

`get_edgecolor()`

`get_edgecolors()`

`get_facecolor()`

`get_facecolors()`

`get_figure()`
Return the Figure instance the artist belongs to.

`get_fill()`
return whether fill is set

`get_gid()`
Returns the group id.

`get_hatch()`
Return the current hatching pattern.

`get_joinstyle()`

`get_label()`
Get the label used for this artist in the legend.

`get_linestyle()`

`get_linestyles()`

`get_linewidth()`

`get_linewidths()`

`get_offset_position()`

Returns how offsets are applied for the collection. If `offset_position` is 'screen', the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`get_offset_transform()`

`get_offsets()`

Return the offsets for the collection.

`get_path_effects()`

`get_paths()`

`get_picker()`

Return the picker object used by this artist.

`get_pickradius()`

`get_rasterized()`

Return whether the artist is to be rasterized.

`get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

`sketch_params` [tuple or `None`]

A 3-tuple with the following elements:

- `scale`: The amplitude of the wiggle perpendicular to the source line.
- `length`: The length of the wiggle along the line.
- `randomness`: The scale factor by which the length is shrunk or expanded.

May return 'None' if no sketch parameters were set.

`get_snap()`

Returns the snap setting which may be:

- `True`: snap vertices to the nearest pixel center
- `False`: leave vertices as-is
- `None`: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`get_transform()`
Return the Transform instance used by this artist.

`get_transformed_clip_path_and_affine()`
Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`get_transforms()`

`get_url()`
Returns the url.

`get_urls()`

`get_visible()`
Return the artist's visibility

`get_window_extent(renderer)`
Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`get_zorder()`
Return the artist's zorder.

`have_units()`
Return True if units are set on the x or y axes

`hitlist(event)`
Deprecated since version 2.2: The hitlist function was deprecated in version 2.2.

List the children of the artist which contain the mouse event event.

`is_figure_set()`
Deprecated since version 2.2: `artist.figure` is not None

Returns whether the artist is assigned to a Figure.

`is_transform_set()`
Returns True if Artist has a transform explicitly set.

`mouseover`

`pchanged()`
Fire an event when property changed, calling all of the registered callbacks.

`pick(mouseevent)`
Process pick event

each child artist will fire a pick event if `mouseevent` is over the artist and the artist has picker set

`pickable()`
Return True if Artist is pickable.

`properties()`
return a dictionary mapping property name -> value for all Artist props

`remove()`
Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`remove_callback(oid)`
Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`set(**kwargs)`
A property batch setter. Pass kwargs to set properties.

`set_agg_filter(filter_func)`
Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`set_alpha(alpha)`
Set the alpha transparencies of the collection. `alpha` must be a float or None.

ACCEPTS: float or None

`set_animated(b)`
Set the artist's animation state.

Parameters

`b` [bool]

`set_antialiased(aa)`
Set the antialiasing state for rendering.

ACCEPTS: Boolean or sequence of booleans

`set_antialiaseds(aa)`
 alias for `set_antialiased`

`set_array(A)`
 Set the image array from numpy array A.

Parameters

A [ndarray]

`set_capstyle(cs)`
 Set the capstyle for the collection. The capstyle can only be set globally for all elements in the collection

Parameters

cs [['butt' | 'round' | 'projecting']] The capstyle

`set_clim(vmin=None, vmax=None)`
 set the norm limits for image scaling; if vmin is a length2 sequence, interpret it as (vmin, vmax) which is used to support setp

ACCEPTS: a length 2 sequence of floats; may be overridden in methods that have vmin and vmax kwargs.

`set_clip_box(clipbox)`
 Set the artist's clip Bbox.

Parameters

clipbox [Bbox]

`set_clip_on(b)`
 Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

b [bool]

`set_clip_path(path, transform=None)`
 Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

`set_cmap(cmap)`

set the colormap for luminance data

ACCEPTS: a colormap or registered colormap name

`set_color(c)`

Set both the edgecolor and the facecolor.

ACCEPTS: matplotlib color arg or sequence of rgba tuples

See also:

`set_facecolor()`, `set_edgecolor()` For setting the edge or face color individually.

`set_contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

`picker` [callable]

`set_dashes(ls)`

alias for `set_linestyle`

`set_edgecolor(c)`

Set the edgecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'face', the edge color will always be the same as the face color. If it is 'none', the patch boundary will not be drawn.

ACCEPTS: matplotlib color spec or sequence of specs

`set_edgecolors(c)`

alias for `set_edgecolor`

`set_facecolor(c)`

Set the facecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'none', the patch will not be filled.

ACCEPTS: matplotlib color spec or sequence of specs

`set_facecolors(c)`

alias for `set_facecolor`

`set_figure(fig)`
Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`set_gid(gid)`
Sets the (group) id for the artist.

Parameters

`gid` [str]

`set_hatch(hatch)`
Set the hatching pattern

`hatch` can be one of:

```
/ - diagonal hatching
\ - back diagonal
| - vertical
- - horizontal
+ - crossed
x - crossed diagonal
o - small circle
O - large circle
. - dots
* - stars
```

Letters can be combined, in which case all the specified hatchings are done. If same letter repeats, it increases the density of hatching of that pattern.

Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Unlike other properties such as linewidth and colors, hatching can only be specified for the collection as a whole, not separately for each member.

ACCEPTS: ['/' | '\ ' | '|' | '- ' | '+ ' | 'x' | 'o' | 'O' | '.' | '*']

`set_joinstyle(js)`
Set the joinstyle for the collection. The joinstyle can only be set globally for all elements in the collection.

Parameters

`js` [['miter' | 'round' | 'bevel']] The joinstyle

`set_label(s)`
Set the label to `s` for auto legend.

Parameters

`s` [object] `s` will be converted to a string by calling `str` (unicode on Py2).

`set_linestyle(ls)`

Set the linestyle(s) for the collection.

linestyle	description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line

Alternatively a dash tuple of the following form can be provided:

(offset, onoffseq),

where onoffseq is an even length tuple of on and off ink in points.

ACCEPTS: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '--' |
'-.' | ':' | 'None' | ' ' | '']

Parameters

ls [{'-', '--', '-.', ':'} and more see description] The line style.

`set_linestyles(ls)`

alias for `set_linestyle`

`set_linewidth(lw)`

Set the linewidth(s) for the collection. lw can be a scalar or a sequence; if it is a sequence the patches will cycle through the sequence

ACCEPTS: float or sequence of floats

`set_linewidths(lw)`

alias for `set_linewidth`

`set_lw(lw)`

alias for `set_linewidth`

`set_norm(norm)`

Set the normalization instance.

Parameters

norm [Normalize]

`set_offset_position(offset_position)`

Set how offsets are applied. If `offset_position` is 'screen' (default) the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`set_offsets(offsets)`

Set the offsets for the collection. offsets can be a scalar or a sequence.

ACCEPTS: float or sequence of floats

`set_path_effects(path_effects)`

Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`set_paths()`

`set_picker(picker)`

Set the epsilon for picking used by this artist

`picker` can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if `picker` is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if `picker` is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

to determine the hit test. if the mouse event is over the artist, return `hit=True` and `props` is a dictionary of properties you want added to the `PickEvent` attributes.

Parameters

`picker` [None or bool or float or callable]

`set_pickradius(pr)`

Set the pick radius used for containment tests.

Parameters

`d` [float] Pick radius, in points.

`set_rasterized(rasterized)`

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

`rasterized` [bool or None]

`set_sketch_params(scale=None, length=None, randomness=None)`

Sets the sketch parameters.

Parameters

`scale` [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If `scale` is `None`, or not provided, no sketch filter will be provided.

`length` [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

`randomness` [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

`set_snap(snap)`

Sets the snap setting which may be:

- `True`: snap vertices to the nearest pixel center
- `False`: leave vertices as-is
- `None`: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

`snap` [bool or `None`]

`set_transform(t)`

Set the artist transform.

Parameters

`t` [Transform]

`set_url(url)`

Sets the url for the artist.

Parameters

`url` [str]

`set_urls(urls)`

Parameters

`urls` [List[str] or `None`]

`set_visible(b)`

Set the artist's visibility.

Parameters

`b` [bool]

`set_zorder(level)`

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

`level` [float]

`stale`

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

`sticky_edges`

x and y sticky edge lists.

When performing autoscaling, if a data limit coincides with a value in the corresponding `sticky_edges` list, then no margin will be added—the view limit "sticks" to the edge. A typical usecase is histograms, where one usually expects no margin on the bottom edge (0) of the histogram.

This attribute cannot be assigned to; however, the x and y lists can be modified in place as needed.

Examples

```
>>> artist.sticky_edges.x[:] = (xmin, xmax)
>>> artist.sticky_edges.y[:] = (ymin, ymax)
```

`to_rgba(x, alpha=None, bytes=False, norm=True)`

Return a normalized rgba array corresponding to x.

In the normal case, x is a 1-D or 2-D sequence of scalars, and the corresponding ndarray of rgba values will be returned, based on the norm and colormap set for this `ScalarMappable`.

There is one special case, for handling images that are already rgb or rgba, such as might have been read from an image file. If x is an ndarray with 3 dimensions, and the last dimension is either 3 or 4, then it will be treated as an rgb or rgba array, and no mapping will be done. The array can be uint8, or it can be floating point with values in the 0-1 range; otherwise a `ValueError` will be raised. If it is a masked array, the mask will be ignored. If the last dimension is 3, the alpha kwarg (defaulting to 1) will be used to fill in the transparency. If the last dimension is 4, the alpha kwarg is ignored; it does not replace the pre-existing alpha. A `ValueError` will be raised if the third dimension is other than 3 or 4.

In either case, if bytes is False (default), the rgba array will be floats in the 0-1 range; if it is True, the returned rgba array will be uint8 in the 0 to 255 range.

If norm is False, no normalization of the input data is performed, and it is assumed to be in the range (0-1).

`update(props)`

Update this artist's properties from the dictionary prop.

```
update_from(other)
    copy properties from other to self

update_scalarmappable()
    If the scalar mappable array is not none, update colors from scalar data

zorder = 0
```

```
class matplotlib.collections.EventCollection(positions, orientation=None, lineoffset=0,
                                             linelength=1, linewidth=None, color=None,
                                             linestyle='solid', antialiased=None, **kwargs)
```

Bases: `matplotlib.collections.LineCollection`

A collection of discrete events.

The events are given by a 1-dimensional array, usually the position of something along an axis, such as time or length. They do not have an amplitude and are displayed as vertical or horizontal parallel bars.

Parameters

`positions` [1D array-like object] Each value is an event.

`orientation` [{None, 'horizontal', 'vertical'}, optional] The orientation of the collection (the event bars are along the orthogonal direction). Defaults to 'horizontal' if not specified or None.

`lineoffset` [scalar, optional, default: 0] The offset of the center of the markers from the origin, in the direction orthogonal to orientation.

`linelength` [scalar, optional, default: 1] The total height of the marker (i.e. the marker stretches from `lineoffset - linelength/2` to `lineoffset + linelength/2`).

`linewidth` [scalar or None, optional, default: None] If it is None, defaults to its rcParams setting, in sequence form.

`color` [color, sequence of colors or None, optional, default: None] If it is None, defaults to its rcParams setting, in sequence form.

`linestyle` [str or tuple, optional, default: 'solid'] Valid strings are ['solid', 'dashed', 'dashdot', 'dotted', '-', '-.', ':']. Dash tuples should be of the form:

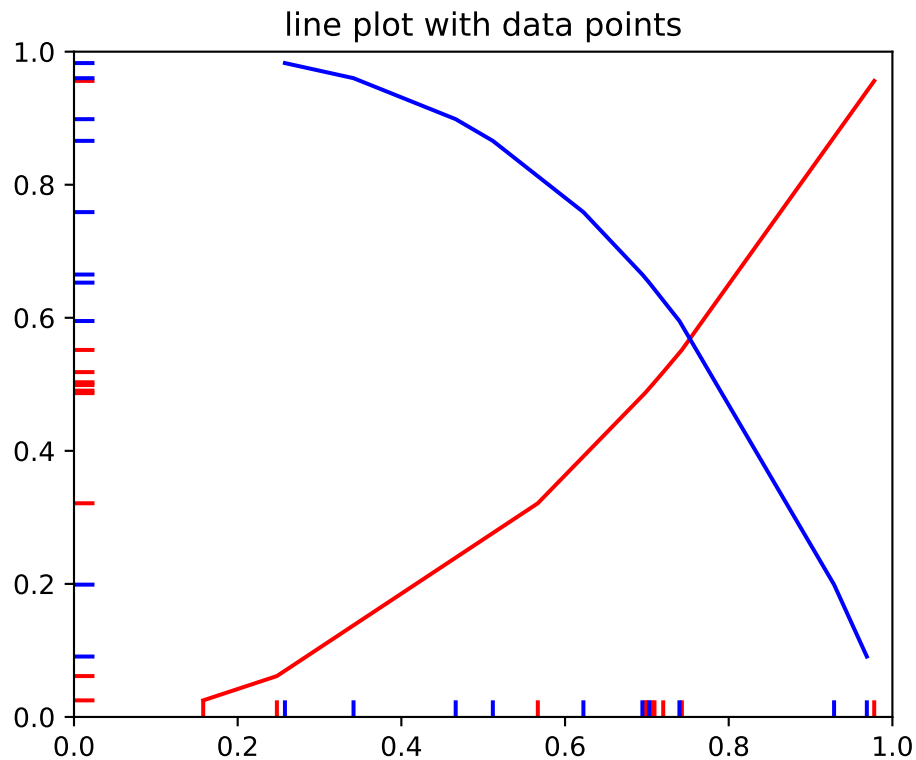
`(offset, onoffseq),`

where `onoffseq` is an even length tuple of on and off ink in points.

`antialiased` [{None, 1, 2}, optional] If it is None, defaults to its rcParams setting, in sequence form.

`**kwargs` [optional] Other keyword arguments are line collection properties. See `LineCollection` for a list of the valid properties.

Examples



`add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`add_checker(checker)`

Add an entry to a dictionary of boolean flags that are set to True when the mappable is changed.

`add_positions(position)`

add one or more events at the specified positions

`aname = 'Artist'`

`append_positions(position)`

add one or more events at the specified positions

`autoscale()`

Autoscale the scalar limits on the norm instance using the current array

`autoscale_None()`

Autoscale the scalar limits on the norm instance using the current array, changing only

limits that are None

axes

The Axes instance the artist resides in, or None.

changed()

Call this whenever the mappable is changed to notify all the callbackSM listeners to the 'changed' signal

check_update(checker)

If mappable has changed since the last check, return True; else return False

contains(mouseevent)

Test whether the mouse event occurred in the collection.

Returns True | False, dict(ind=itemlist), where every item in itemlist contains the event.

convert_xunits(x)

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

convert_yunits(y)

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

draw(renderer)

Derived classes drawing method

extend_positions(position)

add one or more events at the specified positions

findobj(match=None, include_self=True)

Find artist objects.

Recursively find all Artist instances contained in self.

match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is True (default), include self in the list to be checked for a match.

format_cursor_data(data)

Return cursor data string formatted.

get_agg_filter()

Return filter function to be used for agg filter.

get_alpha()

Return the alpha value used for blending - not supported on all backends

get_animated()

Return the artist's animated state

get_array()

Return the array

`get_capstyle()`

`get_children()`

Return a list of the child Artist`s this :class:`Artist` contains.

`get_clim()`

return the min, max of the color limits for image scaling

`get_clip_box()`

Return artist clipbox

`get_clip_on()`

Return whether artist uses clipping

`get_clip_path()`

Return artist clip path

`get_cmap()`

return the colormap

`get_color()`

get the color of the lines used to mark each event

`get_colors()`

`get_contains()`

Return the `_contains` test used by the artist, or `None` for default.

`get_cursor_data(event)`

Get the cursor data for a given event.

`get_dashes()`

`get_datalim(transData)`

`get_edgecolor()`

`get_edgecolors()`

`get_facecolor()`

`get_facecolors()`

`get_figure()`

Return the Figure instance the artist belongs to.

`get_fill()`

return whether fill is set

`get_gid()`
Returns the group id.

`get_hatch()`
Return the current hatching pattern.

`get_joinstyle()`

`get_label()`
Get the label used for this artist in the legend.

`get_linelength()`
get the length of the lines used to mark each event

`get_lineoffset()`
get the offset of the lines used to mark each event

`get_linestyle()`
get the style of the lines used to mark each event ['solid' | 'dashed' | 'dashdot' | 'dotted']

`get_linestyles()`

`get_linewidth()`
get the width of the lines used to mark each event

`get_linewidths()`

`get_offset_position()`
Returns how offsets are applied for the collection. If `offset_position` is 'screen', the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`get_offset_transform()`

`get_offsets()`
Return the offsets for the collection.

`get_orientation()`
get the orientation of the event line, may be: ['horizontal' | 'vertical']

`get_path_effects()`

`get_paths()`

`get_picker()`
Return the picker object used by this artist.

`get_pickradius()`

`get_positions()`

return an array containing the floating-point values of the positions

`get_rasterized()`

Return whether the artist is to be rasterized.

`get_segments()`

Returns

segments [list] List of segments in the LineCollection. Each list item contains an array of vertices.

`get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

sketch_params [tuple or `None`]

A 3-tuple with the following elements:

- scale: The amplitude of the wiggle perpendicular to the source line.
- length: The length of the wiggle along the line.
- randomness: The scale factor by which the length is shrunk or expanded.

May return ‘None’ if no sketch parameters were set.

`get_snap()`

Returns the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`get_transform()`

Return the Transform instance used by this artist.

`get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`get_transforms()`

`get_url()`

Returns the url.

`get_urls()`

`get_visible()`

Return the artist's visibility

`get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`get_zorder()`

Return the artist's zorder.

`have_units()`

Return True if units are set on the x or y axes

`hitlist(event)`

Deprecated since version 2.2: The hitlist function was deprecated in version 2.2.

List the children of the artist which contain the mouse event event.

`is_figure_set()`

Deprecated since version 2.2: `artist.figure` is not None

Returns whether the artist is assigned to a Figure.

`is_horizontal()`

True if the eventcollection is horizontal, False if vertical

`is_transform_set()`

Returns True if Artist has a transform explicitly set.

`mouseover`

`pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`pick(mouseevent)`

Process pick event

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

`pickable()`

Return True if Artist is pickable.

`properties()`

return a dictionary mapping property name -> value for all Artist props

`remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`set(**kwargs)`

A property batch setter. Pass kwargs to set properties.

`set_agg_filter(filter_func)`

Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`set_alpha(alpha)`

Set the alpha transparencies of the collection. `alpha` must be a float or `None`.

ACCEPTS: float or `None`

`set_animated(b)`

Set the artist's animation state.

Parameters

`b` [bool]

`set_antialiased(aa)`

Set the antialiasing state for rendering.

ACCEPTS: Boolean or sequence of booleans

`set_antialiaseds(aa)`

alias for `set_antialiased`

`set_array(A)`

Set the image array from numpy array `A`.

Parameters

`A` [ndarray]

`set_capstyle(cs)`

Set the capstyle for the collection. The capstyle can only be set globally for all elements in the collection

Parameters

`cs` `['butt' | 'round' | 'projecting']` The capstyle

`set_clim(vmin=None, vmax=None)`

set the norm limits for image scaling; if `vmin` is a `length2` sequence, interpret it as `(vmin, vmax)` which is used to support `setp`

ACCEPTS: a length 2 sequence of floats; may be overridden in methods that have `vmin` and `vmax` kwargs.

`set_clip_box(clipbox)`

Set the artist's clip Bbox.

Parameters

`clipbox` `[Bbox]`

`set_clip_on(b)`

Set whether artist uses clipping.

When `False` artists will be visible out side of the axes which can lead to unexpected results.

Parameters

`b` `[bool]`

`set_clip_path(path, transform=None)`

Set the artist's clip path, which may be:

- a `Patch` (or subclass) instance; or
- a `Path` instance, in which case a `Transform` instance, which will be applied to the path before using it for clipping, must be provided; or
- `None`, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to `None`.

ACCEPTS: `[(Path, Transform) | Patch | None]`

`set_cmap(cmap)`

set the colormap for luminance data

ACCEPTS: a colormap or registered colormap name

`set_color(c)`

Set the color(s) of the `LineCollection`.

Parameters

`c` : Matplotlib color argument (all patches have same color), or a sequence or rgba tuples; if it is a sequence the patches will cycle through the sequence.

`set_contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

`picker` [callable]

`set_dashes(ls)`

alias for `set_linestyle`

`set_edgecolor(c)`

Set the edgecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'face', the edge color will always be the same as the face color. If it is 'none', the patch boundary will not be drawn.

ACCEPTS: matplotlib color spec or sequence of specs

`set_edgecolors(c)`

alias for `set_edgecolor`

`set_facecolor(c)`

Set the facecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'none', the patch will not be filled.

ACCEPTS: matplotlib color spec or sequence of specs

`set_facecolors(c)`

alias for `set_facecolor`

`set_figure(fig)`

Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`set_gid(gid)`

Sets the (group) id for the artist.

Parameters

gid [str]
set_hatch(hatch)
Set the hatching pattern

hatch can be one of:

/ - diagonal hatching
\ - back diagonal
| - vertical
- - horizontal
+ - crossed
x - crossed diagonal
o - small circle
O - large circle
. - dots
* - stars

Letters can be combined, in which case all the specified hatchings are done. If same letter repeats, it increases the density of hatching of that pattern.

Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Unlike other properties such as linewidth and colors, hatching can only be specified for the collection as a whole, not separately for each member.

ACCEPTS: ['/' | '\' | '|' | '-' | '+' | 'x' | 'o' | 'O' | '.' | '*']

set_joinstyle(js)
Set the joinstyle for the collection. The joinstyle can only be set globally for all elements in the collection.

Parameters

js [['miter' | 'round' | 'bevel']] The joinstyle

set_label(s)
Set the label to s for auto legend.

Parameters

s [object] s will be converted to a string by calling `str` (unicode on Py2).

set_linelength(linlength)
set the length of the lines used to mark each event

set_lineoffset(lineoffset)
set the offset of the lines used to mark each event

set_linestyle(ls)
Set the linestyle(s) for the collection.

linestyle	description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line

Alternatively a dash tuple of the following form can be provided:

(offset, onoffseq),

where onoffseq is an even length tuple of on and off ink in points.

ACCEPTS: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '--' | '-.' | ':' | 'None' | '' | '']

Parameters

ls [{'-', '-', '-.', ':'}] and more see description] The line style.

set_linestyles(ls)

alias for set_linestyle

set_linewidth(lw)

Set the linewidth(s) for the collection. lw can be a scalar or a sequence; if it is a sequence the patches will cycle through the sequence

ACCEPTS: float or sequence of floats

set_linewidths(lw)

alias for set_linewidth

set_lw(lw)

alias for set_linewidth

set_norm(norm)

Set the normalization instance.

Parameters

norm [Normalize]

set_offset_position(offset_position)

Set how offsets are applied. If offset_position is 'screen' (default) the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If offset_position is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

set_offsets(offsets)

Set the offsets for the collection. offsets can be a scalar or a sequence.

ACCEPTS: float or sequence of floats

`set_orientation(orientation=None)`

set the orientation of the event line ['horizontal' | 'vertical' | None] defaults to 'horizontal' if not specified or None

`set_path_effects(path_effects)`

Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`set_paths(segments)`

`set_picker(picker)`

Set the epsilon for picking used by this artist

picker can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if picker is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if picker is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

to determine the hit test. if the mouse event is over the artist, return hit=True and props is a dictionary of properties you want added to the PickEvent attributes.

Parameters

`picker` [None or bool or float or callable]

`set_pickradius(pr)`

Set the pick radius used for containment tests.

Parameters

`d` [float] Pick radius, in points.

`set_positions(positions)`

set the positions of the events to the specified value

`set_rasterized(rasterized)`

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

rasterized [bool or None]

set__segments(segments)

set__sketch__params(scale=None, length=None, randomness=None)

Sets the sketch parameters.

Parameters

scale [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If scale is `None`, or not provided, no sketch filter will be provided.

length [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

randomness [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

set__snap(snap)

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

snap [bool or None]

set__transform(t)

Set the artist transform.

Parameters

t [Transform]

set__url(url)

Sets the url for the artist.

Parameters

url [str]

set__urls(urls)

Parameters

urls [List[str] or None]

`set_verts(segments)`

`set_visible(b)`

Set the artist's visibility.

Parameters

`b` [bool]

`set_zorder(level)`

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

`level` [float]

`stale`

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

`sticky_edges`

x and y sticky edge lists.

When performing autoscaling, if a data limit coincides with a value in the corresponding `sticky_edges` list, then no margin will be added—the view limit "sticks" to the edge. A typical usecase is histograms, where one usually expects no margin on the bottom edge (0) of the histogram.

This attribute cannot be assigned to; however, the x and y lists can be modified in place as needed.

Examples

```
>>> artist.sticky_edges.x[:] = (xmin, xmax)
>>> artist.sticky_edges.y[:] = (ymin, ymax)
```

`switch_orientation()`

switch the orientation of the event line, either from vertical to horizontal or vice versus

`to_rgba(x, alpha=None, bytes=False, norm=True)`

Return a normalized rgba array corresponding to x.

In the normal case, x is a 1-D or 2-D sequence of scalars, and the corresponding ndarray of rgba values will be returned, based on the norm and colormap set for this `ScalarMappable`.

There is one special case, for handling images that are already rgb or rgba, such as might have been read from an image file. If x is an ndarray with 3 dimensions, and the last dimension is either 3 or 4, then it will be treated as an rgb or rgba array, and no mapping will be done. The array can be uint8, or it can be floating point with values in the 0-1 range; otherwise a `ValueError` will be raised. If it is a masked array, the mask will be ignored. If the last dimension is 3, the alpha kwarg (defaulting to 1) will be used

to fill in the transparency. If the last dimension is 4, the alpha kwarg is ignored; it does not replace the pre-existing alpha. A `ValueError` will be raised if the third dimension is other than 3 or 4.

In either case, if `bytes` is `False` (default), the `rgba` array will be floats in the 0-1 range; if it is `True`, the returned `rgba` array will be `uint8` in the 0 to 255 range.

If `norm` is `False`, no normalization of the input data is performed, and it is assumed to be in the range (0-1).

`update(props)`

Update this artist's properties from the dictionary `prop`.

`update_from(other)`

copy properties from `other` to self

`update_scalarmappable()`

If the scalar mappable array is not none, update colors from scalar data

`zorder = 0`

```
class matplotlib.collections.LineCollection(segments, linewidths=None, colors=None, antialiaseds=None,
                                             linestyle='solid', offsets=None,
                                             transOffset=None, norm=None, cmap=None,
                                             pickradius=5, zorder=2, facecolors='none',
                                             **kwargs)
```

Bases: `matplotlib.collections.Collection`

All parameters must be sequences or scalars; if scalars, they will be converted to sequences. The property of the *i*th line segment is:

`prop[i % len(props)]`

i.e., the properties cycle if the `len` of `props` is less than the number of segments.

Parameters

`segments` : A sequence of (line0, line1, line2), where:

`linen = (x0, y0), (x1, y1), ... (xm, ym)`

or the equivalent numpy array with two columns. Each line can be a different length.

`colors` [sequence, optional] A sequence of RGBA tuples (e.g., arbitrary color strings, etc, not allowed).

`antialiaseds` [sequence, optional] A sequence of ones or zeros.

`linestyles` [string, tuple, optional] Either one of ['solid' | 'dashed' | 'dashdot' | 'dotted'], or a dash tuple. The dash tuple is:

`(offset, onoffseq)`

where `onoffseq` is an even length tuple of on and off ink in points.

`norm` [Normalize, optional] Normalize instance.

`cmap` [string or Colormap, optional] Colormap name or Colormap instance.

`pickradius` [float, optional] The tolerance in points for mouse clicks picking a line. Default is 5 pt.

`zorder` [int, optional] zorder of the LineCollection. Default is 2.

`facecolors` [optional] The facecolors of the LineCollection. Default is 'none'. Setting to a value other than 'none' will lead to a filled polygon being drawn between points on each line.

Notes

If `linewidths`, `colors`, or `antialiaseds` is `None`, they default to their `rcParams` setting, in sequence form.

If `offsets` and `transOffset` are not `None`, then `offsets` are transformed by `transOffset` and applied after the segments have been transformed to display coordinates.

If `offsets` is not `None` but `transOffset` is `None`, then the `offsets` are added to the segments before any transformation. In this case, a single offset can be specified as:

`offsets=(xo,yo)`

and this value will be added cumulatively to each successive segment, so as to produce a set of successively offset curves.

The use of `ScalarMappable` is optional. If the `ScalarMappable` array `__A` is not `None` (i.e., a call to `set_array()` has been made), at draw time a call to `scalar mappable` will be made to set the colors.

`add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`add_checker(checker)`

Add an entry to a dictionary of boolean flags that are set to `True` when the mappable is changed.

`aname = 'Artist'`

`autoscale()`

Autoscale the scalar limits on the norm instance using the current array

`autoscale_None()`

Autoscale the scalar limits on the norm instance using the current array, changing only limits that are `None`

`axes`
The Axes instance the artist resides in, or None.

`changed()`
Call this whenever the mappable is changed to notify all the callbackSM listeners to the 'changed' signal

`check_update(checker)`
If mappable has changed since the last check, return True; else return False

`contains(mouseevent)`
Test whether the mouse event occurred in the collection.

Returns True | False, dict(ind=itemlist), where every item in itemlist contains the event.

`convert_xunits(x)`
For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`convert_yunits(y)`
For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`draw(renderer)`
Derived classes drawing method

`findobj(match=None, include_self=True)`
Find artist objects.

Recursively find all Artist instances contained in self.

match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is True (default), include self in the list to be checked for a match.

`format_cursor_data(data)`
Return cursor data string formatted.

`get_agg_filter()`
Return filter function to be used for agg filter.

`get_alpha()`
Return the alpha value used for blending - not supported on all backends

`get_animated()`
Return the artist's animated state

`get_array()`
Return the array

`get_capstyle()`

`get_children()`
Return a list of the child Artist's this :class:`Artist` contains.

`get_clim()`
return the min, max of the color limits for image scaling

`get_clip_box()`
Return artist clipbox

`get_clip_on()`
Return whether artist uses clipping

`get_clip_path()`
Return artist clip path

`get_cmap()`
return the colormap

`get_color()`

`get_colors()`

`get_contains()`
Return the `__contains` test used by the artist, or None for default.

`get_cursor_data(event)`
Get the cursor data for a given event.

`get_dashes()`

`get_datalim(transData)`

`get_edgecolor()`

`get_edgecolors()`

`get_facecolor()`

`get_facecolors()`

`get_figure()`
Return the Figure instance the artist belongs to.

`get_fill()`
return whether fill is set

`get_gid()`
Returns the group id.

`get_hatch()`

Return the current hatching pattern.

`get_joinstyle()`

`get_label()`

Get the label used for this artist in the legend.

`get_linestyle()`

`get_linestyles()`

`get_linewidth()`

`get_linewidths()`

`get_offset_position()`

Returns how offsets are applied for the collection. If `offset_position` is 'screen', the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`get_offset_transform()`

`get_offsets()`

Return the offsets for the collection.

`get_path_effects()`

`get_paths()`

`get_picker()`

Return the picker object used by this artist.

`get_pickradius()`

`get_rasterized()`

Return whether the artist is to be rasterized.

`get_segments()`

Returns

`segments` [list] List of segments in the `LineCollection`. Each list item contains an array of vertices.

`get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

`sketch_params` [tuple or `None`]

A 3-tuple with the following elements:

- `scale`: The amplitude of the wiggle perpendicular to the source line.
- `length`: The length of the wiggle along the line.
- `randomness`: The scale factor by which the length is shrunk or expanded.

May return `'None'` if no sketch parameters were set.

`get_snap()`

Returns the snap setting which may be:

- `True`: snap vertices to the nearest pixel center
- `False`: leave vertices as-is
- `None`: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`get_transform()`

Return the Transform instance used by this artist.

`get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`get_transforms()`

`get_url()`

Returns the url.

`get_urls()`

`get_visible()`

Return the artist's visibility

`get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done

when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`get_zorder()`

Return the artist's zorder.

`have_units()`

Return True if units are set on the x or y axes

`hitlist(event)`

Deprecated since version 2.2: The hitlist function was deprecated in version 2.2.

List the children of the artist which contain the mouse event event.

`is_figure_set()`

Deprecated since version 2.2: `artist.figure` is not None

Returns whether the artist is assigned to a Figure.

`is_transform_set()`

Returns True if Artist has a transform explicitly set.

`mouseover`

`pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`pick(mouseevent)`

Process pick event

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

`pickable()`

Return True if Artist is pickable.

`properties()`

return a dictionary mapping property name -> value for all Artist props

`remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`set(**kwargs)`

A property batch setter. Pass kwargs to set properties.

`set_agg_filter(filter_func)`

Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`set_alpha(alpha)`

Set the alpha transparencies of the collection. alpha must be a float or None.

ACCEPTS: float or None

`set_animated(b)`

Set the artist's animation state.

Parameters

`b` [bool]

`set_antialiased(aa)`

Set the antialiasing state for rendering.

ACCEPTS: Boolean or sequence of booleans

`set_antialiaseds(aa)`

alias for `set_antialiased`

`set_array(A)`

Set the image array from numpy array A.

Parameters

`A` [ndarray]

`set_capstyle(cs)`

Set the capstyle for the collection. The capstyle can only be set globally for all elements in the collection

Parameters

`cs` [['butt' | 'round' | 'projecting']] The capstyle

`set_clim(vmin=None, vmax=None)`

set the norm limits for image scaling; if `vmin` is a length2 sequence, interpret it as (`vmin`, `vmax`) which is used to support `setp`

ACCEPTS: a length 2 sequence of floats; may be overridden in methods that have `vmin` and `vmax` kwargs.

`set_clip_box(clipbox)`

Set the artist's clip Bbox.

Parameters

`clipbox` [Bbox]

`set_clip_on(b)`

Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

`b` [bool]

`set_clip_path(path, transform=None)`

Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

`set_cmap(cmap)`

set the colormap for luminance data

ACCEPTS: a colormap or registered colormap name

`set_color(c)`

Set the color(s) of the LineCollection.

Parameters

`c` : Matplotlib color argument (all patches have same color), or a sequence or rgba tuples; if it is a sequence the patches will cycle through the sequence.

`set_contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

`picker` [callable]

`set_dashes(ls)`

alias for `set_linestyle`

`set_edgecolor(c)`

Set the edgecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have

same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'face', the edge color will always be the same as the face color. If it is 'none', the patch boundary will not be drawn.

ACCEPTS: matplotlib color spec or sequence of specs

`set_edgecolors(c)`
alias for `set_edgecolor`

`set_facecolor(c)`
Set the facecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'none', the patch will not be filled.

ACCEPTS: matplotlib color spec or sequence of specs

`set_facecolors(c)`
alias for `set_facecolor`

`set_figure(fig)`
Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`set_gid(gid)`
Sets the (group) id for the artist.

Parameters

`gid` [str]

`set_hatch(hatch)`
Set the hatching pattern

`hatch` can be one of:

/	- diagonal hatching
\	- back diagonal
	- vertical
-	- horizontal
+	- crossed
x	- crossed diagonal
o	- small circle
O	- large circle
.	- dots
*	- stars

Letters can be combined, in which case all the specified hatchings are done. If same letter repeats, it increases the density of hatching of that pattern.

Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Unlike other properties such as linewidth and colors, hatching can only be specified for the collection as a whole, not separately for each member.

ACCEPTS: ['/' | '\ ' | '|' | '- ' | '+ ' | 'x' | 'o' | 'O' | '?' | '*']

set_joinstyle(js)

Set the joinstyle for the collection. The joinstyle can only be set globally for all elements in the collection.

Parameters

js [['miter' | 'round' | 'bevel']] The joinstyle

set_label(s)

Set the label to s for auto legend.

Parameters

s [object] s will be converted to a string by calling `str` (unicode on Py2).

set_linestyle(ls)

Set the linestyle(s) for the collection.

linestyle	description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line

Alternatively a dash tuple of the following form can be provided:

(offset, onoffseq),

where onoffseq is an even length tuple of on and off ink in points.

ACCEPTS: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '--' | '-.' | ':' | 'None' | '' | '']

Parameters

ls [{ '-', '--', '-.', ':' } and more see description] The line style.

set_linestyles(ls)

alias for set_linestyle

set_linewidth(lw)

Set the linewidth(s) for the collection. lw can be a scalar or a sequence; if it is a sequence the patches will cycle through the sequence

ACCEPTS: float or sequence of floats

`set_linewidths(lw)`
alias for `set_linewidth`

`set_lw(lw)`
alias for `set_linewidth`

`set_norm(norm)`
Set the normalization instance.

Parameters

`norm` [Normalize]

`set_offset_position(offset_position)`
Set how offsets are applied. If `offset_position` is 'screen' (default) the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`set_offsets(offsets)`
Set the offsets for the collection. `offsets` can be a scalar or a sequence.

ACCEPTS: float or sequence of floats

`set_path_effects(path_effects)`
Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`set_paths(segments)`

`set_picker(picker)`
Set the epsilon for picking used by this artist

`picker` can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if `picker` is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if `picker` is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

`hit, props = picker(artist, mouseevent)`

to determine the hit test. if the mouse event is over the artist, return `hit=True` and `props` is a dictionary of properties you want added to the `PickEvent` attributes.

Parameters

`picker` [None or bool or float or callable]

`set_pickradius(pr)`

Set the pick radius used for containment tests.

Parameters

`d` [float] Pick radius, in points.

`set_rasterized(rasterized)`

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

`rasterized` [bool or None]

`set_segments(segments)`

`set_sketch_params(scale=None, length=None, randomness=None)`

Sets the sketch parameters.

Parameters

`scale` [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If `scale` is `None`, or not provided, no sketch filter will be provided.

`length` [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

`randomness` [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

`set_snap(snap)`

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

`snap` [bool or None]

`set_transform(t)`

Set the artist transform.

Parameters

`t` [Transform]

`set_url(url)`

Sets the url for the artist.

Parameters

`url` [str]

`set_urls(urls)`

Parameters

`urls` [List[str] or None]

`set_verts(segments)`

`set_visible(b)`

Set the artist's visibility.

Parameters

`b` [bool]

`set_zorder(level)`

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

`level` [float]

`stale`

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

`sticky_edges`

x and y sticky edge lists.

When performing autoscaling, if a data limit coincides with a value in the corresponding `sticky_edges` list, then no margin will be added—the view limit "sticks" to the edge. A typical usecase is histograms, where one usually expects no margin on the bottom edge (0) of the histogram.

This attribute cannot be assigned to; however, the x and y lists can be modified in place as needed.

Examples

```
>>> artist.sticky_edges.x[:] = (xmin, xmax)
>>> artist.sticky_edges.y[:] = (ymin, ymax)
```

`to_rgba(x, alpha=None, bytes=False, norm=True)`

Return a normalized rgba array corresponding to x.

In the normal case, x is a 1-D or 2-D sequence of scalars, and the corresponding ndarray of rgba values will be returned, based on the norm and colormap set for this ScalarMappable.

There is one special case, for handling images that are already rgb or rgba, such as might have been read from an image file. If x is an ndarray with 3 dimensions, and the last dimension is either 3 or 4, then it will be treated as an rgb or rgba array, and no mapping will be done. The array can be uint8, or it can be floating point with values in the 0-1 range; otherwise a `ValueError` will be raised. If it is a masked array, the mask will be ignored. If the last dimension is 3, the alpha kwarg (defaulting to 1) will be used to fill in the transparency. If the last dimension is 4, the alpha kwarg is ignored; it does not replace the pre-existing alpha. A `ValueError` will be raised if the third dimension is other than 3 or 4.

In either case, if bytes is `False` (default), the rgba array will be floats in the 0-1 range; if it is `True`, the returned rgba array will be uint8 in the 0 to 255 range.

If norm is `False`, no normalization of the input data is performed, and it is assumed to be in the range (0-1).

`update(props)`

Update this artist's properties from the dictionary prop.

`update_from(other)`

copy properties from other to self

`update_scalarmappable()`

If the scalar mappable array is not none, update colors from scalar data

`zorder = 0`

`class matplotlib.collections.PatchCollection(patches, match_original=False, **kwargs)`

Bases: `matplotlib.collections.Collection`

A generic collection of patches.

This makes it easier to assign a color map to a heterogeneous collection of patches.

This also may improve plotting speed, since `PatchCollection` will draw faster than a large number of patches.

`patches` a sequence of `Patch` objects. This list may include a heterogeneous assortment of different patch types.

`match_original` If True, use the colors and linewidths of the original patches. If False, new colors may be assigned by providing the standard collection arguments, `facecolor`, `edgecolor`, `linewidths`, `norm` or `cmap`.

If any of `edgecolors`, `facecolors`, `linewidths`, `antialiaseds` are None, they default to their `matplotlib.rcParams` patch setting, in sequence form.

The use of `ScalarMappable` is optional. If the `ScalarMappable` matrix `__A` is not None (i.e., a call to `set_array` has been made), at draw time a call to scalar mappable will be made to set the face colors.

`add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`add_checker(checker)`

Add an entry to a dictionary of boolean flags that are set to True when the mappable is changed.

`aname = 'Artist'`

`autoscale()`

Autoscale the scalar limits on the norm instance using the current array

`autoscale_None()`

Autoscale the scalar limits on the norm instance using the current array, changing only limits that are None

`axes`

The Axes instance the artist resides in, or None.

`changed()`

Call this whenever the mappable is changed to notify all the callbackSM listeners to the 'changed' signal

`check_update(checker)`

If mappable has changed since the last check, return True; else return False

`contains(mouseevent)`

Test whether the mouse event occurred in the collection.

Returns True | False, dict(ind=itemlist), where every item in itemlist contains the event.

`convert_xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`convert_yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`draw(renderer)`

Derived classes drawing method

`findobj(match=None, include_self=True)`
 Find artist objects.
 Recursively find all Artist instances contained in self.
 match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is `True` (default), include self in the list to be checked for a match.

`format_cursor_data(data)`
 Return cursor data string formatted.

`get_agg_filter()`
 Return filter function to be used for agg filter.

`get_alpha()`
 Return the alpha value used for blending - not supported on all backends

`get_animated()`
 Return the artist's animated state

`get_array()`
 Return the array

`get_capstyle()`

`get_children()`
 Return a list of the child Artist's this :class:`Artist` contains.

`get_clim()`
 return the min, max of the color limits for image scaling

`get_clip_box()`
 Return artist clipbox

`get_clip_on()`
 Return whether artist uses clipping

`get_clip_path()`
 Return artist clip path

`get_cmap()`
 return the colormap

`get_contains()`
 Return the `__contains` test used by the artist, or `None` for default.

`get_cursor_data(event)`
 Get the cursor data for a given event.

`get_dashes()`

`get_datalim(transData)`

`get_edgecolor()`

`get_edgecolors()`

`get_facecolor()`

`get_facecolors()`

`get_figure()`

Return the Figure instance the artist belongs to.

`get_fill()`

return whether fill is set

`get_gid()`

Returns the group id.

`get_hatch()`

Return the current hatching pattern.

`get_joinstyle()`

`get_label()`

Get the label used for this artist in the legend.

`get_linestyle()`

`get_linestyles()`

`get_linewidth()`

`get_linewidths()`

`get_offset_position()`

Returns how offsets are applied for the collection. If `offset_position` is 'screen', the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`get_offset_transform()`

`get_offsets()`

Return the offsets for the collection.

`get_path_effects()`

`get_paths()`

`get_picker()`

Return the picker object used by this artist.

`get_pickradius()`

`get_rasterized()`

Return whether the artist is to be rasterized.

`get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

`sketch_params` [tuple or `None`]

A 3-tuple with the following elements:

- `scale`: The amplitude of the wiggle perpendicular to the source line.
- `length`: The length of the wiggle along the line.
- `randomness`: The scale factor by which the length is shrunk or expanded.

May return `'None'` if no sketch parameters were set.

`get_snap()`

Returns the snap setting which may be:

- `True`: snap vertices to the nearest pixel center
- `False`: leave vertices as-is
- `None`: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`get_transform()`

Return the Transform instance used by this artist.

`get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`get_transforms()`

`get_url()`

Returns the url.

`get_urls()`

`get_visible()`

Return the artist's visibility

`get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`get_zorder()`

Return the artist's zorder.

`have_units()`

Return True if units are set on the x or y axes

`hitlist(event)`

Deprecated since version 2.2: The hitlist function was deprecated in version 2.2.

List the children of the artist which contain the mouse event event.

`is_figure_set()`

Deprecated since version 2.2: `artist.figure` is not None

Returns whether the artist is assigned to a Figure.

`is_transform_set()`

Returns True if Artist has a transform explicitly set.

`mouseover`

`pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`pick(mouseevent)`

Process pick event

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

`pickable()`

Return True if Artist is pickable.

`properties()`

return a dictionary mapping property name -> value for all Artist props

`remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`set(**kwargs)`

A property batch setter. Pass kwargs to set properties.

`set_agg_filter(filter_func)`

Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`set_alpha(alpha)`

Set the alpha transparencies of the collection. `alpha` must be a float or `None`.

ACCEPTS: float or `None`

`set_animated(b)`

Set the artist's animation state.

Parameters

`b` [bool]

`set_antialiased(aa)`

Set the antialiasing state for rendering.

ACCEPTS: Boolean or sequence of booleans

`set_antialiaseds(aa)`

alias for `set_antialiased`

`set_array(A)`

Set the image array from numpy array `A`.

Parameters

`A` [ndarray]

`set_capstyle(cs)`

Set the capstyle for the collection. The capstyle can only be set globally for all elements in the collection

Parameters

`cs` `['butt' | 'round' | 'projecting']` The capstyle

`set_clim(vmin=None, vmax=None)`

set the norm limits for image scaling; if `vmin` is a length2 sequence, interpret it as (`vmin`, `vmax`) which is used to support `setp`

ACCEPTS: a length 2 sequence of floats; may be overridden in methods that have `vmin` and `vmax` kwargs.

`set_clip_box(clipbox)`

Set the artist's clip Bbox.

Parameters

`clipbox` `[Bbox]`

`set_clip_on(b)`

Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

`b` `[bool]`

`set_clip_path(path, transform=None)`

Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: `[(Path, Transform) | Patch | None]`

`set_cmap(cmap)`

set the colormap for luminance data

ACCEPTS: a colormap or registered colormap name

`set_color(c)`

Set both the edgecolor and the facecolor.

ACCEPTS: matplotlib color arg or sequence of rgba tuples

See also:

`set_facecolor()`, `set_edgecolor()` For setting the edge or face color individually.

`set_contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

`picker` [callable]

`set_dashes(ls)`

alias for `set_linestyle`

`set_edgecolor(c)`

Set the edgecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'face', the edge color will always be the same as the face color. If it is 'none', the patch boundary will not be drawn.

ACCEPTS: matplotlib color spec or sequence of specs

`set_edgecolors(c)`

alias for `set_edgecolor`

`set_facecolor(c)`

Set the facecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'none', the patch will not be filled.

ACCEPTS: matplotlib color spec or sequence of specs

`set_facecolors(c)`

alias for `set_facecolor`

`set_figure(fig)`

Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`set_gid(gid)`

Sets the (group) id for the artist.

Parameters

`gid` [str]

`set_hatch(hatch)`
Set the hatching pattern

hatch can be one of:

/	- diagonal hatching
\	- back diagonal
	- vertical
-	- horizontal
+	- crossed
x	- crossed diagonal
o	- small circle
O	- large circle
.	- dots
*	- stars

Letters can be combined, in which case all the specified hatchings are done. If same letter repeats, it increases the density of hatching of that pattern.

Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Unlike other properties such as linewidth and colors, hatching can only be specified for the collection as a whole, not separately for each member.

ACCEPTS: ['/' | '\' | '|' | '-' | '+' | 'x' | 'o' | 'O' | '.' | '*']

`set_joinstyle(js)`
Set the joinstyle for the collection. The joinstyle can only be set globally for all elements in the collection.

Parameters

js [['miter' | 'round' | 'bevel']] The joinstyle

`set_label(s)`
Set the label to s for auto legend.

Parameters

s [object] s will be converted to a string by calling `str` (unicode on Py2).

`set_linestyle(ls)`
Set the linestyle(s) for the collection.

linestyle	description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line

Alternatively a dash tuple of the following form can be provided:

(offset, onoffseq),

where onoffseq is an even length tuple of on and off ink in points.

ACCEPTS: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '--' |
 '-.' | ':' | 'None' | '' | '']

Parameters

ls [{ '-', '-', '-.', ':'}] and more see description] The line style.

set_linestyles(ls)

alias for set_linestyle

set_linewidth(lw)

Set the linewidth(s) for the collection. lw can be a scalar or a sequence; if it is a sequence the patches will cycle through the sequence

ACCEPTS: float or sequence of floats

set_linewidths(lw)

alias for set_linewidth

set_lw(lw)

alias for set_linewidth

set_norm(norm)

Set the normalization instance.

Parameters

norm [Normalize]

set_offset_position(offset_position)

Set how offsets are applied. If offset_position is 'screen' (default) the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If offset_position is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

set_offsets(offsets)

Set the offsets for the collection. offsets can be a scalar or a sequence.

ACCEPTS: float or sequence of floats

set_path_effects(path_effects)

Set the path effects.

Parameters

path_effects [AbstractPathEffect]

set_paths(patches)

`set_picker(picker)`

Set the epsilon for picking used by this artist

`picker` can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if `picker` is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if `picker` is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

to determine the hit test. if the mouse event is over the artist, return `hit=True` and `props` is a dictionary of properties you want added to the `PickEvent` attributes.

Parameters

`picker` [None or bool or float or callable]

`set_pickradius(pr)`

Set the pick radius used for containment tests.

Parameters

`d` [float] Pick radius, in points.

`set_rasterized(rasterized)`

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

`rasterized` [bool or None]

`set_sketch_params(scale=None, length=None, randomness=None)`

Sets the sketch parameters.

Parameters

`scale` [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If `scale` is `None`, or not provided, no sketch filter will be provided.

`length` [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

randomness [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

set_snap(snap)

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

snap [bool or None]

set_transform(t)

Set the artist transform.

Parameters

t [Transform]

set_url(url)

Sets the url for the artist.

Parameters

url [str]

set_urls(urls)

Parameters

urls [List[str] or None]

set_visible(b)

Set the artist's visibility.

Parameters

b [bool]

set_zorder(level)

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

level [float]

stale

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

`sticky_edges`

x and y sticky edge lists.

When performing autoscaling, if a data limit coincides with a value in the corresponding `sticky_edges` list, then no margin will be added—the view limit “sticks” to the edge. A typical usecase is histograms, where one usually expects no margin on the bottom edge (0) of the histogram.

This attribute cannot be assigned to; however, the x and y lists can be modified in place as needed.

Examples

```
>>> artist.sticky_edges.x[:] = (xmin, xmax)
>>> artist.sticky_edges.y[:] = (ymin, ymax)
```

`to_rgba(x, alpha=None, bytes=False, norm=True)`

Return a normalized rgba array corresponding to x.

In the normal case, x is a 1-D or 2-D sequence of scalars, and the corresponding ndarray of rgba values will be returned, based on the norm and colormap set for this `ScalarMappable`.

There is one special case, for handling images that are already rgb or rgba, such as might have been read from an image file. If x is an ndarray with 3 dimensions, and the last dimension is either 3 or 4, then it will be treated as an rgb or rgba array, and no mapping will be done. The array can be uint8, or it can be floating point with values in the 0-1 range; otherwise a `ValueError` will be raised. If it is a masked array, the mask will be ignored. If the last dimension is 3, the alpha kwarg (defaulting to 1) will be used to fill in the transparency. If the last dimension is 4, the alpha kwarg is ignored; it does not replace the pre-existing alpha. A `ValueError` will be raised if the third dimension is other than 3 or 4.

In either case, if `bytes` is `False` (default), the rgba array will be floats in the 0-1 range; if it is `True`, the returned rgba array will be uint8 in the 0 to 255 range.

If `norm` is `False`, no normalization of the input data is performed, and it is assumed to be in the range (0-1).

`update(props)`

Update this artist’s properties from the dictionary prop.

`update_from(other)`

copy properties from other to self

`update_scalarmappable()`

If the scalar mappable array is not none, update colors from scalar data

`zorder = 0`

```
class matplotlib.collections.PathCollection(paths, sizes=None, **kwargs)
    Bases: matplotlib.collections._CollectionWithSizes
```

This is the most basic Collection subclass.

paths is a sequence of matplotlib.path.Path instances.

Valid Collection keyword arguments:

- edgecolors: None
- facecolors: None
- linewidths: None
- antialiaseds: None
- offsets: None
- transOffset: transforms.IdentityTransform()
- norm: None (optional for matplotlib.cm.ScalarMappable)
- cmap: None (optional for matplotlib.cm.ScalarMappable)

offsets and transOffset are used to translate the patch after rendering (default no offsets)

If any of edgecolors, facecolors, linewidths, antialiaseds are None, they default to their matplotlib.rcParams patch setting, in sequence form.

```
add_callback(func)
```

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with remove_callback() later.

```
add_checker(checker)
```

Add an entry to a dictionary of boolean flags that are set to True when the mappable is changed.

```
aname = 'Artist'
```

```
autoscale()
```

Autoscale the scalar limits on the norm instance using the current array

```
autoscale_None()
```

Autoscale the scalar limits on the norm instance using the current array, changing only limits that are None

```
axes
```

The Axes instance the artist resides in, or None.

```
changed()
```

Call this whenever the mappable is changed to notify all the callbackSM listeners to the 'changed' signal

`check_update(checker)`

If mappable has changed since the last check, return True; else return False

`contains(mouseevent)`

Test whether the mouse event occurred in the collection.

Returns True | False, dict(ind=itemlist), where every item in itemlist contains the event.

`convert_xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`convert_yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`draw(renderer)`

Derived classes drawing method

`findobj(match=None, include_self=True)`

Find artist objects.

Recursively find all Artist instances contained in self.

match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is True (default), include self in the list to be checked for a match.

`format_cursor_data(data)`

Return cursor data string formatted.

`get_agg_filter()`

Return filter function to be used for agg filter.

`get_alpha()`

Return the alpha value used for blending - not supported on all backends

`get_animated()`

Return the artist's animated state

`get_array()`

Return the array

`get_capstyle()`

`get_children()`

Return a list of the child Artist's this :class:`Artist` contains.

`get_clim()`

return the min, max of the color limits for image scaling

`get_clip_box()`

Return artist clipbox

`get_clip_on()`
Return whether artist uses clipping

`get_clip_path()`
Return artist clip path

`get_cmap()`
return the colormap

`get_contains()`
Return the `__contains` test used by the artist, or `None` for default.

`get_cursor_data(event)`
Get the cursor data for a given event.

`get_dashes()`

`get_datalim(transData)`

`get_edgecolor()`

`get_edgecolors()`

`get_facecolor()`

`get_facecolors()`

`get_figure()`
Return the Figure instance the artist belongs to.

`get_fill()`
return whether fill is set

`get_gid()`
Returns the group id.

`get_hatch()`
Return the current hatching pattern.

`get_joinstyle()`

`get_label()`
Get the label used for this artist in the legend.

`get_linestyle()`

`get_linestyles()`

`get_linewidth()`

`get_linewidths()`

`get_offset_position()`

Returns how offsets are applied for the collection. If `offset_position` is 'screen', the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`get_offset_transform()`

`get_offsets()`

Return the offsets for the collection.

`get_path_effects()`

`get_paths()`

`get_picker()`

Return the picker object used by this artist.

`get_pickradius()`

`get_rasterized()`

Return whether the artist is to be rasterized.

`get_sizes()`

Returns the sizes of the elements in the collection. The value represents the 'area' of the element.

Returns

sizes [array] The 'area' of each element.

`get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

sketch_params [tuple or `None`]

A 3-tuple with the following elements:

- scale: The amplitude of the wiggle perpendicular to the source line.
- length: The length of the wiggle along the line.
- randomness: The scale factor by which the length is shrunk or expanded.

May return 'None' if no sketch parameters were set.

`get_snap()`

Returns the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`get_transform()`

Return the Transform instance used by this artist.

`get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`get_transforms()`

`get_url()`

Returns the url.

`get_urls()`

`get_visible()`

Return the artist's visibility

`get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`get_zorder()`

Return the artist's zorder.

`have_units()`

Return True if units are set on the x or y axes

`hitlist(event)`

Deprecated since version 2.2: The hitlist function was deprecated in version 2.2.

List the children of the artist which contain the mouse event event.

`is_figure_set()`

Deprecated since version 2.2: `artist.figure` is not None

Returns whether the artist is assigned to a Figure.

`is_transform_set()`

Returns True if Artist has a transform explicitly set.

`mouseover`

`pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`pick(mouseevent)`

Process pick event

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

`pickable()`

Return True if Artist is pickable.

`properties()`

return a dictionary mapping property name -> value for all Artist props

`remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`set(**kwargs)`

A property batch setter. Pass kwargs to set properties.

`set_agg_filter(filter_func)`

Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`set_alpha(alpha)`

Set the alpha transparencies of the collection. alpha must be a float or None.

ACCEPTS: float or None

`set_animated(b)`

Set the artist's animation state.

Parameters

b [bool]

set_antialiased(aa)

Set the antialiasing state for rendering.

ACCEPTS: Boolean or sequence of booleans

set_antialiaseds(aa)

alias for set_antialiased

set_array(A)

Set the image array from numpy array A.

Parameters

A [ndarray]

set_capstyle(cs)

Set the capstyle for the collection. The capstyle can only be set globally for all elements in the collection

Parameters

cs [['butt' | 'round' | 'projecting']] The capstyle

set_clim(vmin=None, vmax=None)

set the norm limits for image scaling; if vmin is a length2 sequence, interpret it as (vmin, vmax) which is used to support setp

ACCEPTS: a length 2 sequence of floats; may be overridden in methods that have vmin and vmax kwargs.

set_clip_box(clipbox)

Set the artist's clip Bbox.

Parameters

clipbox [Bbox]

set_clip_on(b)

Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

b [bool]

set_clip_path(path, transform=None)

Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or

- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

`set_cmap(cmap)`

set the colormap for luminance data

ACCEPTS: a colormap or registered colormap name

`set_color(c)`

Set both the edgecolor and the facecolor.

ACCEPTS: matplotlib color arg or sequence of rgba tuples

See also:

`set_facecolor()`, `set_edgecolor()` For setting the edge or face color individually.

`set_contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

`picker` [callable]

`set_dashes(ls)`

alias for `set_linestyle`

`set_edgecolor(c)`

Set the edgecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'face', the edge color will always be the same as the face color. If it is 'none', the patch boundary will not be drawn.

ACCEPTS: matplotlib color spec or sequence of specs

`set_edgecolors(c)`

alias for `set_edgecolor`

`set_facecolor(c)`

Set the facecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'none', the patch will not be filled.

ACCEPTS: matplotlib color spec or sequence of specs

`set_facecolors(c)`
alias for `set_facecolor`

`set_figure(fig)`
Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`set_gid(gid)`
Sets the (group) id for the artist.

Parameters

`gid` [str]

`set_hatch(hatch)`
Set the hatching pattern

hatch can be one of:

/	- diagonal hatching
\	- back diagonal
	- vertical
-	- horizontal
+	- crossed
x	- crossed diagonal
o	- small circle
O	- large circle
.	- dots
*	- stars

Letters can be combined, in which case all the specified hatchings are done. If same letter repeats, it increases the density of hatching of that pattern.

Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Unlike other properties such as linewidth and colors, hatching can only be specified for the collection as a whole, not separately for each member.

ACCEPTS: ['/' | '\ ' | '|' | '-' | '+' | 'x' | 'o' | 'O' | '.' | '*']

`set_joinstyle(js)`
Set the joinstyle for the collection. The joinstyle can only be set globally for all elements in the collection.

Parameters

`js` [['miter' | 'round' | 'bevel']] The joinstyle

`set_label(s)`
Set the label to s for auto legend.

Parameters

s [object] s will be converted to a string by calling `str` (unicode on Py2).

`set_linestyle(ls)`

Set the linestyle(s) for the collection.

linestyle	description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line

Alternatively a dash tuple of the following form can be provided:

(offset, onoffseq),

where onoffseq is an even length tuple of on and off ink in points.

ACCEPTS: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '--' |
'-.' | ':' | 'None' | ' ' | '']

Parameters

ls [{ '-', '-.', ':-', ':'}] and more see description] The line style.

`set_linestyles(ls)`

alias for `set_linestyle`

`set_linewidth(lw)`

Set the linewidth(s) for the collection. `lw` can be a scalar or a sequence; if it is a sequence the patches will cycle through the sequence

ACCEPTS: float or sequence of floats

`set_linewidths(lw)`

alias for `set_linewidth`

`set_lw(lw)`

alias for `set_linewidth`

`set_norm(norm)`

Set the normalization instance.

Parameters

norm [Normalize]

`set_offset_position(offset_position)`

Set how offsets are applied. If `offset_position` is 'screen' (default) the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`set_offsets(offsets)`

Set the offsets for the collection. `offsets` can be a scalar or a sequence.

ACCEPTS: float or sequence of floats

`set_path_effects(path_effects)`

Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`set_paths(paths)`

`set_picker(picker)`

Set the epsilon for picking used by this artist

`picker` can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if `picker` is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if `picker` is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

<code>hit, props = picker(artist, mouseevent)</code>
--

to determine the hit test. if the mouse event is over the artist, return `hit=True` and `props` is a dictionary of properties you want added to the `PickEvent` attributes.

Parameters

`picker` [None or bool or float or callable]

`set_pickradius(pr)`

Set the pick radius used for containment tests.

Parameters

`d` [float] Pick radius, in points.

`set_rasterized(rasterized)`

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

rasterized [bool or None]

set_sizes(sizes, dpi=72.0)

Set the sizes of each member of the collection.

Parameters

sizes [ndarray or None] The size to set for each element of the collection.
The value is the 'area' of the element.

dpi [float] The dpi of the canvas. Defaults to 72.0.

set_sketch_params(scale=None, length=None, randomness=None)

Sets the sketch parameters.

Parameters

scale [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If scale is `None`, or not provided, no sketch filter will be provided.

length [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

randomness [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

set_snap(snap)

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

snap [bool or None]

set_transform(t)

Set the artist transform.

Parameters

t [Transform]

set_url(url)

Sets the url for the artist.

Parameters

url [str]

set_urls(urls)

Parameters

urls [List[str] or None]

set_visible(b)

Set the artist's visibility.

Parameters

b [bool]

set_zorder(level)

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

level [float]

stale

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

sticky_edges

x and y sticky edge lists.

When performing autoscaling, if a data limit coincides with a value in the corresponding sticky_edges list, then no margin will be added—the view limit "sticks" to the edge. A typical usecase is histograms, where one usually expects no margin on the bottom edge (0) of the histogram.

This attribute cannot be assigned to; however, the x and y lists can be modified in place as needed.

Examples

```
>>> artist.sticky_edges.x[:] = (xmin, xmax)
>>> artist.sticky_edges.y[:] = (ymin, ymax)
```

to_rgba(x, alpha=None, bytes=False, norm=True)

Return a normalized rgba array corresponding to x.

In the normal case, x is a 1-D or 2-D sequence of scalars, and the corresponding ndarray of rgba values will be returned, based on the norm and colormap set for this ScalarMappable.

There is one special case, for handling images that are already rgb or rgba, such as might have been read from an image file. If x is an ndarray with 3 dimensions, and the last dimension is either 3 or 4, then it will be treated as an rgb or rgba array, and no mapping will be done. The array can be uint8, or it can be floating point with values in the 0-1 range; otherwise a ValueError will be raised. If it is a masked array, the mask will be ignored. If the last dimension is 3, the alpha kwarg (defaulting to 1) will be used to fill in the transparency. If the last dimension is 4, the alpha kwarg is ignored; it does

not replace the pre-existing alpha. A `ValueError` will be raised if the third dimension is other than 3 or 4.

In either case, if `bytes` is `False` (default), the `rgba` array will be floats in the 0-1 range; if it is `True`, the returned `rgba` array will be `uint8` in the 0 to 255 range.

If `norm` is `False`, no normalization of the input data is performed, and it is assumed to be in the range (0-1).

`update(props)`

Update this artist's properties from the dictionary `prop`.

`update_from(other)`

copy properties from `other` to self

`update_scalarmappable()`

If the scalar mappable array is not none, update colors from scalar data

`zorder = 0`

`class matplotlib.collections.PolyCollection(verts, sizes=None, closed=True, **kwargs)`

Bases: `matplotlib.collections._CollectionWithSizes`

`verts` is a sequence of (`verts0`, `verts1`, ...) where `verts_i` is a sequence of xy tuples of vertices, or an equivalent `numpy` array of shape `(nv, 2)`.

`sizes` is `None` (default) or a sequence of floats that scale the corresponding `verts_i`. The scaling is applied before the Artist master transform; if the latter is an identity transform, then the overall scaling is such that if `verts_i` specify a unit square, then `sizes_i` is the area of that square in `points^2`. If `len(sizes) < nv`, the additional values will be taken cyclically from the array.

`closed`, when `True`, will explicitly close the polygon.

Valid Collection keyword arguments:

- `edgecolors`: `None`
- `facecolors`: `None`
- `linewidths`: `None`
- `antialiaseds`: `None`
- `offsets`: `None`
- `transOffset`: `transforms.IdentityTransform()`
- `norm`: `None` (optional for `matplotlib.cm.ScalarMappable`)
- `cmap`: `None` (optional for `matplotlib.cm.ScalarMappable`)

`offsets` and `transOffset` are used to translate the patch after rendering (default no offsets)

If any of `edgecolors`, `facecolors`, `linewidths`, `antialiaseds` are `None`, they default to their `matplotlib.rcParams` patch setting, in sequence form.

`add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`add_checker(checker)`

Add an entry to a dictionary of boolean flags that are set to True when the mappable is changed.

`aname = 'Artist'`

`autoscale()`

Autoscale the scalar limits on the norm instance using the current array

`autoscale_None()`

Autoscale the scalar limits on the norm instance using the current array, changing only limits that are None

`axes`

The Axes instance the artist resides in, or None.

`changed()`

Call this whenever the mappable is changed to notify all the callbackSM listeners to the 'changed' signal

`check_update(checker)`

If mappable has changed since the last check, return True; else return False

`contains(mouseevent)`

Test whether the mouse event occurred in the collection.

Returns True | False, dict(ind=itemlist), where every item in itemlist contains the event.

`convert_xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`convert_yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`draw(renderer)`

Derived classes drawing method

`findobj(match=None, include_self=True)`

Find artist objects.

Recursively find all Artist instances contained in self.

match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is True (default), include self in the list to be checked for a match.

`format_cursor_data(data)`
Return cursor data string formatted.

`get_agg_filter()`
Return filter function to be used for agg filter.

`get_alpha()`
Return the alpha value used for blending - not supported on all backends

`get_animated()`
Return the artist's animated state

`get_array()`
Return the array

`get_capstyle()`

`get_children()`
Return a list of the child Artist's this :class:`Artist` contains.

`get_clim()`
return the min, max of the color limits for image scaling

`get_clip_box()`
Return artist clipbox

`get_clip_on()`
Return whether artist uses clipping

`get_clip_path()`
Return artist clip path

`get_cmap()`
return the colormap

`get_contains()`
Return the `__contains` test used by the artist, or `None` for default.

`get_cursor_data(event)`
Get the cursor data for a given event.

`get_dashes()`

`get_datalim(transData)`

`get_edgecolor()`

`get_edgecolors()`

`get_facecolor()`

`get_facecolors()`

`get_figure()`

Return the Figure instance the artist belongs to.

`get_fill()`

return whether fill is set

`get_gid()`

Returns the group id.

`get_hatch()`

Return the current hatching pattern.

`get_joinstyle()`

`get_label()`

Get the label used for this artist in the legend.

`get_linestyle()`

`get_linestyles()`

`get_linewidth()`

`get_linewidths()`

`get_offset_position()`

Returns how offsets are applied for the collection. If `offset_position` is 'screen', the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`get_offset_transform()`

`get_offsets()`

Return the offsets for the collection.

`get_path_effects()`

`get_paths()`

`get_picker()`

Return the picker object used by this artist.

`get_pickradius()`

`get_rasterized()`

Return whether the artist is to be rasterized.

`get_sizes()`

Returns the sizes of the elements in the collection. The value represents the 'area' of the element.

Returns

sizes [array] The 'area' of each element.

`get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

sketch_params [tuple or `None`]

A 3-tuple with the following elements:

- scale: The amplitude of the wiggle perpendicular to the source line.
- length: The length of the wiggle along the line.
- randomness: The scale factor by which the length is shrunk or expanded.

May return 'None' if no sketch parameters were set.

`get_snap()`

Returns the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`get_transform()`

Return the Transform instance used by this artist.

`get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`get_transforms()`

`get_url()`

Returns the url.

`get_urls()`

`get_visible()`

Return the artist's visibility

`get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`get_zorder()`

Return the artist's zorder.

`have_units()`

Return True if units are set on the x or y axes

`hitlist(event)`

Deprecated since version 2.2: The hitlist function was deprecated in version 2.2.

List the children of the artist which contain the mouse event event.

`is_figure_set()`

Deprecated since version 2.2: `artist.figure` is not None

Returns whether the artist is assigned to a Figure.

`is_transform_set()`

Returns True if Artist has a transform explicitly set.

`mouseover`

`pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`pick(mouseevent)`

Process pick event

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

`pickable()`

Return True if Artist is pickable.

`properties()`

return a dictionary mapping property name -> value for all Artist props

`remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`set(**kwargs)`

A property batch setter. Pass kwargs to set properties.

`set_agg_filter(filter_func)`

Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`set_alpha(alpha)`

Set the alpha transparencies of the collection. alpha must be a float or None.

ACCEPTS: float or None

`set_animated(b)`

Set the artist's animation state.

Parameters

`b` [bool]

`set_antialiased(aa)`

Set the antialiasing state for rendering.

ACCEPTS: Boolean or sequence of booleans

`set_antialiaseds(aa)`

alias for `set_antialiased`

`set_array(A)`

Set the image array from numpy array A.

Parameters

`A` [ndarray]

`set_capstyle(cs)`

Set the capstyle for the collection. The capstyle can only be set globally for all elements in the collection

Parameters

`cs` [['butt' | 'round' | 'projecting']] The capstyle

`set_clim(vmin=None, vmax=None)`

set the norm limits for image scaling; if `vmin` is a length2 sequence, interpret it as (`vmin`, `vmax`) which is used to support `setp`

ACCEPTS: a length 2 sequence of floats; may be overridden in methods that have vmin and vmax kwargs.

`set_clip_box(clipbox)`

Set the artist's clip Bbox.

Parameters

clipbox [Bbox]

`set_clip_on(b)`

Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

b [bool]

`set_clip_path(path, transform=None)`

Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

`set_cmap(cmap)`

set the colormap for luminance data

ACCEPTS: a colormap or registered colormap name

`set_color(c)`

Set both the edgecolor and the facecolor.

ACCEPTS: matplotlib color arg or sequence of rgba tuples

See also:

`set_facecolor()`, `set_edgecolor()` For setting the edge or face color individually.

`set_contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return hit = True and props is a dictionary of properties you want returned with the contains test.

Parameters

picker [callable]

set__dashes(ls)

alias for set__linestyle

set__edgecolor(c)

Set the edgecolor(s) of the collection. c can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If c is 'face', the edge color will always be the same as the face color. If it is 'none', the patch boundary will not be drawn.

ACCEPTS: matplotlib color spec or sequence of specs

set__edgecolors(c)

alias for set__edgecolor

set__facecolor(c)

Set the facecolor(s) of the collection. c can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If c is 'none', the patch will not be filled.

ACCEPTS: matplotlib color spec or sequence of specs

set__facecolors(c)

alias for set__facecolor

set__figure(fig)

Set the Figure instance the artist belongs to.

Parameters

fig [Figure]

set__gid(gid)

Sets the (group) id for the artist.

Parameters

gid [str]

set__hatch(hatch)

Set the hatching pattern

hatch can be one of:

/	- diagonal hatching
\	- back diagonal
	- vertical
-	- horizontal
+	- crossed

(continues on next page)

(continued from previous page)

```

x - crossed diagonal
o - small circle
O - large circle
. - dots
* - stars

```

Letters can be combined, in which case all the specified hatchings are done. If same letter repeats, it increases the density of hatching of that pattern.

Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Unlike other properties such as linewidth and colors, hatching can only be specified for the collection as a whole, not separately for each member.

ACCEPTS: ['/' | '\ ' | ' ' | ' ' | ' ' | ' ' | 'x' | 'o' | 'O' | '?' | '*']

`set_joinstyle(js)`

Set the joinstyle for the collection. The joinstyle can only be set globally for all elements in the collection.

Parameters

`js` [['miter' | 'round' | 'bevel']] The joinstyle

`set_label(s)`

Set the label to `s` for auto legend.

Parameters

`s` [object] `s` will be converted to a string by calling `str` (unicode on Py2).

`set_linestyle(ls)`

Set the linestyle(s) for the collection.

linestyle	description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line

Alternatively a dash tuple of the following form can be provided:

```
(offset, onoffseq),
```

where `onoffseq` is an even length tuple of on and off ink in points.

ACCEPTS: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '--' | '-.' | ':' | 'None' | '' | '']

Parameters

`ls` [{ '-' | '--' | '-.' | ':' } and more see description] The line style.

`set_linestyles(ls)`

alias for `set_linestyle`

`set_linewidth(lw)`

Set the linewidth(s) for the collection. `lw` can be a scalar or a sequence; if it is a sequence the patches will cycle through the sequence

ACCEPTS: float or sequence of floats

`set_linewidths(lw)`

alias for `set_linewidth`

`set_lw(lw)`

alias for `set_linewidth`

`set_norm(norm)`

Set the normalization instance.

Parameters

`norm` [Normalize]

`set_offset_position(offset_position)`

Set how offsets are applied. If `offset_position` is 'screen' (default) the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`set_offsets(offsets)`

Set the offsets for the collection. `offsets` can be a scalar or a sequence.

ACCEPTS: float or sequence of floats

`set_path_effects(path_effects)`

Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`set_paths(verts, closed=True)`

This allows one to delay initialization of the vertices.

`set_picker(picker)`

Set the epsilon for picking used by this artist

`picker` can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if `picker` is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data

to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event

- A function: if picker is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

to determine the hit test. if the mouse event is over the artist, return hit=True and props is a dictionary of properties you want added to the PickEvent attributes.

Parameters

picker [None or bool or float or callable]

set_pickradius(pr)

Set the pick radius used for containment tests.

Parameters

d [float] Pick radius, in points.

set_rasterized(rasterized)

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

rasterized [bool or None]

set_sizes(sizes, dpi=72.0)

Set the sizes of each member of the collection.

Parameters

sizes [ndarray or None] The size to set for each element of the collection.
The value is the 'area' of the element.

dpi [float] The dpi of the canvas. Defaults to 72.0.

set_sketch_params(scale=None, length=None, randomness=None)

Sets the sketch parameters.

Parameters

scale [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If scale is `None`, or not provided, no sketch filter will be provided.

length [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

randomness [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

`set_snap(snap)`

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

`snap` [bool or None]

`set_transform(t)`

Set the artist transform.

Parameters

`t` [Transform]

`set_url(url)`

Sets the url for the artist.

Parameters

`url` [str]

`set_urls(urls)`

Parameters

`urls` [List[str] or None]

`set_verts(verts, closed=True)`

This allows one to delay initialization of the vertices.

`set_verts_and_codes(verts, codes)`

This allows one to initialize vertices with path codes.

`set_visible(b)`

Set the artist's visibility.

Parameters

`b` [bool]

`set_zorder(level)`

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

`level` [float]

`stale`

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

`sticky_edges`

x and y sticky edge lists.

When performing autoscaling, if a data limit coincides with a value in the corresponding `sticky_edges` list, then no margin will be added—the view limit “sticks” to the edge. A typical usecase is histograms, where one usually expects no margin on the bottom edge (0) of the histogram.

This attribute cannot be assigned to; however, the x and y lists can be modified in place as needed.

Examples

```
>>> artist.sticky_edges.x[:] = (xmin, xmax)
>>> artist.sticky_edges.y[:] = (ymin, ymax)
```

`to_rgba(x, alpha=None, bytes=False, norm=True)`

Return a normalized rgba array corresponding to x.

In the normal case, x is a 1-D or 2-D sequence of scalars, and the corresponding ndarray of rgba values will be returned, based on the norm and colormap set for this `ScalarMappable`.

There is one special case, for handling images that are already rgb or rgba, such as might have been read from an image file. If x is an ndarray with 3 dimensions, and the last dimension is either 3 or 4, then it will be treated as an rgb or rgba array, and no mapping will be done. The array can be uint8, or it can be floating point with values in the 0-1 range; otherwise a `ValueError` will be raised. If it is a masked array, the mask will be ignored. If the last dimension is 3, the alpha kwarg (defaulting to 1) will be used to fill in the transparency. If the last dimension is 4, the alpha kwarg is ignored; it does not replace the pre-existing alpha. A `ValueError` will be raised if the third dimension is other than 3 or 4.

In either case, if bytes is `False` (default), the rgba array will be floats in the 0-1 range; if it is `True`, the returned rgba array will be uint8 in the 0 to 255 range.

If norm is `False`, no normalization of the input data is performed, and it is assumed to be in the range (0-1).

`update(props)`

Update this artist’s properties from the dictionary prop.

`update_from(other)`

copy properties from other to self

`update_scalarmappable()`

If the scalar mappable array is not none, update colors from scalar data

`zorder = 0`

```
class matplotlib.collections.QuadMesh(meshWidth, meshHeight, coordinates, an-
                                     tialiased=True, shading='flat', **kwargs)
```

Bases: matplotlib.collections.Collection

Class for the efficient drawing of a quadrilateral mesh.

A quadrilateral mesh consists of a grid of vertices. The dimensions of this array are $(\text{meshWidth} + 1, \text{meshHeight} + 1)$. Each vertex in the mesh has a different set of "mesh coordinates" representing its position in the topology of the mesh. For any values (m, n) such that $0 \leq m \leq \text{meshWidth}$ and $0 \leq n \leq \text{meshHeight}$, the vertices at mesh coordinates (m, n) , $(m, n + 1)$, $(m + 1, n + 1)$, and $(m + 1, n)$ form one of the quadrilaterals in the mesh. There are thus $(\text{meshWidth} * \text{meshHeight})$ quadrilaterals in the mesh. The mesh need not be regular and the polygons need not be convex.

A quadrilateral mesh is represented by a $(2 \times ((\text{meshWidth} + 1) * (\text{meshHeight} + 1)))$ numpy array `coordinates`, where each row is the x and y coordinates of one of the vertices. To define the function that maps from a data point to its corresponding color, use the `set_cmap()` method. Each of these arrays is indexed in row-major order by the mesh coordinates of the vertex (or the mesh coordinates of the lower left vertex, in the case of the colors).

For example, the first entry in `coordinates` is the coordinates of the vertex at mesh coordinates $(0, 0)$, then the one at $(0, 1)$, then at $(0, 2) \dots (0, \text{meshHeight})$, $(1, 0)$, $(1, 1)$, and so on.

`shading` may be 'flat', or 'gouraud'

`add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`add_checker(checker)`

Add an entry to a dictionary of boolean flags that are set to True when the mappable is changed.

`aname = 'Artist'`

`autoscale()`

Autoscale the scalar limits on the norm instance using the current array

`autoscale_None()`

Autoscale the scalar limits on the norm instance using the current array, changing only limits that are None

`axes`

The Axes instance the artist resides in, or None.

`changed()`

Call this whenever the mappable is changed to notify all the callbackSM listeners to the 'changed' signal

`check_update(checker)`

If mappable has changed since the last check, return True; else return False

`contains(mouseevent)`

Test whether the mouse event occurred in the collection.

Returns True | False, dict(ind=itemlist), where every item in itemlist contains the event.

`static convert__mesh__to__paths(meshWidth, meshHeight, coordinates)`

Converts a given mesh into a sequence of `matplotlib.path.Path` objects for easier rendering by backends that do not directly support quadmeshes.

This function is primarily of use to backend implementers.

`convert__mesh__to__triangles(meshWidth, meshHeight, coordinates)`

Converts a given mesh into a sequence of triangles, each point with its own color. This is useful for experiments using `draw__qouraud__triangle`.

`convert__xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`convert__yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`draw(renderer)`

Derived classes drawing method

`findobj(match=None, include_self=True)`

Find artist objects.

Recursively find all Artist instances contained in self.

match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is True (default), include self in the list to be checked for a match.

`format__cursor__data(data)`

Return cursor data string formatted.

`get__agg__filter()`

Return filter function to be used for agg filter.

`get__alpha()`

Return the alpha value used for blending - not supported on all backends

`get__animated()`

Return the artist's animated state

`get__array()`

Return the array

`get__capstyle()`

`get_children()`
Return a list of the child Artist's this :class:`Artist` contains.

`get_clim()`
return the min, max of the color limits for image scaling

`get_clip_box()`
Return artist clipbox

`get_clip_on()`
Return whether artist uses clipping

`get_clip_path()`
Return artist clip path

`get_cmap()`
return the colormap

`get_contains()`
Return the `_contains` test used by the artist, or `None` for default.

`get_cursor_data(event)`
Get the cursor data for a given event.

`get_dashes()`

`get_datalim(transData)`

`get_edgecolor()`

`get_edgecolors()`

`get_facecolor()`

`get_facecolors()`

`get_figure()`
Return the Figure instance the artist belongs to.

`get_fill()`
return whether fill is set

`get_gid()`
Returns the group id.

`get_hatch()`
Return the current hatching pattern.

`get_joinstyle()`

`get_label()`

Get the label used for this artist in the legend.

`get_linestyle()`

`get_linestyles()`

`get_linewidth()`

`get_linewidths()`

`get_offset_position()`

Returns how offsets are applied for the collection. If `offset_position` is 'screen', the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`get_offset_transform()`

`get_offsets()`

Return the offsets for the collection.

`get_path_effects()`

`get_paths()`

`get_picker()`

Return the picker object used by this artist.

`get_pickradius()`

`get_rasterized()`

Return whether the artist is to be rasterized.

`get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

`sketch_params` [tuple or `None`]

A 3-tuple with the following elements:

- `scale`: The amplitude of the wiggle perpendicular to the source line.
- `length`: The length of the wiggle along the line.
- `randomness`: The scale factor by which the length is shrunk or expanded.

May return 'None' if no sketch parameters were set.

`get_snap()`

Returns the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`get_transform()`

Return the Transform instance used by this artist.

`get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`get_transforms()`

`get_url()`

Returns the url.

`get_urls()`

`get_visible()`

Return the artist's visibility

`get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`get_zorder()`

Return the artist's zorder.

`have_units()`

Return True if units are set on the x or y axes

`hitlist(event)`

Deprecated since version 2.2: The hitlist function was deprecated in version 2.2.

List the children of the artist which contain the mouse event event.

`is_figure_set()`

Deprecated since version 2.2: `artist.figure` is not None

Returns whether the artist is assigned to a Figure.

`is_transform_set()`

Returns True if Artist has a transform explicitly set.

`mouseover`

`pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`pick(mouseevent)`

Process pick event

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

`pickable()`

Return True if Artist is pickable.

`properties()`

return a dictionary mapping property name -> value for all Artist props

`remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`set(**kwargs)`

A property batch setter. Pass kwargs to set properties.

`set_agg_filter(filter_func)`

Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`set_alpha(alpha)`

Set the alpha transparencies of the collection. alpha must be a float or None.

ACCEPTS: float or None

`set_animated(b)`

Set the artist's animation state.

Parameters

`b` [bool]

`set_antialiased(aa)`

Set the antialiasing state for rendering.

ACCEPTS: Boolean or sequence of booleans

`set_antialiaseds(aa)`

alias for `set_antialiased`

`set_array(A)`

Set the image array from numpy array `A`.

Parameters

`A` [ndarray]

`set_capstyle(cs)`

Set the capstyle for the collection. The capstyle can only be set globally for all elements in the collection

Parameters

`cs` [['butt' | 'round' | 'projecting']] The capstyle

`set_clim(vmin=None, vmax=None)`

set the norm limits for image scaling; if `vmin` is a length2 sequence, interpret it as (`vmin`, `vmax`) which is used to support `setp`

ACCEPTS: a length 2 sequence of floats; may be overridden in methods that have `vmin` and `vmax` kwargs.

`set_clip_box(clipbox)`

Set the artist's clip Bbox.

Parameters

`clipbox` [Bbox]

`set_clip_on(b)`

Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

`b` [bool]

`set_clip_path(path, transform=None)`

Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or

- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

`set_cmap(cmap)`

set the colormap for luminance data

ACCEPTS: a colormap or registered colormap name

`set_color(c)`

Set both the edgecolor and the facecolor.

ACCEPTS: matplotlib color arg or sequence of rgba tuples

See also:

`set_facecolor()`, `set_edgecolor()` For setting the edge or face color individually.

`set_contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

`picker` [callable]

`set_dashes(ls)`

alias for `set_linestyle`

`set_edgecolor(c)`

Set the edgecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'face', the edge color will always be the same as the face color. If it is 'none', the patch boundary will not be drawn.

ACCEPTS: matplotlib color spec or sequence of specs

`set_edgecolors(c)`

alias for `set_edgecolor`

`set_facecolor(c)`

Set the facecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have

same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'none', the patch will not be filled.

ACCEPTS: matplotlib color spec or sequence of specs

`set_facecolors(c)`
alias for `set_facecolor`

`set_figure(fig)`
Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`set_gid(gid)`
Sets the (group) id for the artist.

Parameters

`gid` [str]

`set_hatch(hatch)`
Set the hatching pattern

`hatch` can be one of:

/	- diagonal hatching
\	- back diagonal
	- vertical
-	- horizontal
+	- crossed
x	- crossed diagonal
o	- small circle
O	- large circle
.	- dots
*	- stars

Letters can be combined, in which case all the specified hatchings are done. If same letter repeats, it increases the density of hatching of that pattern.

Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Unlike other properties such as linewidth and colors, hatching can only be specified for the collection as a whole, not separately for each member.

ACCEPTS: ['/' | '\ ' | '|' | '-' | '+' | 'x' | 'o' | 'O' | '.' | '*']

`set_joinstyle(js)`
Set the joinstyle for the collection. The joinstyle can only be set globally for all elements in the collection.

Parameters

js `[['miter' | 'round' | 'bevel']]` The joinstyle

set_label(s)

Set the label to s for auto legend.

Parameters

s `[object]` s will be converted to a string by calling `str` (unicode on Py2).

set_linestyle(ls)

Set the linestyle(s) for the collection.

linestyle	description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line

Alternatively a dash tuple of the following form can be provided:

(offset, onoffseq),

where onoffseq is an even length tuple of on and off ink in points.

ACCEPTS: `['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '--' | '-.' | ':' | 'None' | '' | '']`

Parameters

ls `[{ '-', '--', '-.', ':' } and more see description]` The line style.

set_linestyles(ls)

alias for set_linestyle

set_linewidth(lw)

Set the linewidth(s) for the collection. lw can be a scalar or a sequence; if it is a sequence the patches will cycle through the sequence

ACCEPTS: float or sequence of floats

set_linewidths(lw)

alias for set_linewidth

set_lw(lw)

alias for set_linewidth

set_norm(norm)

Set the normalization instance.

Parameters

norm `[Normalize]`

`set_offset_position(offset_position)`

Set how offsets are applied. If `offset_position` is 'screen' (default) the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`set_offsets(offsets)`

Set the offsets for the collection. `offsets` can be a scalar or a sequence.

ACCEPTS: float or sequence of floats

`set_path_effects(path_effects)`

Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`set_paths()`

`set_picker(picker)`

Set the epsilon for picking used by this artist

`picker` can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if `picker` is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if `picker` is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

`hit, props = picker(artist, mouseevent)`

to determine the hit test. if the mouse event is over the artist, return `hit=True` and `props` is a dictionary of properties you want added to the `PickEvent` attributes.

Parameters

`picker` [None or bool or float or callable]

`set_pickradius(pr)`

Set the pick radius used for containment tests.

Parameters

`d` [float] Pick radius, in points.

`set_rasterized(rasterized)`

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

`rasterized` [bool or None]

`set_sketch_params(scale=None, length=None, randomness=None)`

Sets the sketch parameters.

Parameters

`scale` [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If scale is `None`, or not provided, no sketch filter will be provided.

`length` [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

`randomness` [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

`set_snap(snap)`

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

`snap` [bool or None]

`set_transform(t)`

Set the artist transform.

Parameters

`t` [Transform]

`set_url(url)`

Sets the url for the artist.

Parameters

`url` [str]

`set_urls(urls)`

Parameters

`urls` [List[str] or None]

`set_visible(b)`

Set the artist's visibility.

Parameters

`b` [bool]

`set_zorder(level)`

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

`level` [float]

`stale`

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

`sticky_edges`

x and y sticky edge lists.

When performing autoscaling, if a data limit coincides with a value in the corresponding `sticky_edges` list, then no margin will be added—the view limit "sticks" to the edge. A typical usecase is histograms, where one usually expects no margin on the bottom edge (0) of the histogram.

This attribute cannot be assigned to; however, the x and y lists can be modified in place as needed.

Examples

```
>>> artist.sticky_edges.x[:] = (xmin, xmax)
>>> artist.sticky_edges.y[:] = (ymin, ymax)
```

`to_rgba(x, alpha=None, bytes=False, norm=True)`

Return a normalized rgba array corresponding to x.

In the normal case, x is a 1-D or 2-D sequence of scalars, and the corresponding ndarray of rgba values will be returned, based on the norm and colormap set for this `ScalarMap`-able.

There is one special case, for handling images that are already rgb or rgba, such as might have been read from an image file. If x is an ndarray with 3 dimensions, and the last dimension is either 3 or 4, then it will be treated as an rgb or rgba array, and no mapping will be done. The array can be uint8, or it can be floating point with values in the 0-1 range; otherwise a `ValueError` will be raised. If it is a masked array, the mask will be ignored. If the last dimension is 3, the alpha kwarg (defaulting to 1) will be used to fill in the transparency. If the last dimension is 4, the alpha kwarg is ignored; it does not replace the pre-existing alpha. A `ValueError` will be raised if the third dimension is other than 3 or 4.

In either case, if `bytes` is `False` (default), the rgba array will be floats in the 0-1 range; if it is `True`, the returned rgba array will be uint8 in the 0 to 255 range.

If norm is False, no normalization of the input data is performed, and it is assumed to be in the range (0-1).

update(props)

Update this artist's properties from the dictionary prop.

update_from(other)

copy properties from other to self

update_scalarmappable()

If the scalar mappable array is not none, update colors from scalar data

zorder = 0

```
class matplotlib.collections.RegularPolyCollection(numsides, rotation=0, sizes=(1, ),
                                                    **kwargs)
```

Bases: matplotlib.collections._CollectionWithSizes

Draw a collection of regular polygons with numsides.

numsides the number of sides of the polygon

rotation the rotation of the polygon in radians

sizes gives the area of the circle circumscribing the regular polygon in points²

Valid Collection keyword arguments:

- edgecolors: None
- facecolors: None
- linewidths: None
- antialiaseds: None
- offsets: None
- transOffset: transforms.IdentityTransform()
- norm: None (optional for matplotlib.cm.ScalarMappable)
- cmap: None (optional for matplotlib.cm.ScalarMappable)

offsets and transOffset are used to translate the patch after rendering (default no offsets)

If any of edgecolors, facecolors, linewidths, antialiaseds are None, they default to their matplotlib.rcParams patch setting, in sequence form.

Example: see examples/dynamic_collection.py for complete example:

```
offsets = np.random.rand(20,2)
facecolors = [cm.jet(x) for x in np.random.rand(20)]
black = (0,0,0,1)

collection = RegularPolyCollection(
    numsides=5, # a pentagon
```

(continues on next page)

(continued from previous page)

```
rotation=0, sizes=(50,),
facecolors = facecolors,
edgecolors = (black,),
linewidths = (1,),
offsets = offsets,
transOffset = ax.transData,
)
```

`add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`add_checker(checker)`

Add an entry to a dictionary of boolean flags that are set to True when the mappable is changed.

`aname = 'Artist'`

`autoscale()`

Autoscale the scalar limits on the norm instance using the current array

`autoscale_None()`

Autoscale the scalar limits on the norm instance using the current array, changing only limits that are None

`axes`

The Axes instance the artist resides in, or None.

`changed()`

Call this whenever the mappable is changed to notify all the callbackSM listeners to the 'changed' signal

`check_update(checker)`

If mappable has changed since the last check, return True; else return False

`contains(mouseevent)`

Test whether the mouse event occurred in the collection.

Returns True | False, dict(ind=itemlist), where every item in itemlist contains the event.

`convert_xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`convert_yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`draw(renderer)`

Derived classes drawing method

`findobj(match=None, include_self=True)`
 Find artist objects.
 Recursively find all Artist instances contained in self.
 match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is `True` (default), include self in the list to be checked for a match.

`format_cursor_data(data)`
 Return cursor data string formatted.

`get_agg_filter()`
 Return filter function to be used for agg filter.

`get_alpha()`
 Return the alpha value used for blending - not supported on all backends

`get_animated()`
 Return the artist's animated state

`get_array()`
 Return the array

`get_capstyle()`

`get_children()`
 Return a list of the child Artist's this :class:`Artist` contains.

`get_clim()`
 return the min, max of the color limits for image scaling

`get_clip_box()`
 Return artist clipbox

`get_clip_on()`
 Return whether artist uses clipping

`get_clip_path()`
 Return artist clip path

`get_cmap()`
 return the colormap

`get_contains()`
 Return the `__contains` test used by the artist, or `None` for default.

`get_cursor_data(event)`
 Get the cursor data for a given event.

`get_dashes()`

`get_datalim(transData)`

`get_edgecolor()`

`get_edgecolors()`

`get_facecolor()`

`get_facecolors()`

`get_figure()`

Return the Figure instance the artist belongs to.

`get_fill()`

return whether fill is set

`get_gid()`

Returns the group id.

`get_hatch()`

Return the current hatching pattern.

`get_joinstyle()`

`get_label()`

Get the label used for this artist in the legend.

`get_linestyle()`

`get_linestyles()`

`get_linewidth()`

`get_linewidths()`

`get_numsides()`

`get_offset_position()`

Returns how offsets are applied for the collection. If `offset_position` is 'screen', the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`get_offset_transform()`

`get_offsets()`

Return the offsets for the collection.

`get_path_effects()`

`get_paths()`

`get_picker()`

Return the picker object used by this artist.

`get_pickradius()`

`get_rasterized()`

Return whether the artist is to be rasterized.

`get_rotation()`

`get_sizes()`

Returns the sizes of the elements in the collection. The value represents the 'area' of the element.

Returns

sizes [array] The 'area' of each element.

`get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

sketch_params [tuple or `None`]

A 3-tuple with the following elements:

- scale: The amplitude of the wiggle perpendicular to the source line.
- length: The length of the wiggle along the line.
- randomness: The scale factor by which the length is shrunk or expanded.

May return 'None' if no sketch parameters were set.

`get_snap()`

Returns the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`get_transform()`

Return the Transform instance used by this artist.

`get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`get_transforms()`

`get_url()`

Returns the url.

`get_urls()`

`get_visible()`

Return the artist's visibility

`get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`get_zorder()`

Return the artist's zorder.

`have_units()`

Return True if units are set on the x or y axes

`hitlist(event)`

Deprecated since version 2.2: The hitlist function was deprecated in version 2.2.

List the children of the artist which contain the mouse event event.

`is_figure_set()`

Deprecated since version 2.2: `artist.figure` is not None

Returns whether the artist is assigned to a Figure.

`is_transform_set()`

Returns True if Artist has a transform explicitly set.

`mouseover`

`pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`pick(mouseevent)`
Process pick event

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

`pickable()`
Return True if Artist is pickable.

`properties()`
return a dictionary mapping property name -> value for all Artist props

`remove()`
Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`remove_callback(oid)`
Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`set(**kwargs)`
A property batch setter. Pass kwargs to set properties.

`set_agg_filter(filter_func)`
Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`set_alpha(alpha)`
Set the alpha transparencies of the collection. alpha must be a float or None.

ACCEPTS: float or None

`set_animated(b)`
Set the artist's animation state.

Parameters

`b` [bool]

`set_antialiased(aa)`
Set the antialiasing state for rendering.

ACCEPTS: Boolean or sequence of booleans

`set_antialiaseds(aa)`
alias for `set_antialiased`

`set_array(A)`
Set the image array from numpy array A.

Parameters

A [ndarray]

`set_capstyle(cs)`
Set the capstyle for the collection. The capstyle can only be set globally for all elements in the collection

Parameters

cs [['butt' | 'round' | 'projecting']] The capstyle

`set_clim(vmin=None, vmax=None)`
set the norm limits for image scaling; if `vmin` is a length2 sequence, interpret it as (`vmin`, `vmax`) which is used to support `setp`

ACCEPTS: a length 2 sequence of floats; may be overridden in methods that have `vmin` and `vmax` kwargs.

`set_clip_box(clipbox)`
Set the artist's clip Bbox.

Parameters

clipbox [Bbox]

`set_clip_on(b)`
Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

b [bool]

`set_clip_path(path, transform=None)`
Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

`set_cmap(cmap)`

set the colormap for luminance data

ACCEPTS: a colormap or registered colormap name

`set_color(c)`

Set both the edgecolor and the facecolor.

ACCEPTS: matplotlib color arg or sequence of rgba tuples

See also:

`set_facecolor()`, `set_edgecolor()` For setting the edge or face color individually.

`set_contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

`picker` [callable]

`set_dashes(ls)`

alias for `set_linestyle`

`set_edgecolor(c)`

Set the edgecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'face', the edge color will always be the same as the face color. If it is 'none', the patch boundary will not be drawn.

ACCEPTS: matplotlib color spec or sequence of specs

`set_edgecolors(c)`

alias for `set_edgecolor`

`set_facecolor(c)`

Set the facecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'none', the patch will not be filled.

ACCEPTS: matplotlib color spec or sequence of specs

`set_facecolors(c)`

alias for `set_facecolor`

`set_figure(fig)`

Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`set_gid(gid)`

Sets the (group) id for the artist.

Parameters

`gid` [str]

`set_hatch(hatch)`

Set the hatching pattern

`hatch` can be one of:

/ - diagonal hatching
\ - back diagonal
| - vertical
- - horizontal
+ - crossed
x - crossed diagonal
o - small circle
O - large circle
. - dots
* - stars

Letters can be combined, in which case all the specified hatchings are done. If same letter repeats, it increases the density of hatching of that pattern.

Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Unlike other properties such as linewidth and colors, hatching can only be specified for the collection as a whole, not separately for each member.

ACCEPTS: ['/' | '\ ' | '|' | '- ' | '+ ' | 'x' | 'o' | 'O' | '.' | '*']

`set_joinstyle(js)`

Set the joinstyle for the collection. The joinstyle can only be set globally for all elements in the collection.

Parameters

`js` [['miter' | 'round' | 'bevel']] The joinstyle

`set_label(s)`

Set the label to `s` for auto legend.

Parameters

`s` [object] `s` will be converted to a string by calling `str` (unicode on Py2).

`set_linestyle(ls)`

Set the linestyle(s) for the collection.

linestyle	description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line

Alternatively a dash tuple of the following form can be provided:

(offset, onoffseq),

where onoffseq is an even length tuple of on and off ink in points.

ACCEPTS: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '--' |
 '-.' | ':' | 'None' | ' ' | '']

Parameters

ls [{ '-', '--', '-.', ':'}] and more see description] The line style.

`set_linestyles(ls)`

alias for `set_linestyle`

`set_linewidth(lw)`

Set the linewidth(s) for the collection. lw can be a scalar or a sequence; if it is a sequence the patches will cycle through the sequence

ACCEPTS: float or sequence of floats

`set_linewidths(lw)`

alias for `set_linewidth`

`set_lw(lw)`

alias for `set_linewidth`

`set_norm(norm)`

Set the normalization instance.

Parameters

norm [Normalize]

`set_offset_position(offset_position)`

Set how offsets are applied. If `offset_position` is 'screen' (default) the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`set_offsets(offsets)`

Set the offsets for the collection. offsets can be a scalar or a sequence.

ACCEPTS: float or sequence of floats

`set_path_effects(path_effects)`

Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`set_paths()`

`set_picker(picker)`

Set the epsilon for picking used by this artist

picker can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if picker is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if picker is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

`hit, props = picker(artist, mouseevent)`

to determine the hit test. if the mouse event is over the artist, return `hit=True` and `props` is a dictionary of properties you want added to the `PickEvent` attributes.

Parameters

`picker` [None or bool or float or callable]

`set_pickradius(pr)`

Set the pick radius used for containment tests.

Parameters

`d` [float] Pick radius, in points.

`set_rasterized(rasterized)`

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

`rasterized` [bool or None]

`set_sizes(sizes, dpi=72.0)`

Set the sizes of each member of the collection.

Parameters

`sizes` [ndarray or None] The size to set for each element of the collection.
The value is the 'area' of the element.

`dpi` [float] The dpi of the canvas. Defaults to 72.0.

`set_sketch_params(scale=None, length=None, randomness=None)`

Sets the sketch parameters.

Parameters

`scale` [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If scale is `None`, or not provided, no sketch filter will be provided.

`length` [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

`randomness` [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

`set_snap(snap)`

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

`snap` [bool or None]

`set_transform(t)`

Set the artist transform.

Parameters

`t` [Transform]

`set_url(url)`

Sets the url for the artist.

Parameters

`url` [str]

`set_urls(urls)`

Parameters

urls [List[str] or None]

set_visible(b)

Set the artist's visibility.

Parameters

b [bool]

set_zorder(level)

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

level [float]

stale

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

sticky_edges

x and y sticky edge lists.

When performing autoscaling, if a data limit coincides with a value in the corresponding sticky_edges list, then no margin will be added—the view limit "sticks" to the edge. A typical usecase is histograms, where one usually expects no margin on the bottom edge (0) of the histogram.

This attribute cannot be assigned to; however, the x and y lists can be modified in place as needed.

Examples

```
>>> artist.sticky_edges.x[:] = (xmin, xmax)
>>> artist.sticky_edges.y[:] = (ymin, ymax)
```

to_rgba(x, alpha=None, bytes=False, norm=True)

Return a normalized rgba array corresponding to x.

In the normal case, x is a 1-D or 2-D sequence of scalars, and the corresponding ndarray of rgba values will be returned, based on the norm and colormap set for this ScalarMappable.

There is one special case, for handling images that are already rgb or rgba, such as might have been read from an image file. If x is an ndarray with 3 dimensions, and the last dimension is either 3 or 4, then it will be treated as an rgb or rgba array, and no mapping will be done. The array can be uint8, or it can be floating point with values in the 0-1 range; otherwise a ValueError will be raised. If it is a masked array, the mask will be ignored. If the last dimension is 3, the alpha kwarg (defaulting to 1) will be used to fill in the transparency. If the last dimension is 4, the alpha kwarg is ignored; it does not replace the pre-existing alpha. A ValueError will be raised if the third dimension is other than 3 or 4.

In either case, if `bytes` is `False` (default), the `rgba` array will be floats in the 0-1 range; if it is `True`, the returned `rgba` array will be `uint8` in the 0 to 255 range.

If `norm` is `False`, no normalization of the input data is performed, and it is assumed to be in the range (0-1).

`update(props)`

Update this artist's properties from the dictionary `prop`.

`update_from(other)`

copy properties from `other` to self

`update_scalarmappable()`

If the scalar mappable array is not `None`, update colors from scalar data

`zorder = 0`

```
class matplotlib.collections.StarPolygonCollection(numsides, rotation=0, sizes=(1, ),
                                                    **kwargs)
```

Bases: `matplotlib.collections.RegularPolyCollection`

Draw a collection of regular stars with `numsides` points.

`numsides` the number of sides of the polygon

`rotation` the rotation of the polygon in radians

`sizes` gives the area of the circle circumscribing the regular polygon in `points2`

Valid Collection keyword arguments:

- `edgecolors`: `None`
- `facecolors`: `None`
- `linewidths`: `None`
- `antialiaseds`: `None`
- `offsets`: `None`
- `transOffset`: `transforms.IdentityTransform()`
- `norm`: `None` (optional for `matplotlib.cm.ScalarMappable`)
- `cmap`: `None` (optional for `matplotlib.cm.ScalarMappable`)

`offsets` and `transOffset` are used to translate the patch after rendering (default no offsets)

If any of `edgecolors`, `facecolors`, `linewidths`, `antialiaseds` are `None`, they default to their `matplotlib.rcParams` patch setting, in sequence form.

Example: see `examples/dynamic_collection.py` for complete example:

```
offsets = np.random.rand(20,2)
facecolors = [cm.jet(x) for x in np.random.rand(20)]
black = (0,0,0,1)
```

(continues on next page)

(continued from previous page)

```
collection = RegularPolyCollection(  
    numsides=5, # a pentagon  
    rotation=0, sizes=(50,),  
    facecolors = facecolors,  
    edgecolors = (black,),  
    linewidths = (1,),  
    offsets = offsets,  
    transOffset = ax.transData,  
)
```

`add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`add_checker(checker)`

Add an entry to a dictionary of boolean flags that are set to True when the mappable is changed.

`aname = 'Artist'`

`autoscale()`

Autoscale the scalar limits on the norm instance using the current array

`autoscale_None()`

Autoscale the scalar limits on the norm instance using the current array, changing only limits that are None

`axes`

The Axes instance the artist resides in, or None.

`changed()`

Call this whenever the mappable is changed to notify all the callbackSM listeners to the 'changed' signal

`check_update(checker)`

If mappable has changed since the last check, return True; else return False

`contains(mouseevent)`

Test whether the mouse event occurred in the collection.

Returns True | False, dict(ind=itemlist), where every item in itemlist contains the event.

`convert_xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`convert_yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`draw(renderer)`
Derived classes drawing method

`findobj(match=None, include_self=True)`
Find artist objects.
Recursively find all Artist instances contained in self.
match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is `True` (default), include self in the list to be checked for a match.

`format_cursor_data(data)`
Return cursor data string formatted.

`get_agg_filter()`
Return filter function to be used for agg filter.

`get_alpha()`
Return the alpha value used for blending - not supported on all backends

`get_animated()`
Return the artist's animated state

`get_array()`
Return the array

`get_capstyle()`

`get_children()`
Return a list of the child Artist's this :class:`Artist` contains.

`get_clim()`
return the min, max of the color limits for image scaling

`get_clip_box()`
Return artist clipbox

`get_clip_on()`
Return whether artist uses clipping

`get_clip_path()`
Return artist clip path

`get_cmap()`
return the colormap

`get_contains()`
Return the `_contains` test used by the artist, or `None` for default.

`get_cursor_data(event)`

Get the cursor data for a given event.

`get_dashes()`

`get_datalim(transData)`

`get_edgecolor()`

`get_edgecolors()`

`get_facecolor()`

`get_facecolors()`

`get_figure()`

Return the Figure instance the artist belongs to.

`get_fill()`

return whether fill is set

`get_gid()`

Returns the group id.

`get_hatch()`

Return the current hatching pattern.

`get_joinstyle()`

`get_label()`

Get the label used for this artist in the legend.

`get_linestyle()`

`get_linestyles()`

`get_linewidth()`

`get_linewidths()`

`get_numsides()`

`get_offset_position()`

Returns how offsets are applied for the collection. If `offset_position` is 'screen', the offset is applied after the master transform has been applied, that is, the offsets are in

screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`get_offset_transform()`

`get_offsets()`

Return the offsets for the collection.

`get_path_effects()`

`get_paths()`

`get_picker()`

Return the picker object used by this artist.

`get_pickradius()`

`get_rasterized()`

Return whether the artist is to be rasterized.

`get_rotation()`

`get_sizes()`

Returns the sizes of the elements in the collection. The value represents the 'area' of the element.

Returns

sizes [array] The 'area' of each element.

`get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

sketch_params [tuple or `None`]

A 3-tuple with the following elements:

- scale: The amplitude of the wiggle perpendicular to the source line.
- length: The length of the wiggle along the line.
- randomness: The scale factor by which the length is shrunk or expanded.

May return 'None' if no sketch parameters were set.

`get_snap()`

Returns the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is

- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`get_transform()`

Return the Transform instance used by this artist.

`get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`get_transforms()`

`get_url()`

Returns the url.

`get_urls()`

`get_visible()`

Return the artist's visibility

`get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`get_zorder()`

Return the artist's zorder.

`have_units()`

Return True if units are set on the x or y axes

`hitlist(event)`

Deprecated since version 2.2: The hitlist function was deprecated in version 2.2.

List the children of the artist which contain the mouse event event.

`is_figure_set()`

Deprecated since version 2.2: `artist.figure` is not None

Returns whether the artist is assigned to a Figure.

`is_transform_set()`

Returns True if Artist has a transform explicitly set.

`mouseover`

`pchanged()`
Fire an event when property changed, calling all of the registered callbacks.

`pick(mouseevent)`
Process pick event

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

`pickable()`
Return True if Artist is pickable.

`properties()`
return a dictionary mapping property name -> value for all Artist props

`remove()`
Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`remove_callback(oid)`
Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`set(**kwargs)`
A property batch setter. Pass kwargs to set properties.

`set_agg_filter(filter_func)`
Set the agg filter.

Parameters

 `filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`set_alpha(alpha)`
Set the alpha transparencies of the collection. alpha must be a float or None.

ACCEPTS: float or None

`set_animated(b)`
Set the artist's animation state.

Parameters

 `b` [bool]

`set_antialiased(aa)`

Set the antialiasing state for rendering.

ACCEPTS: Boolean or sequence of booleans

`set_antialiaseds(aa)`

alias for `set_antialiased`

`set_array(A)`

Set the image array from numpy array `A`.

Parameters

`A` [ndarray]

`set_capstyle(cs)`

Set the capstyle for the collection. The capstyle can only be set globally for all elements in the collection

Parameters

`cs` [['butt' | 'round' | 'projecting']] The capstyle

`set_clim(vmin=None, vmax=None)`

set the norm limits for image scaling; if `vmin` is a length2 sequence, interpret it as (`vmin`, `vmax`) which is used to support `setp`

ACCEPTS: a length 2 sequence of floats; may be overridden in methods that have `vmin` and `vmax` kwargs.

`set_clip_box(clipbox)`

Set the artist's clip Bbox.

Parameters

`clipbox` [Bbox]

`set_clip_on(b)`

Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

`b` [bool]

`set_clip_path(path, transform=None)`

Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

`set_cmap(cmap)`

set the colormap for luminance data

ACCEPTS: a colormap or registered colormap name

`set_color(c)`

Set both the edgecolor and the facecolor.

ACCEPTS: matplotlib color arg or sequence of rgba tuples

See also:

`set_facecolor()`, `set_edgecolor()` For setting the edge or face color individually.

`set_contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

`picker` [callable]

`set_dashes(ls)`

alias for `set_linestyle`

`set_edgecolor(c)`

Set the edgecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'face', the edge color will always be the same as the face color. If it is 'none', the patch boundary will not be drawn.

ACCEPTS: matplotlib color spec or sequence of specs

`set_edgecolors(c)`

alias for `set_edgecolor`

`set_facecolor(c)`

Set the facecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'none', the patch will not be filled.

ACCEPTS: matplotlib color spec or sequence of specs

`set_facecolors(c)`

alias for `set_facecolor`

`set_figure(fig)`
Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`set_gid(gid)`
Sets the (group) id for the artist.

Parameters

`gid` [str]

`set_hatch(hatch)`
Set the hatching pattern

`hatch` can be one of:

/	- diagonal hatching
\	- back diagonal
	- vertical
-	- horizontal
+	- crossed
x	- crossed diagonal
o	- small circle
O	- large circle
.	- dots
*	- stars

Letters can be combined, in which case all the specified hatchings are done. If same letter repeats, it increases the density of hatching of that pattern.

Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Unlike other properties such as linewidth and colors, hatching can only be specified for the collection as a whole, not separately for each member.

ACCEPTS: ['/' | '\' | '|' | '-' | '+' | 'x' | 'o' | 'O' | '.' | '*']

`set_joinstyle(js)`
Set the joinstyle for the collection. The joinstyle can only be set globally for all elements in the collection.

Parameters

`js` [['miter' | 'round' | 'bevel']] The joinstyle

`set_label(s)`
Set the label to `s` for auto legend.

Parameters

`s` [object] `s` will be converted to a string by calling `str` (unicode on Py2).

`set_linestyle(ls)`

Set the linestyle(s) for the collection.

linestyle	description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line

Alternatively a dash tuple of the following form can be provided:

`(offset, onoffseq),`

where onoffseq is an even length tuple of on and off ink in points.

ACCEPTS: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '--' |
'-.' | ':' | 'None' | ' ' | '']

Parameters

ls [{ '-', '--', '-.', ':'}] and more see description] The line style.

`set_linestyles(ls)`

alias for `set_linestyle`

`set_linewidth(lw)`

Set the linewidth(s) for the collection. lw can be a scalar or a sequence; if it is a sequence the patches will cycle through the sequence

ACCEPTS: float or sequence of floats

`set_linewidths(lw)`

alias for `set_linewidth`

`set_lw(lw)`

alias for `set_linewidth`

`set_norm(norm)`

Set the normalization instance.

Parameters

norm [Normalize]

`set_offset_position(offset_position)`

Set how offsets are applied. If `offset_position` is 'screen' (default) the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`set_offsets(offsets)`

Set the offsets for the collection. offsets can be a scalar or a sequence.

ACCEPTS: float or sequence of floats

`set_path_effects(path_effects)`

Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`set_paths()`

`set_picker(picker)`

Set the epsilon for picking used by this artist

picker can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if picker is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if picker is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

`hit, props = picker(artist, mouseevent)`

to determine the hit test. if the mouse event is over the artist, return `hit=True` and `props` is a dictionary of properties you want added to the `PickEvent` attributes.

Parameters

`picker` [None or bool or float or callable]

`set_pickradius(pr)`

Set the pick radius used for containment tests.

Parameters

`d` [float] Pick radius, in points.

`set_rasterized(rasterized)`

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

`rasterized` [bool or None]

`set_sizes(sizes, dpi=72.0)`

Set the sizes of each member of the collection.

Parameters

`sizes` [ndarray or None] The size to set for each element of the collection.
The value is the 'area' of the element.

`dpi` [float] The dpi of the canvas. Defaults to 72.0.

`set_sketch_params(scale=None, length=None, randomness=None)`

Sets the sketch parameters.

Parameters

`scale` [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If scale is `None`, or not provided, no sketch filter will be provided.

`length` [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

`randomness` [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

`set_snap(snap)`

Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

`snap` [bool or None]

`set_transform(t)`

Set the artist transform.

Parameters

`t` [Transform]

`set_url(url)`

Sets the url for the artist.

Parameters

`url` [str]

`set_urls(urls)`

Parameters

urls [List[str] or None]

set_visible(b)

Set the artist's visibility.

Parameters

b [bool]

set_zorder(level)

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

level [float]

stale

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

sticky_edges

x and y sticky edge lists.

When performing autoscaling, if a data limit coincides with a value in the corresponding sticky_edges list, then no margin will be added—the view limit "sticks" to the edge. A typical usecase is histograms, where one usually expects no margin on the bottom edge (0) of the histogram.

This attribute cannot be assigned to; however, the x and y lists can be modified in place as needed.

Examples

```
>>> artist.sticky_edges.x[:] = (xmin, xmax)
>>> artist.sticky_edges.y[:] = (ymin, ymax)
```

to_rgba(x, alpha=None, bytes=False, norm=True)

Return a normalized rgba array corresponding to x.

In the normal case, x is a 1-D or 2-D sequence of scalars, and the corresponding ndarray of rgba values will be returned, based on the norm and colormap set for this ScalarMappable.

There is one special case, for handling images that are already rgb or rgba, such as might have been read from an image file. If x is an ndarray with 3 dimensions, and the last dimension is either 3 or 4, then it will be treated as an rgb or rgba array, and no mapping will be done. The array can be uint8, or it can be floating point with values in the 0-1 range; otherwise a ValueError will be raised. If it is a masked array, the mask will be ignored. If the last dimension is 3, the alpha kwarg (defaulting to 1) will be used to fill in the transparency. If the last dimension is 4, the alpha kwarg is ignored; it does not replace the pre-existing alpha. A ValueError will be raised if the third dimension is other than 3 or 4.

In either case, if `bytes` is `False` (default), the `rgba` array will be floats in the 0-1 range; if it is `True`, the returned `rgba` array will be `uint8` in the 0 to 255 range.

If `norm` is `False`, no normalization of the input data is performed, and it is assumed to be in the range (0-1).

`update(props)`

Update this artist's properties from the dictionary `prop`.

`update_from(other)`

copy properties from `other` to self

`update_scalarmappable()`

If the scalar mappable array is not `None`, update colors from scalar data

`zorder = 0`

`class matplotlib.collections.TriMesh(triangulation, **kwargs)`

Bases: `matplotlib.collections.Collection`

Class for the efficient drawing of a triangular mesh using Gouraud shading.

A triangular mesh is a `Triangulation` object.

`add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`add_checker(checker)`

Add an entry to a dictionary of boolean flags that are set to `True` when the mappable is changed.

`aname = 'Artist'`

`autoscale()`

Autoscale the scalar limits on the norm instance using the current array

`autoscale_None()`

Autoscale the scalar limits on the norm instance using the current array, changing only limits that are `None`

`axes`

The `Axes` instance the artist resides in, or `None`.

`changed()`

Call this whenever the mappable is changed to notify all the `callbackSM` listeners to the 'changed' signal

`check_update(checker)`

If mappable has changed since the last check, return `True`; else return `False`

`contains(mouseevent)`

Test whether the mouse event occurred in the collection.

Returns True | False, dict(ind=itemlist), where every item in itemlist contains the event.

static `convert_mesh_to_paths(tri)`

Converts a given mesh into a sequence of `matplotlib.path.Path` objects for easier rendering by backends that do not directly support meshes.

This function is primarily of use to backend implementers.

`convert_xunits(x)`

For artists in an axes, if the xaxis has units support, convert x using xaxis unit type

`convert_yunits(y)`

For artists in an axes, if the yaxis has units support, convert y using yaxis unit type

`draw(renderer)`

Derived classes drawing method

`findobj(match=None, include_self=True)`

Find artist objects.

Recursively find all Artist instances contained in self.

match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is True (default), include self in the list to be checked for a match.

`format_cursor_data(data)`

Return cursor data string formatted.

`get_agg_filter()`

Return filter function to be used for agg filter.

`get_alpha()`

Return the alpha value used for blending - not supported on all backends

`get_animated()`

Return the artist's animated state

`get_array()`

Return the array

`get_capstyle()`

`get_children()`

Return a list of the child Artist's this :class:`Artist` contains.

`get_clim()`

return the min, max of the color limits for image scaling

`get_clip_box()`

Return artist clipbox

`get_clip_on()`
Return whether artist uses clipping

`get_clip_path()`
Return artist clip path

`get_cmap()`
return the colormap

`get_contains()`
Return the `__contains` test used by the artist, or `None` for default.

`get_cursor_data(event)`
Get the cursor data for a given event.

`get_dashes()`

`get_datalim(transData)`

`get_edgecolor()`

`get_edgecolors()`

`get_facecolor()`

`get_facecolors()`

`get_figure()`
Return the Figure instance the artist belongs to.

`get_fill()`
return whether fill is set

`get_gid()`
Returns the group id.

`get_hatch()`
Return the current hatching pattern.

`get_joinstyle()`

`get_label()`
Get the label used for this artist in the legend.

`get_linestyle()`

`get_linestyles()`

`get_linewidth()`

`get_linewidths()`

`get_offset_position()`

Returns how offsets are applied for the collection. If `offset_position` is 'screen', the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`get_offset_transform()`

`get_offsets()`

Return the offsets for the collection.

`get_path_effects()`

`get_paths()`

`get_picker()`

Return the picker object used by this artist.

`get_pickradius()`

`get_rasterized()`

Return whether the artist is to be rasterized.

`get_sketch_params()`

Returns the sketch parameters for the artist.

Returns

`sketch_params` [tuple or `None`]

A 3-tuple with the following elements:

- `scale`: The amplitude of the wiggle perpendicular to the source line.
- `length`: The length of the wiggle along the line.
- `randomness`: The scale factor by which the length is shrunk or expanded.

May return 'None' if no sketch parameters were set.

`get_snap()`

Returns the snap setting which may be:

- `True`: snap vertices to the nearest pixel center
- `False`: leave vertices as-is

- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

`get_transform()`

Return the Transform instance used by this artist.

`get_transformed_clip_path_and_affine()`

Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

`get_transforms()`

`get_url()`

Returns the url.

`get_urls()`

`get_visible()`

Return the artist's visibility

`get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`get_zorder()`

Return the artist's zorder.

`have_units()`

Return True if units are set on the x or y axes

`hitlist(event)`

Deprecated since version 2.2: The hitlist function was deprecated in version 2.2.

List the children of the artist which contain the mouse event event.

`is_figure_set()`

Deprecated since version 2.2: `artist.figure` is not None

Returns whether the artist is assigned to a Figure.

`is_transform_set()`

Returns True if Artist has a transform explicitly set.

`mouseover`

`pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`pick(mouseevent)`

Process pick event

each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

`pickable()`

Return True if Artist is pickable.

`properties()`

return a dictionary mapping property name -> value for all Artist props

`remove()`

Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with `autolim = True`.

Note: there is no support for removing the artist's legend entry.

`remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`set(**kwargs)`

A property batch setter. Pass kwargs to set properties.

`set_agg_filter(filter_func)`

Set the agg filter.

Parameters

`filter_func` [callable] A filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array.

`set_alpha(alpha)`

Set the alpha transparencies of the collection. alpha must be a float or None.

ACCEPTS: float or None

`set_animated(b)`

Set the artist's animation state.

Parameters

`b` [bool]

`set_antialiased(aa)`

Set the antialiasing state for rendering.

ACCEPTS: Boolean or sequence of booleans

`set_antialiaseds(aa)`

alias for `set_antialiased`

`set_array(A)`

Set the image array from numpy array A.

Parameters

A [ndarray]

`set_capstyle(cs)`

Set the capstyle for the collection. The capstyle can only be set globally for all elements in the collection

Parameters

cs [['butt' | 'round' | 'projecting']] The capstyle

`set_clim(vmin=None, vmax=None)`

set the norm limits for image scaling; if vmin is a length2 sequence, interpret it as (vmin, vmax) which is used to support setp

ACCEPTS: a length 2 sequence of floats; may be overridden in methods that have vmin and vmax kwargs.

`set_clip_box(clipbox)`

Set the artist's clip Bbox.

Parameters

clipbox [Bbox]

`set_clip_on(b)`

Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

Parameters

b [bool]

`set_clip_path(path, transform=None)`

Set the artist's clip path, which may be:

- a Patch (or subclass) instance; or
- a Path instance, in which case a Transform instance, which will be applied to the path before using it for clipping, must be provided; or
- None, to remove a previously set clipping path.

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

`set_cmap(cmap)`

set the colormap for luminance data

ACCEPTS: a colormap or registered colormap name

`set_color(c)`

Set both the edgecolor and the facecolor.

ACCEPTS: matplotlib color arg or sequence of rgba tuples

See also:

`set_facecolor()`, `set_edgecolor()` For setting the edge or face color individually.

`set_contains(picker)`

Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return `hit = True` and `props` is a dictionary of properties you want returned with the contains test.

Parameters

`picker` [callable]

`set_dashes(ls)`

alias for `set_linestyle`

`set_edgecolor(c)`

Set the edgecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'face', the edge color will always be the same as the face color. If it is 'none', the patch boundary will not be drawn.

ACCEPTS: matplotlib color spec or sequence of specs

`set_edgecolors(c)`

alias for `set_edgecolor`

`set_facecolor(c)`

Set the facecolor(s) of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'none', the patch will not be filled.

ACCEPTS: matplotlib color spec or sequence of specs

`set_facecolors(c)`

alias for `set_facecolor`

`set_figure(fig)`
Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`set_gid(gid)`
Sets the (group) id for the artist.

Parameters

`gid` [str]

`set_hatch(hatch)`
Set the hatching pattern

`hatch` can be one of:

```
/ - diagonal hatching
\ - back diagonal
| - vertical
- - horizontal
+ - crossed
x - crossed diagonal
o - small circle
O - large circle
. - dots
* - stars
```

Letters can be combined, in which case all the specified hatchings are done. If same letter repeats, it increases the density of hatching of that pattern.

Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Unlike other properties such as linewidth and colors, hatching can only be specified for the collection as a whole, not separately for each member.

ACCEPTS: ['/' | '\ ' | '|' | '- ' | '+ ' | 'x' | 'o' | 'O' | '.' | '*']

`set_joinstyle(js)`
Set the joinstyle for the collection. The joinstyle can only be set globally for all elements in the collection.

Parameters

`js` [['miter' | 'round' | 'bevel']] The joinstyle

`set_label(s)`
Set the label to `s` for auto legend.

Parameters

`s` [object] `s` will be converted to a string by calling `str` (unicode on Py2).

`set_linestyle(ls)`

Set the linestyle(s) for the collection.

linestyle	description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line

Alternatively a dash tuple of the following form can be provided:

(offset, onoffseq),

where onoffseq is an even length tuple of on and off ink in points.

ACCEPTS: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '--' |
'-.' | ':' | 'None' | ' ' | '']

Parameters

ls [{'-', '--', '-.', ':'} and more see description] The line style.

`set_linestyles(ls)`

alias for `set_linestyle`

`set_linewidth(lw)`

Set the linewidth(s) for the collection. lw can be a scalar or a sequence; if it is a sequence the patches will cycle through the sequence

ACCEPTS: float or sequence of floats

`set_linewidths(lw)`

alias for `set_linewidth`

`set_lw(lw)`

alias for `set_linewidth`

`set_norm(norm)`

Set the normalization instance.

Parameters

norm [Normalize]

`set_offset_position(offset_position)`

Set how offsets are applied. If `offset_position` is 'screen' (default) the offset is applied after the master transform has been applied, that is, the offsets are in screen coordinates. If `offset_position` is 'data', the offset is applied before the master transform, i.e., the offsets are in data coordinates.

`set_offsets(offsets)`

Set the offsets for the collection. offsets can be a scalar or a sequence.

ACCEPTS: float or sequence of floats

`set_path_effects(path_effects)`

Set the path effects.

Parameters

`path_effects` [AbstractPathEffect]

`set_paths()`

`set_picker(picker)`

Set the epsilon for picking used by this artist

`picker` can be one of the following:

- None: picking is disabled for this artist (default)
- A boolean: if True then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if `picker` is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if `picker` is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

to determine the hit test. if the mouse event is over the artist, return `hit=True` and `props` is a dictionary of properties you want added to the `PickEvent` attributes.

Parameters

`picker` [None or bool or float or callable]

`set_pickradius(pr)`

Set the pick radius used for containment tests.

Parameters

`d` [float] Pick radius, in points.

`set_rasterized(rasterized)`

Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior.

Parameters

`rasterized` [bool or None]

`set_sketch_params(scale=None, length=None, randomness=None)`

Sets the sketch parameters.

Parameters

`scale` [float, optional] The amplitude of the wiggle perpendicular to the source line, in pixels. If `scale` is `None`, or not provided, no sketch filter will be provided.

`length` [float, optional] The length of the wiggle along the line, in pixels (default 128.0)

`randomness` [float, optional] The scale factor by which the length is shrunk or expanded (default 16.0)

`set_snap(snap)`

Sets the snap setting which may be:

- `True`: snap vertices to the nearest pixel center
- `False`: leave vertices as-is
- `None`: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

Parameters

`snap` [bool or None]

`set_transform(t)`

Set the artist transform.

Parameters

`t` [Transform]

`set_url(url)`

Sets the url for the artist.

Parameters

`url` [str]

`set_urls(urls)`

Parameters

`urls` [List[str] or None]

`set_visible(b)`

Set the artist's visibility.

Parameters

`b` [bool]

`set_zorder(level)`

Set the zorder for the artist. Artists with lower zorder values are drawn first.

Parameters

`level` [float]

`stale`

If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

`sticky_edges`

x and y sticky edge lists.

When performing autoscaling, if a data limit coincides with a value in the corresponding `sticky_edges` list, then no margin will be added—the view limit "sticks" to the edge. A typical usecase is histograms, where one usually expects no margin on the bottom edge (0) of the histogram.

This attribute cannot be assigned to; however, the x and y lists can be modified in place as needed.

Examples

```
>>> artist.sticky_edges.x[:] = (xmin, xmax)
>>> artist.sticky_edges.y[:] = (ymin, ymax)
```

`to_rgba(x, alpha=None, bytes=False, norm=True)`

Return a normalized rgba array corresponding to x.

In the normal case, x is a 1-D or 2-D sequence of scalars, and the corresponding ndarray of rgba values will be returned, based on the norm and colormap set for this `ScalarMappable`.

There is one special case, for handling images that are already rgb or rgba, such as might have been read from an image file. If x is an ndarray with 3 dimensions, and the last dimension is either 3 or 4, then it will be treated as an rgb or rgba array, and no mapping will be done. The array can be uint8, or it can be floating point with values in the 0-1 range; otherwise a `ValueError` will be raised. If it is a masked array, the mask will be ignored. If the last dimension is 3, the alpha kwarg (defaulting to 1) will be used to fill in the transparency. If the last dimension is 4, the alpha kwarg is ignored; it does not replace the pre-existing alpha. A `ValueError` will be raised if the third dimension is other than 3 or 4.

In either case, if bytes is False (default), the rgba array will be floats in the 0-1 range; if it is True, the returned rgba array will be uint8 in the 0 to 255 range.

If norm is False, no normalization of the input data is performed, and it is assumed to be in the range (0-1).

`update(props)`

Update this artist's properties from the dictionary prop.

`update_from(other)`

copy properties from other to self

`update_scalarmappable()`

If the scalar mappable array is not none, update colors from scalar data

`zorder = 0`

COLORBAR

41.1 matplotlib.colorbar

Colorbar toolkit with two classes and a function:

`ColorbarBase` the base class with full colorbar drawing functionality. It can be used as-is to make a colorbar for a given colormap; a mappable object (e.g., image) is not needed.

`Colorbar` the derived class for use with images or contour plots.

`make_axes()` a function for resizing an axes and adding a second axes suitable for a colorbar

The `colorbar()` method uses `make_axes()` and `Colorbar`; the `colorbar()` function is a thin wrapper over `colorbar()`.

```
class matplotlib.colorbar.Colorbar(ax, mappable, **kw)
```

Bases: `matplotlib.colorbar.ColorbarBase`

This class connects a `ColorbarBase` to a `ScalarMappable` such as a `AxesImage` generated via `imshow()`.

It is not intended to be instantiated directly; instead, use `colorbar()` or `colorbar()` to make your colorbar.

```
add_lines(CS, erase=True)
```

Add the lines from a non-filled `ContourSet` to the colorbar.

Set `erase` to `False` if these lines should be added to any pre-existing lines.

```
on_mappable_changed(mappable)
```

Updates this colorbar to match the mappable's properties.

Typically this is automatically registered as an event handler by `colorbar_factory()` and should not be called manually.

```
remove()
```

Remove this colorbar from the figure. If the colorbar was created with `use_gridspec=True` then restore the `gridspec` to its previous value.

`update_bruteforce(mappable)`

Destroy and rebuild the colorbar. This is intended to become obsolete, and will probably be deprecated and then removed. It is not called when the `pyplot.colorbar` function or the `Figure.colorbar` method are used to create the colorbar.

`update_normal(mappable)`

update solid, lines, etc. Unlike `update_bruteforce`, it does not clear the axes. This is meant to be called when the image or contour plot to which this colorbar belongs is changed.

```
class matplotlib.colorbar.ColorbarBase(ax, cmap=None, norm=None, alpha=None, val-
                                         ues=None, boundaries=None, orientation='vertical',
                                         ticklocation='auto', extend='neither', spac-
                                         ing='uniform', ticks=None, format=None,
                                         drawedges=False, filled=True, extendfrac=None,
                                         extendrect=False, label="")
```

Bases: `matplotlib.cm.ScalarMappable`

Draw a colorbar in an existing axes.

This is a base class for the `Colorbar` class, which is the basis for the `colorbar()` function and the `colorbar()` method, which are the usual ways of creating a colorbar.

It is also useful by itself for showing a colormap. If the `cmap` kwarg is given but `boundaries` and `values` are left as `None`, then the colormap will be displayed on a 0-1 scale. To show the under- and over-value colors, specify the `norm` as:

`colors.Normalize(clip=False)`

To show the colors versus index instead of on the 0-1 scale, use:

`norm=colors.NoNorm.`

Useful public methods are `set_label()` and `add_lines()`.

Attributes

`ax` [Axes] The Axes instance in which the colorbar is drawn.

`lines` [list] A list of `LineCollection` if lines were drawn, otherwise an empty list.

`dividers` [LineCollection] A `LineCollection` if `drawedges` is `True`, otherwise `None`.

`add_lines(levels, colors, linewidths, erase=True)`

Draw lines on the colorbar.

`colors` and `linewidths` must be scalars or sequences the same length as `levels`.

Set `erase` to `False` to add lines without first removing any previously added lines.

`ax = None`

The axes that this colorbar lives in.

`config_axis()`

`draw_all()`

Calculate any free parameters based on the current cmap and norm, and do all the drawing.

`get_ticks(minor=False)`

Return the x ticks as a list of locations

`n_rasterize = 50`

`remove()`

Remove this colorbar from the figure

`set_alpha(alpha)`

`set_label(label, **kw)`

Label the long axis of the colorbar

`set_ticklabels(ticklabels, update_ticks=True)`

set tick labels. Tick labels are updated immediately unless `update_ticks` is False. To manually update the ticks, call `update_ticks` method explicitly.

`set_ticks(ticks, update_ticks=True)`

Set tick locations.

Parameters

`ticks` [{None, sequence, Locator instance}] If None, a default Locator will be used.

`update_ticks` [{True, False}, optional] If True, tick locations are updated immediately. If False, use `update_ticks()` to manually update the ticks.

`update_ticks()`

Force the update of the ticks and ticklabels. This must be called whenever the tick locator and/or tick formatter changes.

`class matplotlib.colorbar.ColorbarPatch(ax, mappable, **kw)`

Bases: `matplotlib.colorbar.Colorbar`

A Colorbar which is created using Patch rather than the default `pcolor()`.

It uses a list of Patch instances instead of a PatchCollection because the latter does not allow the hatch pattern to vary among the members of the collection.

`matplotlib.colorbar.colorbar_factory(cax, mappable, **kwargs)`

Creates a colorbar on the given axes for the given mappable.

Typically, for automatic colorbar placement given only a mappable use `colorbar()`.

`matplotlib.colorbar.make_axes(parents, location=None, orientation=None, fraction=0.15, shrink=1.0, aspect=20, **kw)`

Resize and reposition parent axes, and return a child axes suitable for a colorbar.

Keyword arguments may include the following (with defaults):

`location` `[[None|'left'|'right'|'top'|'bottom']]` The position, relative to parents, where the colorbar axes should be created. If `None`, the value will either come from the given orientation, else it will default to `'right'`.

`orientation` `[[None|'vertical'|'horizontal']]` The orientation of the colorbar. Typically, this keyword shouldn't be used, as it can be derived from the `location` keyword.

Property	Description
<code>orientation</code>	vertical or horizontal
<code>fraction</code>	0.15; fraction of original axes to use for colorbar
<code>pad</code>	0.05 if vertical, 0.15 if horizontal; fraction of original axes between colorbar and new image axes
<code>shrink</code>	1.0; fraction by which to multiply the size of the colorbar
<code>aspect</code>	20; ratio of long to short dimensions
<code>anchor</code>	(0.0, 0.5) if vertical; (0.5, 1.0) if horizontal; the anchor point of the colorbar axes
<code>panchor</code>	(1.0, 0.5) if vertical; (0.5, 0.0) if horizontal; the anchor point of the colorbar parent axes. If <code>False</code> , the parent axes' anchor will be unchanged

Returns (`cax`, `kw`), the child axes and the reduced `kw` dictionary to be passed when creating the colorbar instance.

`matplotlib.colorbar.make_axes_gridspec(parent, **kw)`

Resize and reposition a parent axes, and return a child axes suitable for a colorbar. This function is similar to `make_axes`. Primary differences are

- `make_axes_gridspec` only handles the `orientation` keyword and cannot handle the `"location"` keyword.
- `make_axes_gridspec` should only be used with a subplot parent.
- `make_axes` creates an instance of `Axes`. `make_axes_gridspec` creates an instance of `Subplot`.
- `make_axes` updates the position of the parent. `make_axes_gridspec` replaces the `grid_spec` attribute of the parent with a new one.

While this function is meant to be compatible with `make_axes`, there could be some minor differences.

Keyword arguments may include the following (with defaults):

`orientation` `'vertical'` or `'horizontal'`

Property	Description
orientation	vertical or horizontal
fraction	0.15; fraction of original axes to use for colorbar
pad	0.05 if vertical, 0.15 if horizontal; fraction of original axes between colorbar and new image axes
shrink	1.0; fraction by which to multiply the size of the colorbar
aspect	20; ratio of long to short dimensions
anchor	(0.0, 0.5) if vertical; (0.5, 1.0) if horizontal; the anchor point of the colorbar axes
panchor	(1.0, 0.5) if vertical; (0.5, 0.0) if horizontal; the anchor point of the colorbar parent axes. If False, the parent axes' anchor will be unchanged

All but the first of these are stripped from the input kw set.

Returns (cax, kw), the child axes and the reduced kw dictionary to be passed when creating the colorbar instance.

COLORS

For a visual representation of the Matplotlib colormaps, see:

- The `color_examples` examples for examples of controlling color with Matplotlib.
- The [Colors](#) tutorial for an in-depth guide on controlling color.

42.1 `matplotlib.colors`

A module for converting numbers or color arguments to RGB or RGBA

RGB and RGBA are sequences of, respectively, 3 or 4 floats in the range 0-1.

This module includes functions and classes for color specification conversions, and for mapping numbers to colors in a 1-D array of colors called a colormap. Colormapping typically involves two steps: a data array is first mapped onto the range 0-1 using an instance of `Normalize` or of a subclass; then this number in the 0-1 range is mapped to a color using an instance of a subclass of `Colormap`. Two are provided here: `LinearSegmentedColormap`, which is used to generate all the built-in colormap instances, but is also useful for making custom colormaps, and `ListedColormap`, which is used for generating a custom colormap from a list of color specifications.

The module also provides functions for checking whether an object can be interpreted as a color (`is_color_like()`), for converting such an object to an RGBA tuple (`to_rgba()`) or to an HTML-like hex string in the `#rrggbb` format (`to_hex()`), and a sequence of colors to an (n, 4) RGBA array (`to_rgba_array()`). Caching is used for efficiency.

Matplotlib recognizes the following formats to specify a color:

- an RGB or RGBA tuple of float values in $[0, 1]$ (e.g., `(0.1, 0.2, 0.5)` or `(0.1, 0.2, 0.5, 0.3)`);
- a hex RGB or RGBA string (e.g., `'#0F0F0F'` or `'#0F0F0F0F'`);
- a string representation of a float value in $[0, 1]$ inclusive for gray level (e.g., `'0.5'`);
- one of `{'b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'}`;
- a X11/CSS4 color name;
- a name from the [xkcd color survey](#); prefixed with `'xkcd:'` (e.g., `'xkcd:sky blue'`);

- one of {'tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown', 'tab:pink', 'tab:gray', 'tab:olive', 'tab:cyan'} which are the Tableau Colors from the 'T10' categorical palette (which is the default color cycle);
- a "CN" color spec, i.e. 'C' followed by a single digit, which is an index into the default property cycle (matplotlib.rcParams['axes.prop_cycle']); the indexing occurs at artist creation time and defaults to black if the cycle does not include color.

All string specifications of color, other than "CN", are case-insensitive.

42.1.1.1 Classes

BoundaryNorm(boundaries, ncolors[, clip])	Generate a colormap index based on discrete intervals.
Colormap(name[, N])	Baseclass for all scalar to RGBA mappings.
LightSource([azdeg, altdeg, hsv_min_val, ...])	Create a light source coming from the specified azimuth and elevation.
LinearSegmentedColormap(name, segment-data[, ...])	Colormap objects based on lookup tables using linear segments.
ListedColormap(colors[, name, N])	Colormap object generated from a list of colors.
LogNorm([vmin, vmax, clip])	Normalize a given value to the 0-1 range on a log scale
NoNorm([vmin, vmax, clip])	Dummy replacement for Normalize, for the case where we want to use indices directly in a ScalarMappable .
Normalize([vmin, vmax, clip])	A class which, when called, can normalize data into the [0.0, 1.0] interval.
PowerNorm(gamma[, vmin, vmax, clip])	Normalize a given value to the [0, 1] interval with a power-law scaling.
SymLogNorm(linthresh[, linscale, vmin, ...])	The symmetrical logarithmic scale is logarithmic in both the positive and negative directions from the origin.

matplotlib.colors.BoundaryNorm

class matplotlib.colors.BoundaryNorm(boundaries, ncolors, clip=False)

Generate a colormap index based on discrete intervals.

Unlike Normalize or LogNorm, BoundaryNorm maps values to integers instead of to the interval 0-1.

Mapping to the 0-1 interval could have been done via piece-wise linear interpolation, but using integers seems simpler, and reduces the number of conversions back and forth between integer and floating point.

Parameters

boundaries [array-like] Monotonically increasing sequence of boundaries

`ncolors` [int] Number of colors in the colormap to be used

`clip` [bool, optional] If `clip` is `True`, out of range values are mapped to 0 if they are below `boundaries[0]` or mapped to `ncolors - 1` if they are above `boundaries[-1]`.

If `clip` is `False`, out of range values are mapped to -1 if they are below `boundaries[0]` or mapped to `ncolors` if they are above `boundaries[-1]`. These are then converted to valid indices by `Colormap.__call__()`.

Notes

`boundaries` defines the edges of bins, and data falling within a bin is mapped to the color with the same index.

If the number of bins doesn't equal `ncolors`, the color is chosen by linear interpolation of the bin number onto color numbers.

`inverse(value)`

Raises

`ValueError` `BoundaryNorm` is not invertible, so calling this method will always raise an error

Examples using `matplotlib.colors.BoundaryNorm`

- [sphx_glr_gallery_lines_bars_and_markers_multicolored_line.py](#)
- [sphx_glr_gallery_images_contours_and_fields_pcolormesh_levels.py](#)
- [sphx_glr_gallery_images_contours_and_fields_image_masked.py](#)
- [sphx_glr_gallery_images_contours_and_fields_image_annotated_heatmap.py](#)
- [sphx_glr_gallery_specialty_plots_leftventricle_bulleye.py](#)
- [sphx_glr_gallery_userdemo_colormap_normalizations_bounds.py](#)
- [sphx_glr_gallery_userdemo_colormap_normalizations.py](#)
- [Customized Colorbars Tutorial](#)
- [Colormap Normalization](#)

`matplotlib.colors.Colormap`

`class matplotlib.colors.Colormap(name, N=256)`
 Baseclass for all scalar to RGBA mappings.

Typically `Colormap` instances are used to convert data values (floats) from the interval `[0, 1]` to the RGBA color that the respective `Colormap` represents. For scaling of data into

the `[0, 1]` interval see `matplotlib.colors.Normalize`. It is worth noting that `matplotlib.cm`. `ScalarMappable` subclasses make heavy use of this `data->normalize->map-to-color` processing chain.

Parameters

`name` [str] The name of the colormap.

`N` [int] The number of rgb quantization levels.

`colorbar_extend` = None

When this colormap exists on a scalar mappable and `colorbar_extend` is not `False`, colorbar creation will pick up `colorbar_extend` as the default value for the `extend` keyword in the `matplotlib.colorbar.Colorbar` constructor.

`is_gray()`

`reversed(name=None)`

Make a reversed instance of the Colormap.

Note: Function not implemented for base class.

Parameters

`name` [str, optional] The name for the reversed colormap. If it's `None` the name will be the name of the parent colormap + `"_r"`.

Notes

See `LinearSegmentedColormap.reversed()` and `ListedColormap.reversed()`

`set_bad(color='k', alpha=None)`

Set color to be used for masked values.

`set_over(color='k', alpha=None)`

Set color to be used for high out-of-range values. Requires `norm.clip = False`

`set_under(color='k', alpha=None)`

Set color to be used for low out-of-range values. Requires `norm.clip = False`

Examples using `matplotlib.colors.Colormap`

- `sphx_glr_gallery_images_contours_and_fields_contourf_demo.py`

`matplotlib.colors.LightSource`

```
class matplotlib.colors.LightSource(azdeg=315,          altdeg=45,          hsv_min_val=0,
                                   hsv_max_val=1, hsv_min_sat=1, hsv_max_sat=0)
```

Create a light source coming from the specified azimuth and elevation. Angles are in degrees, with the azimuth measured clockwise from north and elevation up from the zero plane of the surface.

The `shade()` is used to produce "shaded" rgb values for a data array. `shade_rgb()` can be used to combine an rgb image with The `shade_rgb()` The `hillshade()` produces an illumination map of a surface.

Specify the azimuth (measured clockwise from south) and altitude (measured up from the plane of the surface) of the light source in degrees.

Parameters

`azdeg` [number, optional] The azimuth (0-360, degrees clockwise from North) of the light source. Defaults to 315 degrees (from the northwest).

`altdeg` [number, optional] The altitude (0-90, degrees up from horizontal) of the light source. Defaults to 45 degrees from horizontal.

Notes

For backwards compatibility, the parameters `hsv_min_val`, `hsv_max_val`, `hsv_min_sat`, and `hsv_max_sat` may be supplied at initialization as well. However, these parameters will only be used if "`blend_mode='hsv'`" is passed into `shade()` or `shade_rgb()`. See the documentation for `blend_hsv()` for more details.

```
blend_hsv(rgb,          intensity,          hsv_max_sat=None,          hsv_max_val=None,
          hsv_min_val=None, hsv_min_sat=None)
```

Take the input data array, convert to HSV values in the given colormap, then adjust those color values to give the impression of a shaded relief map with a specified light source. RGBA values are returned, which can then be used to plot the shaded image with `imshow`.

The color of the resulting image will be darkened by moving the (s,v) values (in hsv colorspace) toward (`hsv_min_sat`, `hsv_min_val`) in the shaded regions, or lightened by sliding (s,v) toward (`hsv_max_sat` `hsv_max_val`) in regions that are illuminated. The default extremes are chose so that completely shaded points are nearly black (`s = 1`, `v = 0`) and completely illuminated points are nearly white (`s = 0`, `v = 1`).

Parameters

`rgb` [ndarray] An MxNx3 RGB array of floats ranging from 0 to 1 (color image).

`intensity` [ndarray] An MxNx1 array of floats ranging from 0 to 1 (grayscale image).

`hsv_max_sat` [number, optional] The maximum saturation value that the intensity map can shift the output image to. Defaults to 1.

`hsv_min_sat` [number, optional] The minimum saturation value that the intensity map can shift the output image to. Defaults to 0.

`hsv_max_val` [number, optional] The maximum value ("v" in "hsv") that the intensity map can shift the output image to. Defaults to 1.

`hsv_min_val`: number, optional The minimum value ("v" in "hsv") that the intensity map can shift the output image to. Defaults to 0.

Returns

`rgb` [ndarray] An MxNx3 RGB array representing the combined images.

`blend_overlay(rgb, intensity)`

Combines an rgb image with an intensity map using "overlay" blending.

Parameters

`rgb` [ndarray] An MxNx3 RGB array of floats ranging from 0 to 1 (color image).

`intensity` [ndarray] An MxNx1 array of floats ranging from 0 to 1 (grayscale image).

Returns

`rgb` [ndarray] An MxNx3 RGB array representing the combined images.

`blend_soft_light(rgb, intensity)`

Combines an rgb image with an intensity map using "soft light" blending. Uses the "pegtop" formula.

Parameters

`rgb` [ndarray] An MxNx3 RGB array of floats ranging from 0 to 1 (color image).

`intensity` [ndarray] An MxNx1 array of floats ranging from 0 to 1 (grayscale image).

Returns

`rgb` [ndarray] An MxNx3 RGB array representing the combined images.

`direction`

The unit vector direction towards the light source

`hillshade(elevation, vert_exag=1, dx=1, dy=1, fraction=1.0)`

Calculates the illumination intensity for a surface using the defined azimuth and elevation for the light source.

This computes the normal vectors for the surface, and then passes them on to `shade_normals`

Parameters

`elevation` [array-like] A 2d array (or equivalent) of the height values used to generate an illumination map

`vert_exag` [number, optional] The amount to exaggerate the elevation values by when calculating illumination. This can be used either to correct for differences in units between the x-y coordinate system and the elevation coordinate system (e.g. decimal degrees vs meters) or to exaggerate or de-emphasize topographic effects.

`dx` [number, optional] The x-spacing (columns) of the input elevation grid.

`dy` [number, optional] The y-spacing (rows) of the input elevation grid.

`fraction` [number, optional] Increases or decreases the contrast of the hillshade. Values greater than one will cause intermediate values to move closer to full illumination or shadow (and clipping any values that move beyond 0 or 1). Note that this is not visually or mathematically the same as vertical exaggeration.

Returns

`intensity` [ndarray] A 2d array of illumination values between 0-1, where 0 is completely in shadow and 1 is completely illuminated.

`shade(data, cmap, norm=None, blend_mode='overlay', vmin=None, vmax=None, vert_exag=1, dx=1, dy=1, fraction=1, **kwargs)`
Combine colormapped data values with an illumination intensity map (a.k.a. "hillshade") of the values.

Parameters

`data` [array-like] A 2d array (or equivalent) of the height values used to generate a shaded map.

`cmap` [Colormap instance] The colormap used to color the data array. Note that this must be a Colormap instance. For example, rather than passing in `cmap='gist_earth'`, use `cmap=plt.get_cmap('gist_earth')` instead.

`norm` [Normalize instance, optional] The normalization used to scale values before colormapping. If None, the input will be linearly scaled between its min and max.

`blend_mode` [{`'hsv'`, `'overlay'`, `'soft'`} or callable, optional] The type of blending used to combine the colormapped data values with the illumination intensity. Default is "overlay". Note that for most topographic surfaces, "overlay" or "soft" appear more visually realistic. If a user-defined function is supplied, it is expected to combine an `MxNx3` RGB array of floats (ranging 0 to 1) with an `MxNx1` hillshade array (also 0 to 1). (Call signature `func(rgb, illum, **kwargs)`) Additional kwargs supplied to this function will be passed on to the `blend_mode` function.

`vmin` [scalar or None, optional] The minimum value used in colormapping data. If None the minimum value in data is used. If `norm` is specified, then this argument will be ignored.

`vmax` [scalar or None, optional] The maximum value used in colormapping data. If None the maximum value in data is used. If `norm` is specified, then this argument will be ignored.

`vert_exag` [number, optional] The amount to exaggerate the elevation values by when calculating illumination. This can be used either to correct for differences in units between the x-y coordinate system and the elevation coordinate system (e.g. decimal degrees vs meters) or to exaggerate or de-emphasize topography.

`dx` [number, optional] The x-spacing (columns) of the input elevation grid.

`dy` [number, optional] The y-spacing (rows) of the input elevation grid.

`fraction` [number, optional] Increases or decreases the contrast of the hillshade. Values greater than one will cause intermediate values to move closer to full illumination or shadow (and clipping any values that move beyond 0 or 1). Note that this is not visually or mathematically the same as vertical exaggeration.

Additional kwargs are passed on to the `*blend_mode*` function.

Returns

`rgba` [ndarray] An $M \times N \times 4$ array of floats ranging between 0-1.

`shade_normals(normals, fraction=1.0)`

Calculates the illumination intensity for the normal vectors of a surface using the defined azimuth and elevation for the light source.

Imagine an artificial sun placed at infinity in some azimuth and elevation position illuminating our surface. The parts of the surface that slope toward the sun should brighten while those sides facing away should become darker.

Parameters

`fraction` [number, optional] Increases or decreases the contrast of the hillshade. Values greater than one will cause intermediate values to move closer to full illumination or shadow (and clipping any values that move beyond 0 or 1). Note that this is not visually or mathematically the same as vertical exaggeration.

Returns

`intensity` [ndarray] A 2d array of illumination values between 0-1, where 0 is completely in shadow and 1 is completely illuminated.

`shade_rgb(rgb, elevation, fraction=1.0, blend_mode='hsv', vert_exag=1, dx=1, dy=1, **kwargs)`

Take the input RGB array ($ny \times nx \times 3$) adjust their color values to given the impression of a shaded relief map with a specified light source using the elevation ($ny \times nx$). A new RGB array ($(ny \times nx \times 3)$) is returned.

Parameters

`rgb` [array-like] An MxNx3 RGB array, assumed to be in the range of 0 to 1.

`elevation` [array-like] A 2d array (or equivalent) of the height values used to generate a shaded map.

`fraction` [number] Increases or decreases the contrast of the hillshade. Values greater than one will cause intermediate values to move closer to full illumination or shadow (and clipping any values that move beyond 0 or 1). Note that this is not visually or mathematically the same as vertical exaggeration.

`blend_mode` [{`'hsv'`, `'overlay'`, `'soft'`} or callable, optional] The type of blending used to combine the colormapped data values with the illumination intensity. For backwards compatibility, this defaults to `"hsv"`. Note that for most topographic surfaces, `"overlay"` or `"soft"` appear more visually realistic. If a user-defined function is supplied, it is expected to combine an MxNx3 RGB array of floats (ranging 0 to 1) with an MxNx1 hillshade array (also 0 to 1). (Call signature `func(rgb, illum, **kwargs)`) Additional kwargs supplied to this function will be passed on to the `blend_mode` function.

`vert_exag` [number, optional] The amount to exaggerate the elevation values by when calculating illumination. This can be used either to correct for differences in units between the x-y coordinate system and the elevation coordinate system (e.g. decimal degrees vs meters) or to exaggerate or de-emphasize topography.

`dx` [number, optional] The x-spacing (columns) of the input elevation grid.

`dy` [number, optional] The y-spacing (rows) of the input elevation grid.

Additional kwargs are passed on to the `*blend_mode*` function.

Returns

`shaded_rgb` [ndarray] An MxNx3 array of floats ranging between 0-1.

Examples using `matplotlib.colors.LightSource`

- `sphx_glr_gallery_images_contours_and_fields_shading_example.py`
- `sphx_glr_gallery_showcase_mandelbrot.py`
- `sphx_glr_gallery_frontpage_3D.py`
- `sphx_glr_gallery_misc_demo_agg_filter.py`
- `sphx_glr_gallery_mplot3d_custom_shaded_3d_surface.py`
- `sphx_glr_gallery_specialty_plots_advanced_hillshading.py`
- `sphx_glr_gallery_specialty_plots_topographic_hillshading.py`

`matplotlib.colors.LinearSegmentedColormap`

`class matplotlib.colors.LinearSegmentedColormap(name, segmentdata, N=256, gamma=1.0)`

Colormap objects based on lookup tables using linear segments.

The lookup table is generated using linear interpolation for each primary color, with the 0-1 domain divided into any number of segments.

Create color map from linear mapping segments

`segmentdata` argument is a dictionary with a red, green and blue entries. Each entry should be a list of x, y0, y1 tuples, forming rows in a table. Entries for alpha are optional.

Example: suppose you want red to increase from 0 to 1 over the bottom half, green to do the same over the middle half, and blue over the top half. Then you would use:

```
cdict = {'red': [(0.0, 0.0, 0.0),
                 (0.5, 1.0, 1.0),
                 (1.0, 1.0, 1.0)],
         'green': [(0.0, 0.0, 0.0),
                   (0.25, 0.0, 0.0),
                   (0.75, 1.0, 1.0),
                   (1.0, 1.0, 1.0)],
         'blue': [(0.0, 0.0, 0.0),
                  (0.5, 0.0, 0.0),
                  (1.0, 1.0, 1.0)]}
```

Each row in the table for a given color is a sequence of x, y0, y1 tuples. In each sequence, x must increase monotonically from 0 to 1. For any input value z falling between x[i] and x[i+1], the output value of a given color will be linearly interpolated between y1[i] and y0[i+1]:

```
row i:  x  y0  y1
        /
        /
row i+1: x  y0  y1
```

Hence y0 in the first row and y1 in the last row are never used.

See also:

`LinearSegmentedColormap.from_list()` Static method; factory function for generating a smoothly-varying `LinearSegmentedColormap`.

`makeMappingArray()` For information about making a mapping array.

`static from_list(name, colors, N=256, gamma=1.0)`

Make a linear segmented colormap with name from a sequence of colors which evenly transitions from `colors[0]` at `val=0` to `colors[-1]` at `val=1`. N is the number of rgb quantization levels. Alternatively, a list of (value, color) tuples can be given to divide the range unevenly.

`reversed(name=None)`

Make a reversed instance of the Colormap.

Parameters

`name` [str, optional] The name for the reversed colormap. If it's None the name will be the name of the parent colormap + "_r".

Returns

`LinearSegmentedColormap` The reversed colormap.

`set_gamma(gamma)`

Set a new gamma value and regenerate color map.

Examples using `matplotlib.colors.LinearSegmentedColormap`

- `sphx_glr_gallery_images_contours_and_fields_custom_cmap.py`

`matplotlib.colors.ListedColormap`

`class matplotlib.colors.ListedColormap(colors, name='from_list', N=None)`

Colormap object generated from a list of colors.

This may be most useful when indexing directly into a colormap, but it can also be used to generate special colormaps for ordinary mapping.

Make a colormap from a list of colors.

`colors` a list of matplotlib color specifications, or an equivalent Nx3 or Nx4 floating point array (N rgb or rgba values)

`name` a string to identify the colormap

`N` the number of entries in the map. The default is None, in which case there is one colormap entry for each element in the list of colors. If:

`N < len(colors)`

the list will be truncated at N. If:

`N > len(colors)`

the list will be extended by repetition.

`reversed(name=None)`

Make a reversed instance of the Colormap.

Parameters

`name` [str, optional] The name for the reversed colormap. If it's None the name will be the name of the parent colormap + "_r".

Returns

ListedColormap A reversed instance of the colormap.

Examples using matplotlib.colors.ListedColormap

- [sphx_glr_gallery_lines_bars_and_markers_multicolored_line.py](#)
- [sphx_glr_gallery_specialty_plots_leftventricle_bulleye.py](#)
- [Customized Colorbars Tutorial](#)

matplotlib.colors.LogNorm

class matplotlib.colors.LogNorm(vmin=None, vmax=None, clip=False)

Normalize a given value to the 0-1 range on a log scale

If vmin or vmax is not given, they are initialized from the minimum and maximum value respectively of the first input processed. That is, `__call__(A)` calls `autoscale_None(A)`. If clip is True and the given value falls outside the range, the returned value will be 0 or 1, whichever is closer. Returns 0 if:

`vmin==vmax`

Works with scalars or arrays, including masked arrays. If clip is True, masked values are set to 1; otherwise they remain masked. Clipping silently defeats the purpose of setting the over, under, and masked colors in the colormap, so it is likely to lead to surprises; therefore the default is clip = False.

`autoscale(A)`

Set vmin, vmax to min, max of A.

`autoscale_None(A)`

autoscale only None-valued vmin or vmax.

`inverse(value)`

Examples using matplotlib.colors.LogNorm

- [sphx_glr_gallery_statistics_hist.py](#)
- [sphx_glr_gallery_images_contours_and_fields_pcolor_demo.py](#)
- [sphx_glr_gallery_userdemo_colormap_normalizations_lognorm.py](#)
- [sphx_glr_gallery_userdemo_colormap_normalizations.py](#)
- [Colormap Normalization](#)

`matplotlib.colors.NoNorm`

```
class matplotlib.colors.NoNorm(vmin=None, vmax=None, clip=False)
```

Dummy replacement for Normalize, for the case where we want to use indices directly in a `ScalarMappable`.

If `vmin` or `vmax` is not given, they are initialized from the minimum and maximum value respectively of the first input processed. That is, `__call__(A)` calls `autoscale_None(A)`. If `clip` is `True` and the given value falls outside the range, the returned value will be 0 or 1, whichever is closer. Returns 0 if:

```
vmin==vmax
```

Works with scalars or arrays, including masked arrays. If `clip` is `True`, masked values are set to 1; otherwise they remain masked. Clipping silently defeats the purpose of setting the over, under, and masked colors in the colormap, so it is likely to lead to surprises; therefore the default is `clip = False`.

```
inverse(value)
```

`matplotlib.colors.Normalize`

```
class matplotlib.colors.Normalize(vmin=None, vmax=None, clip=False)
```

A class which, when called, can normalize data into the `[0.0, 1.0]` interval.

If `vmin` or `vmax` is not given, they are initialized from the minimum and maximum value respectively of the first input processed. That is, `__call__(A)` calls `autoscale_None(A)`. If `clip` is `True` and the given value falls outside the range, the returned value will be 0 or 1, whichever is closer. Returns 0 if:

```
vmin==vmax
```

Works with scalars or arrays, including masked arrays. If `clip` is `True`, masked values are set to 1; otherwise they remain masked. Clipping silently defeats the purpose of setting the over, under, and masked colors in the colormap, so it is likely to lead to surprises; therefore the default is `clip = False`.

```
autoscale(A)
```

Set `vmin`, `vmax` to min, max of `A`.

```
autoscale_None(A)
```

autoscale only None-valued `vmin` or `vmax`.

```
inverse(value)
```

```
static process_value(value)
```

Homogenize the input value for easy and efficient normalization.

`value` can be a scalar or sequence.

Returns result, is_scalar, where result is a masked array matching value. Float dtypes are preserved; integer types with two bytes or smaller are converted to np.float32, and larger types are converted to np.float64. Preserving float32 when possible, and using in-place operations, can greatly improve speed for large arrays.

Experimental; we may want to add an option to force the use of float32.

```
scaled()  
    return true if vmin and vmax set
```

Examples using `matplotlib.colors.Normalize`

- [sphx_glr_gallery_statistics_hist.py](#)
- [sphx_glr_gallery_images_contours_and_fields_multi_image.py](#)
- [sphx_glr_gallery_images_contours_and_fields_image_masked.py](#)
- [sphx_glr_gallery_images_contours_and_fields_image_transparency_blend.py](#)
- [sphx_glr_gallery_images_contours_and_fields_contour_image.py](#)
- [sphx_glr_gallery_specialty_plots_advanced_hillshading.py](#)
- [sphx_glr_gallery_specialty_plots_leftventricle_bulleye.py](#)
- [sphx_glr_gallery_userdemo_colormap_normalizations_custom.py](#)
- [sphx_glr_gallery_userdemo_colormap_normalizations.py](#)
- [Constrained Layout Guide](#)
- [Customized Colorbars Tutorial](#)
- [Colormap Normalization](#)

`matplotlib.colors.PowerNorm`

```
class matplotlib.colors.PowerNorm(gamma, vmin=None, vmax=None, clip=False)  
    Normalize a given value to the [0, 1] interval with a power-law scaling. This will clip any  
    negative data points to 0.  
  
    autoscale(A)  
        Set vmin, vmax to min, max of A.  
  
    autoscale_None(A)  
        autoscale only None-valued vmin or vmax.  
  
    inverse(value)
```

Examples using `matplotlib.colors.PowerNorm`

- [sphx_glr_gallery_api_power_norm.py](#)
- [sphx_glr_gallery_showcase_mandelbrot.py](#)
- [sphx_glr_gallery_userdemo_colormap_normalizations_power.py](#)
- [sphx_glr_gallery_userdemo_colormap_normalizations.py](#)
- [Colormap Normalization](#)

`matplotlib.colors.SymLogNorm`

```
class matplotlib.colors.SymLogNorm(linthresh, linscale=1.0, vmin=None, vmax=None,
                                   clip=False)
```

The symmetrical logarithmic scale is logarithmic in both the positive and negative directions from the origin.

Since the values close to zero tend toward infinity, there is a need to have a range around zero that is linear. The parameter `linthresh` allows the user to specify the size of this range `(-linthresh, linthresh)`.

`linthresh`: The range within which the plot is linear (to avoid having the plot go to infinity around zero).

`linscale`: This allows the linear range `(-linthresh to linthresh)` to be stretched relative to the logarithmic range. Its value is the number of decades to use for each half of the linear range. For example, when `linscale == 1.0` (the default), the space used for the positive and negative halves of the linear range will be equal to one decade in the logarithmic range. Defaults to 1.

```
autoscale(A)
```

Set `vmin`, `vmax` to min, max of `A`.

```
autoscale_None(A)
```

autoscale only None-valued `vmin` or `vmax`.

```
inverse(value)
```

Examples using `matplotlib.colors.SymLogNorm`

- [sphx_glr_gallery_userdemo_colormap_normalizations_symlognorm.py](#)
- [sphx_glr_gallery_userdemo_colormap_normalizations.py](#)
- [Colormap Normalization](#)

42.1.2 Functions

<code>from_levels_and_colors(levels, colors[, extend])</code>	A helper routine to generate a cmap and a norm instance which behave similar to <code>contourf</code> 's levels and colors arguments.
<code>hsv_to_rgb(hsv)</code>	convert hsv values in a numpy array to rgb values all values assumed to be in range <code>[0, 1]</code>
<code>rgb_to_hsv(arr)</code>	convert float rgb values (in the range <code>[0, 1]</code>), in a numpy array to hsv values.
<code>to_hex(c[, keep_alpha])</code>	Convert <code>c</code> to a hex color.
<code>to_rgb(c)</code>	Convert <code>c</code> to an RGB color, silently dropping the alpha channel.
<code>to_rgba(c[, alpha])</code>	Convert <code>c</code> to an RGBA color.
<code>to_rgba_array(c[, alpha])</code>	Convert <code>c</code> to a <code>(n, 4)</code> array of RGBA colors.
<code>is_color_like(c)</code>	Return whether <code>c</code> can be interpreted as an RGB(A) color.
<code>makeMappingArray(N, data[, gamma])</code>	Create an <code>N</code> -element 1-d lookup table
<code>get_named_colors_mapping()</code>	Return the global mapping of names to named colors.

`matplotlib.colors.from_levels_and_colors`

`matplotlib.colors.from_levels_and_colors(levels, colors, extend='neither')`

A helper routine to generate a cmap and a norm instance which behave similar to `contourf`'s levels and colors arguments.

Parameters

`levels` [sequence of numbers] The quantization levels used to construct the `BoundaryNorm`. Values `v` are quantized to level `i` if `lev[i] <= v < lev[i+1]`.

`colors` [sequence of colors] The fill color to use for each level. If `extend` is "neither" there must be `n_level - 1` colors. For an `extend` of "min" or "max" add one extra color, and for an `extend` of "both" add two colors.

`extend` [{`'neither'`, `'min'`, `'max'`, `'both'`}, optional] The behaviour when a value falls out of range of the given levels. See `contourf()` for details.

Returns

`(cmap, norm)` [tuple containing a `Colormap` and a `Normalize` instance]

`matplotlib.colors.hsv_to_rgb`

`matplotlib.colors.hsv_to_rgb(hsv)`

convert hsv values in a numpy array to rgb values all values assumed to be in range `[0, 1]`

Parameters

`hsv` [(..., 3) array-like] All values assumed to be in range `[0, 1]`

Returns

rgb [(..., 3) ndarray] Colors converted to RGB values in range [0, 1]

`matplotlib.colors.rgb_to_hsv`

`matplotlib.colors.rgb_to_hsv(arr)`

convert float rgb values (in the range [0, 1]), in a numpy array to hsv values.

Parameters

arr [(..., 3) array-like] All values must be in the range [0, 1]

Returns

hsv [(..., 3) ndarray] Colors converted to hsv values in range [0, 1]

Examples using `matplotlib.colors.rgb_to_hsv`

- `sphx_glr_gallery_color_named_colors.py`

`matplotlib.colors.to_hex`

`matplotlib.colors.to_hex(c, keep_alpha=False)`

Convert c to a hex color.

Uses the `#rrggbb` format if `keep_alpha` is `False` (the default), `#rrggbbbaa` otherwise.

`matplotlib.colors.to_rgb`

`matplotlib.colors.to_rgb(c)`

Convert c to an RGB color, silently dropping the alpha channel.

`matplotlib.colors.to_rgba`

`matplotlib.colors.to_rgba(c, alpha=None)`

Convert c to an RGBA color.

Parameters

c [Matplotlib color]

alpha [scalar, optional] If alpha is not `None`, it forces the alpha value, except if c is "none" (case-insensitive), which always maps to (0, 0, 0, 0).

Returns

tuple Tuple of (r, g, b, a) scalars.

Examples using `matplotlib.colors.to_rgba`

- `sphinx_glr_gallery_color_named_colors.py`
- `sphinx_glr_gallery_shapes_and_collections_line_collection.py`
- `sphinx_glr_gallery_shapes_and_collections_collections.py`
- `sphinx_glr_gallery_event_handling_lasso_demo.py`
- `sphinx_glr_gallery_mplot3d_polys3d.py`
- `sphinx_glr_gallery_widgets_menu.py`

`matplotlib.colors.to_rgba_array`

`matplotlib.colors.to_rgba_array(c, alpha=None)`
Convert `c` to a `(n, 4)` array of RGBA colors.

If `alpha` is not `None`, it forces the `alpha` value. If `c` is "none" (case-insensitive) or an empty list, an empty array is returned.

`matplotlib.colors.is_color_like`

`matplotlib.colors.is_color_like(c)`
Return whether `c` can be interpreted as an RGB(A) color.

`matplotlib.colors.makeMappingArray`

`matplotlib.colors.makeMappingArray(N, data, gamma=1.0)`
Create an `N`-element 1-d lookup table

`data` represented by a list of `x,y0,y1` mapping correspondences. Each element in this list represents how a value between 0 and 1 (inclusive) represented by `x` is mapped to a corresponding value between 0 and 1 (inclusive). The two values of `y` are to allow for discontinuous mapping functions (say as might be found in a sawtooth) where `y0` represents the value of `y` for values of `x` \leq to that given, and `y1` is the value to be used for `x` $>$ than that given). The list must start with `x=0`, end with `x=1`, and all values of `x` must be in increasing order. Values between the given mapping points are determined by simple linear interpolation.

Alternatively, `data` can be a function mapping values between 0 - 1 to 0 - 1.

The function returns an array "result" where `result[x*(N-1)]` gives the closest value for values of `x` between 0 and 1.

`matplotlib.colors.get_named_colors_mapping`

`matplotlib.colors.get_named_colors_mapping()`
Return the global mapping of names to named colors.

CONTOUR

43.1 matplotlib.contour

These are classes to support contour plotting and labelling for the Axes class.

```
class matplotlib.contour.ClabelText(x=0, y=0, text="", color=None, verticalalign='baseline', horizontalalignment='left', multialignment=None, fontproperties=None, rotation=None, linespacing=None, rotation_mode=None, usetex=None, wrap=False, **kwargs)
```

Bases: matplotlib.text.Text

Unlike the ordinary text, the `get_rotation` returns an updated angle in the pixel coordinate assuming that the input rotation is an angle in data coordinate (or whatever transform set).

Create a Text instance at `x, y` with string `text`.

Valid kwargs are

	Property	Description
	<code>agg_filter</code>	a filter function, which takes a (m, n, 3) float array and a dpi value
	<code>alpha</code>	float (0.0 transparent through 1.0 opaque)
	<code>animated</code>	bool
	<code>backgroundcolor</code>	any matplotlib color
	<code>bbox</code>	FancyBboxPatch prop dict
	<code>clip_box</code>	a matplotlib.transforms.Bbox instance
	<code>clip_on</code>	bool
	<code>clip_path</code>	[(Path, Transform) Patch None]
	<code>color</code>	any matplotlib color
	<code>contains</code>	a callable function
	<code>family</code> or <code>fontfamily</code> or <code>fontname</code> or <code>name</code>	[FONTNAME 'serif' 'sans-serif' 'cursive' 'fantasy' 'monospace']
	<code>figure</code>	a Figure instance
	<code>fontproperties</code> or <code>font_properties</code>	a matplotlib.font_manager.FontProperties instance
	<code>gid</code>	an id string
	<code>horizontalalignment</code> or <code>ha</code>	['center' 'right' 'left']
	<code>label</code>	object

Continued on next page

Table 1 – continued from previous page

	Property	Description
linespacing		float (multiple of font size)
multialignment or ma		['left' 'right' 'center']
path_effects		AbstractPathEffect
picker		[None bool float callable]
position		(x,y)
rasterized		bool or None
rotation		[angle in degrees 'vertical' 'horizontal']
rotation_mode		[None "default" "anchor"]
size or fontsize		[size in points 'xx-small' 'x-small' 'small' 'medium' 'large' 'x-large']
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
stretch or fontstretch		[a numeric value in range 0-1000 'ultra-condensed' 'extra-condensed' 'normal' 'condensed' 'expanded']
style or fontstyle		['normal' 'italic' 'oblique']
text		string or anything printable with '%s' conversion.
transform		Transform
url		a url string
usetex		bool or None
variant or fontvariant		['normal' 'small-caps']
verticalalignment or va		['center' 'top' 'bottom' 'baseline']
visible		bool
weight or fontweight		[a numeric value in range 0-1000 'ultralight' 'light' 'normal' 'bold' 'extra-bold']
wrap		bool
x		float
y		float
zorder		float

`get_rotation()`
 return the text angle as float in degrees

class `matplotlib.contour.ContourLabeler`

Bases: `object`

Mixin to provide labelling capability to `ContourSet`.

`add_label(x, y, rotation, lev, cvalue)`
 Add contour label using `Text` class.

`add_label_clabeltext(x, y, rotation, lev, cvalue)`
 Add contour label using `ClabelText` class.

`add_label_near(x, y, inline=True, inline_spacing=5, transform=None)`
 Add a label near the point (x, y). If transform is None (default), (x, y) is in data coordinates; if transform is False, (x, y) is in display coordinates; otherwise, the specified transform will be used to translate (x, y) into display coordinates.

Parameters

`x, y` [float] The approximate location of the label.

`inline` [bool, optional, default: True] If True remove the segment of the contour beneath the label.

`inline_spacing` [int, optional, default: 5] Space in pixels to leave on each side of label when placing inline. This spacing will be exact for labels at locations where the contour is straight, less so for labels on curved contours.

`calc_label_rot_and_inline(slc, ind, lw, lc=None, spacing=5)`

This function calculates the appropriate label rotation given the linecontour coordinates in screen units, the index of the label location and the label width.

It will also break contour and calculate inlining if `lc` is not empty (`lc` defaults to the empty list if None). `spacing` is the space around the label in pixels to leave empty.

Do both of these tasks at once to avoid calculating path lengths multiple times, which is relatively costly.

The method used here involves calculating the path length along the contour in pixel coordinates and then looking approximately label width / 2 away from central point to determine rotation and then to break contour if desired.

`clabel(*args, **kwargs)`

Label a contour plot.

Call signature:

```
clabel(cs, **kwargs)
```

Adds labels to line contours in `cs`, where `cs` is a `ContourSet` object returned by `contour`.

```
clabel(cs, v, **kwargs)
```

only labels contours listed in `v`.

Parameters

`fontsize` [string or float, optional] Size in points or relative size e.g., 'smaller', 'x-large'. See `Text.set_size` for accepted string values.

`colors` : Color of each label

- if None, the color of each label matches the color of the corresponding contour
- if one string color, e.g., `colors = 'r'` or `colors = 'red'`, all labels will be plotted in this color
- if a tuple of matplotlib color args (string, float, rgb, etc), different labels will be plotted in different colors in the order specified

`inline` [bool, optional] If True the underlying contour is removed where the label is placed. Default is True.

`inline_spacing` [float, optional] Space in pixels to leave on each side of label when placing inline. Defaults to 5.

This spacing will be exact for labels at locations where the contour is straight, less so for labels on curved contours.

`fmt` [string or dict, optional] A format string for the label. Default is '%1.3f'

Alternatively, this can be a dictionary matching contour levels with arbitrary strings to use for each contour level (i.e., `fmt[level]=string`), or it can be any callable, such as a `Formatter` instance, that returns a string when called with a numeric contour level.

`manual` [bool or iterable, optional] If `True`, contour labels will be placed manually using mouse clicks. Click the first button near a contour to add a label, click the second button (or potentially both mouse buttons at once) to finish adding labels. The third button can be used to remove the last label added, but only if labels are not inline. Alternatively, the keyboard can be used to select label locations (enter to end label placement, delete or backspace act like the third mouse button, and any other key will select a label location).

`manual` can also be an iterable object of `x,y` tuples. Contour labels will be created as if mouse is clicked at each `x,y` positions.

`rightside_up` [bool, optional] If `True`, label rotations will always be plus or minus 90 degrees from level. Default is `True`.

`use_clabeltext` [bool, optional] If `True`, `ClabelText` class (instead of `Text`) is used to create labels. `ClabelText` recalculates rotation angles of texts during the drawing time, therefore this can be used if aspect of the axes changes. Default is `False`.

`get_label_coords(distances, XX, YY, ysize, lw)`

Return `x`, `y`, and the index of a label location.

Labels are plotted at a location with the smallest deviation of the contour from a straight line unless there is another label nearby, in which case the next best place on the contour is picked up. If all such candidates are rejected, the beginning of the contour is chosen.

`get_label_width(lev, fmt, fsize)`

Return the width of the label in points.

`get_real_label_width(lev, fmt, fsize)`

Deprecated since version 2.2: The `get_real_label_width` function was deprecated in version 2.2.

This computes actual onscreen label width. This uses some black magic to determine onscreen extent of non-drawn label. This magic may not be very robust.

This method is not being used, and may be modified or removed.

`get_text(lev, fmt)`

Get the text of the label.

`labels(inline, inline_spacing)`

`locate_label(linecontour, labelwidth)`

Find good place to draw a label (relatively flat part of the contour).

`pop_label(index=-1)`

Defaults to removing last label, but any index can be supplied

`print_label(linecontour, labelwidth)`

Return False if contours are too short for a label.

`set_label_props(label, text, color)`

Set the label properties - color, fontsize, text.

`too_close(x, y, lw)`

Return True if a label is already near this location.

`class matplotlib.contour.ContourSet(ax, *args, **kwargs)`

Bases: `matplotlib.cm.ScalarMappable`, `matplotlib.contour.ContourLabeler`

Store a set of contour lines or filled regions.

User-callable method: `clabel`

Parameters

`ax` [Axes]

`levels` [[level0, level1, ..., leveln]] A list of floating point numbers indicating the contour levels.

`allsegs` [[level0segs, level1segs, ...]] List of all the polygon segments for all the levels. For contour lines `len(allsegs) == len(levels)`, and for filled contour regions `len(allsegs) = len(levels)-1`. The lists should look like:

```
level0segs = [polygon0, polygon1, ...]
polygon0 = array_like [[x0,y0], [x1,y1], ...]
```

`allkinds` [None or [level0kinds, level1kinds, ...]] Optional list of all the polygon vertex kinds (code types), as described and used in `Path`. This is used to allow multiply- connected paths such as holes within filled polygons. If not None, `len(allkinds) == len(allsegs)`. The lists should look like:

```
level0kinds = [polygon0kinds, ...]
polygon0kinds = [vertexcode0, vertexcode1, ...]
```

If `allkinds` is not None, usually all polygons for a particular contour level are grouped together so that `level0segs = [polygon0]` and `level0kinds = [polygon0kinds]`.

`kwargs` : Keyword arguments are as described in the docstring of `contour`.

Attributes

`ax`: The axes object in which the contours are drawn.

collections: A silent_list of LineCollections or PolyCollections.

levels: Contour levels.

layers: Same as levels for line contours; half-way between levels for filled contours. See `_process_colors()`.

Draw contour lines or filled regions, depending on whether keyword arg `filled` is `False` (default) or `True`.

Call signature:

`ContourSet(ax, levels, allsegs, [allkinds], **kwargs)`

Parameters

`ax` : The Axes object to draw on.

`levels` `[[level0, level1, ..., leveln]]` A list of floating point numbers indicating the contour levels.

`allsegs` `[[level0segs, level1segs, ...]]` List of all the polygon segments for all the levels. For contour lines `len(allsegs) == len(levels)`, and for filled contour regions `len(allsegs) = len(levels)-1`. The lists should look like:

`level0segs = [polygon0, polygon1, ...]`
`polygon0 = array_like [[x0,y0], [x1,y1], ...]`

`allkinds` `[[level0kinds, level1kinds, ...], optional]` Optional list of all the polygon vertex kinds (code types), as described and used in `Path`. This is used to allow multiply- connected paths such as holes within filled polygons. If not `None`, `len(allkinds) == len(allsegs)`. The lists should look like:

`level0kinds = [polygon0kinds, ...]`
`polygon0kinds = [vertexcode0, vertexcode1, ...]`

If `allkinds` is not `None`, usually all polygons for a particular contour level are grouped together so that `level0segs = [polygon0]` and `level0kinds = [polygon0kinds]`.

`**kwargs` Keyword arguments are as described in the docstring of `contour`.

`changed()`

Call this whenever the mappable is changed to notify all the callbackSM listeners to the 'changed' signal

`find_nearest_contour(x, y, indices=None, pixel=True)`

Finds contour that is closest to a point. Defaults to measuring distance in pixels (screen space - useful for manual contour labeling), but this can be controlled via a keyword argument.

Returns a tuple containing the contour, segment, index of segment, x & y of segment point and distance to minimum point.

Optional keyword arguments:

`indices`: Indexes of contour levels to consider when looking for nearest point. Defaults to using all levels.

`pixel`: If True, measure distance in pixel space, if not, measure distance in axes space. Defaults to True.

`get_alpha()`

returns alpha to be applied to all ContourSet artists

`get_transform()`

Return the Transform instance used by this ContourSet.

`legend_elements(variable_name='x', str_format=<class 'str'>)`

Return a list of artists and labels suitable for passing through to `plt.legend()` which represent this ContourSet.

The labels have the form "`0 < x <= 1`" stating the data ranges which the artists represent.

Parameters

`variable_name` [str] The string used inside the inequality used on the labels.

`str_format` [function: float -> str] Function used to format the numbers in the labels.

Returns

`artists` [List[Artist]] A list of the artists.

`labels` [List[str]] A list of the labels.

`set_alpha(alpha)`

Set the alpha blending value for all ContourSet artists. alpha must be between 0 (transparent) and 1 (opaque).

`class matplotlib.contour.QuadContourSet(ax, *args, **kwargs)`

Bases: `matplotlib.contour.ContourSet`

Create and store a set of contour lines or filled regions.

User-callable method: `clabel`

Attributes

`ax`: The axes object in which the contours are drawn.

`collections`: A `silent_list` of `LineCollections` or `PolyCollections`.

`levels`: Contour levels.

`layers`: Same as levels for line contours; half-way between levels for filled contours. See `__process_colors()` method.

Draw contour lines or filled regions, depending on whether keyword arg `filled` is `False` (default) or `True`.

Call signature:

`ContourSet(ax, levels, allsegs, [allkinds], **kwargs)`

Parameters

`ax` : The Axes object to draw on.

`levels` `[[level0, level1, ..., leveln]]` A list of floating point numbers indicating the contour levels.

`allsegs` `[[level0segs, level1segs, ...]]` List of all the polygon segments for all the levels. For contour lines `len(allsegs) == len(levels)`, and for filled contour regions `len(allsegs) = len(levels)-1`. The lists should look like:

`level0segs = [polygon0, polygon1, ...]`
`polygon0 = array_like [[x0,y0], [x1,y1], ...]`

`allkinds` `[[level0kinds, level1kinds, ...], optional]` Optional list of all the polygon vertex kinds (code types), as described and used in `Path`. This is used to allow multiply- connected paths such as holes within filled polygons. If not `None`, `len(allkinds) == len(allsegs)`. The lists should look like:

`level0kinds = [polygon0kinds, ...]`
`polygon0kinds = [vertexcode0, vertexcode1, ...]`

If `allkinds` is not `None`, usually all polygons for a particular contour level are grouped together so that `level0segs = [polygon0]` and `level0kinds = [polygon0kinds]`.

`**kwargs` Keyword arguments are as described in the docstring of `contour`.

44.1 matplotlib.container

`class matplotlib.container.BarContainer(patches, errorbar=None, **kwargs)`

Bases: `matplotlib.container.Container`

Container for the artists of bar plots (e.g. created by `Axes.bar`).

The container can be treated as a tuple of the patches themselves. Additionally, you can access these and further parameters by the attributes.

Attributes

`patches` [list of `Rectangle`] The artists of the bars.

`errorbar` [None or `ErrorbarContainer`] A container for the error bar artists if error bars are present. None otherwise.

`class matplotlib.container.Container(kl, label=None)`

Bases: `tuple`

Base class for containers.

Containers are classes that collect semantically related Artists such as the bars of a bar plot.

`add_callback(func)`

Adds a callback function that will be called whenever one of the Artist's properties changes.

Returns an id that is useful for removing the callback with `remove_callback()` later.

`get_children()`

`get_label()`

Get the label used for this artist in the legend.

`pchanged()`

Fire an event when property changed, calling all of the registered callbacks.

`remove()`

`remove_callback(oid)`

Remove a callback based on its id.

See also:

`add_callback()` For adding callbacks

`set_label(s)`

Set the label to `s` for auto legend.

ACCEPTS: string or anything printable with '%s' conversion.

`set_remove_method(f)`

`class matplotlib.container.ErrorbarContainer(lines, has_xerr=False, has_yerr=False, **kwargs)`

Bases: `matplotlib.container.Container`

Container for the artists of error bars (e.g. created by `Axes.errorbar`).

The container can be treated as the lines tuple itself. Additionally, you can access these and further parameters by the attributes.

Attributes

`lines` [tuple] Tuple of (`data_line`, `caplines`, `barlinecols`).

- `data_line` : `Line2D` instance of x, y plot markers and/or line.
- `caplines` : tuple of `Line2D` instances of the error bar caps.
- `barlinecols` : list of `LineCollection` with the horizontal and vertical error ranges.

`has_xerr`, `has_yerr` [bool] True if the errorbar has x/y errors.

`class matplotlib.container.StemContainer(markerline_stemlines_baseline, **kwargs)`

Bases: `matplotlib.container.Container`

Container for the artists created in a `Axes.stem()` plot.

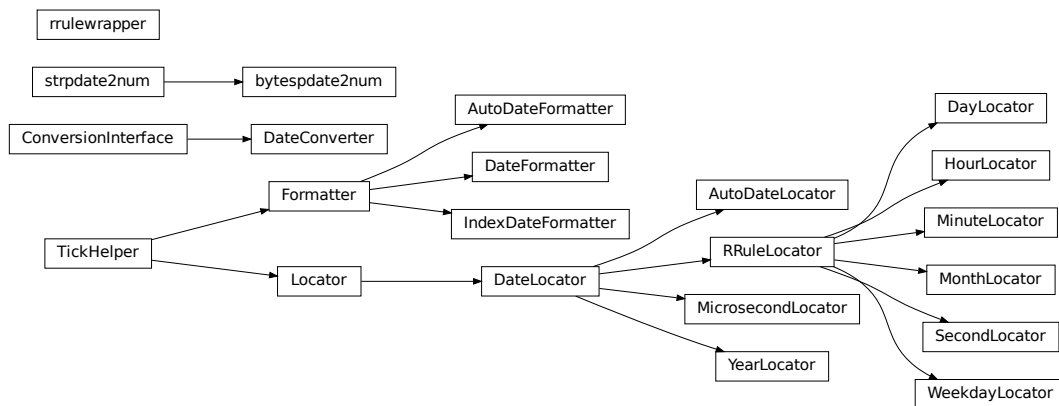
The container can be treated like a `namedtuple` (`markerline`, `stemlines`, `baseline`).

Attributes

`markerline` [`Line2D`] The artist of the markers at the stem heads.

`stemlines` [list of `Line2D`] The artists of the vertical lines for all stems.

`baseline` [`Line2D`] The artist of the horizontal baseline.



45.1 matplotlib.dates

Matplotlib provides sophisticated date plotting capabilities, standing on the shoulders of python `datetime`, the add-on modules `pytz` and `dateutil`.

45.1.1 Matplotlib date format

Matplotlib represents dates using floating point numbers specifying the number of days since 0001-01-01 UTC, plus 1. For example, 0001-01-01, 06:00 is 1.25, not 0.25. Values < 1, i.e. dates before 0001-01-01 UTC are not supported.

There are a number of helper functions to convert between `datetime` objects and Matplotlib dates:

<code>date2num</code>	Convert <code>datetime</code> objects to Matplotlib dates.
<code>num2date</code>	Convert Matplotlib dates to <code>datetime</code> objects.

Continued on next page

Table 1 – continued from previous page

<code>num2timedelta</code>	Convert number of days to a <code>timedelta</code> object.
<code>epoch2num</code>	Convert an epoch or sequence of epochs to the new date format, that is days since 0001.
<code>num2epoch</code>	Convert days since 0001 to epoch.
<code>mx2num</code>	Convert <code>mx</code> datetime instance (or sequence of <code>mx</code> instances) to the new date format.
<code>drange</code>	Return a sequence of equally spaced Matplotlib dates.

Note: Like Python's `datetime`, `mpl` uses the Gregorian calendar for all conversions between dates and floating point numbers. This practice is not universal, and calendar differences can cause confusing differences between what Python and `mpl` give as the number of days since 0001-01-01 and what other software and databases yield. For example, the US Naval Observatory uses a calendar that switches from Julian to Gregorian in October, 1582. Hence, using their calculator, the number of days between 0001-01-01 and 2006-04-01 is 732403, whereas using the Gregorian calendar via the `datetime` module we find:

```
In [1]: date(2006, 4, 1).toordinal() - date(1, 1, 1).toordinal()
Out[1]: 732401
```

All the Matplotlib date converters, tickers and formatters are timezone aware. If no explicit timezone is provided, the `rcParam` timezone is assumed. If you want to use a custom time zone, pass a `pytz.timezone` instance with the `tz` keyword argument to `num2date()`, `plot_date()`, and any custom date tickers or locators you create. See `pytz` for information on `pytz` and timezone handling.

A wide range of specific and general purpose date tick locators and formatters are provided in this module. See `matplotlib.ticker` for general information on tick locators and formatters. These are described below.

The `dateutil` module provides additional code to handle date ticking, making it easy to place ticks on any kinds of dates. See examples below.

45.1.2 Date tickers

Most of the date tickers can locate single or multiple values. For example:

```
# import constants for the days of the week
from matplotlib.dates import MO, TU, WE, TH, FR, SA, SU

# tick on Mondays every week
loc = WeekdayLocator(byweekday=MO, tz=tz)

# tick on Mondays and Saturdays
loc = WeekdayLocator(byweekday=(MO, SA))
```

In addition, most of the constructors take an interval argument:

```
# tick on Mondays every second week
loc = WeekdayLocator(byweekday=MO, interval=2)
```

The `rrule` locator allows completely general date ticking:

```
# tick every 5th easter
rule = rrulewrapper(YEARLY, byeaster=1, interval=5)
loc = RRuleLocator(rule)
```

Here are all the date tickers:

- `MicrosecondLocator`: locate microseconds
- `SecondLocator`: locate seconds
- `MinuteLocator`: locate minutes
- `HourLocator`: locate hours
- `DayLocator`: locate specified days of the month
- `WeekdayLocator`: Locate days of the week, e.g., `MO`, `TU`
- `MonthLocator`: locate months, e.g., 7 for July
- `YearLocator`: locate years that are multiples of base
- `RRuleLocator`: locate using a `matplotlib.dates.rrulewrapper`. The `rrulewrapper` is a simple wrapper around a `dateutil.rrule` ([dateutil](#)) which allow almost arbitrary date tick specifications. See [rrule example](#).
- `AutoDateLocator`: On autoscale, this class picks the best `DateLocator` (e.g., `RRuleLocator`) to set the view limits and the tick locations. If called with `interval_multiples=True` it will make ticks line up with sensible multiples of the tick intervals. E.g. if the interval is 4 hours, it will pick hours 0, 4, 8, etc as ticks. This behaviour is not guaranteed by default.

45.1.3 Date formatters

Here are all the date formatters:

- `AutoDateFormatter`: attempts to figure out the best format to use. This is most useful when used with the `AutoDateLocator`.
- `DateFormatter`: use `strftime()` format strings
- `IndexDateFormatter`: date plots with implicit x indexing.

`matplotlib.dates.date2num(d)`

Convert datetime objects to Matplotlib dates.

Parameters

`d` [[datetime.datetime](#) or `numpy.datetime64` or sequences of these]

Returns

float or sequence of floats Number of days (fraction part represents hours, minutes, seconds, ms) since 0001-01-01 00:00:00 UTC, plus one.

Notes

The addition of one here is a historical artifact. Also, note that the Gregorian calendar is assumed; this is not universal practice. For details see the module docstring.

`matplotlib.dates.num2date(x, tz=None)`

Convert Matplotlib dates to `datetime` objects.

Parameters

`x` [float or sequence of floats] Number of days (fraction part represents hours, minutes, seconds) since 0001-01-01 00:00:00 UTC, plus one.

`tz` [string, optional] Timezone of `x` (defaults to `rcparams` timezone).

Returns

`'~datetime.datetime'` or sequence of `'~datetime.datetime'` Dates are returned in timezone `tz`.

If `x` is a sequence, a sequence of `datetime` objects will be returned.

Notes

The addition of one here is a historical artifact. Also, note that the Gregorian calendar is assumed; this is not universal practice. For details, see the module docstring.

`matplotlib.dates.num2timedelta(x)`

Convert number of days to a `timedelta` object.

If `x` is a sequence, a sequence of `timedelta` objects will be returned.

Parameters

`x` [float, sequence of floats] Number of days. The fraction part represents hours, minutes, seconds.

Returns

`'datetime.timedelta'` or `list['datetime.timedelta']`

`matplotlib.dates.drange(dstart, dend, delta)`

Return a sequence of equally spaced Matplotlib dates.

The dates start at `dstart` and reach up to, but not including `dend`. They are spaced by `delta`.

Parameters

`dstart, dend` [`datetime`] The date limits.

`delta` [`datetime.timedelta`] Spacing of the dates.

Returns

`drange` [[numpy.array](#)] A list floats representing Matplotlib dates.

`matplotlib.dates.epoch2num(e)`

Convert an epoch or sequence of epochs to the new date format, that is days since 0001.

`matplotlib.dates.num2epoch(d)`

Convert days since 0001 to epoch. `d` can be a number or sequence.

`matplotlib.dates.mx2num(mxdates)`

Convert `mx` datetime instance (or sequence of `mx` instances) to the new date format.

class `matplotlib.dates.DateFormatter(fmt, tz=None)`

Bases: `matplotlib.ticker.Formatter`

Tick location is seconds since the epoch. Use a `strftime()` format string.

Python only supports [datetime](#) `strftime()` formatting for years greater than 1900. Thanks to Andrew Dalke, Dalke Scientific Software who contributed the `strftime()` code below to include dates earlier than this year.

`fmt` is a `strftime()` format string; `tz` is the `tzinfo` instance.

```
illegal_s = re.compile('([^\[%]])(%%)*%s')
```

`set_tzinfo(tz)`

`strftime(dt, fmt=None)`

Refer to documentation for [datetime.datetime.strftime\(\)](#)

`fmt` is a [datetime.datetime.strftime\(\)](#) format string.

Warning: For years before 1900, depending upon the current locale it is possible that the year displayed with `%x` might be incorrect. For years before 100, `%y` and `%Y` will yield zero-padded strings.

`strftime_pre_1900(dt, fmt=None)`

Call `time.strftime` for years before 1900 by rolling forward a multiple of 28 years.

`fmt` is a `strftime()` format string.

Dalke: I hope I did this math right. Every 28 years the calendar repeats, except through century leap years excepting the 400 year leap years. But only if you're using the Gregorian calendar.

class `matplotlib.dates.IndexDateFormatter(t, fmt, tz=None)`

Bases: `matplotlib.ticker.Formatter`

Use with `IndexLocator` to cycle format strings by index.

`t` is a sequence of dates (floating point days). `fmt` is a `strftime()` format string.

class `matplotlib.dates.AutoDateFormatter(locator, tz=None, defaultfmt='%Y-%m-%d')`

Bases: `matplotlib.ticker.Formatter`

This class attempts to figure out the best format to use. This is most useful when used with the `AutoDateLocator`.

The `AutoDateFormatter` has a scale dictionary that maps the scale of the tick (the distance in days between one major tick) and a format string. The default looks like this:

```
self.scaled = {
    DAYS_PER_YEAR: rcParams['date.autoformat.year'],
    DAYS_PER_MONTH: rcParams['date.autoformat.month'],
    1.0: rcParams['date.autoformat.day'],
    1. / HOURS_PER_DAY: rcParams['date.autoformat.hour'],
    1. / (MINUTES_PER_DAY): rcParams['date.autoformat.minute'],
    1. / (SEC_PER_DAY): rcParams['date.autoformat.second'],
    1. / (MUSECONDS_PER_DAY): rcParams['date.autoformat.microsecond'],
}
```

The algorithm picks the key in the dictionary that is \geq the current scale and uses that format string. You can customize this dictionary by doing:

```
>>> locator = AutoDateLocator()
>>> formatter = AutoDateFormatter(locator)
>>> formatter.scaled[1/(24.*60.)] = '%M:%S' # only show min and sec
```

A custom `FuncFormatter` can also be used. The following example shows how to use a custom format function to strip trailing zeros from decimal seconds and adds the date to the first ticklabel:

```
>>> def my_format_function(x, pos=None):
...     x = matplotlib.dates.num2date(x)
...     if pos == 0:
...         fmt = '%D %H:%M:%S.%f'
...     else:
...         fmt = '%H:%M:%S.%f'
...     label = x.strftime(fmt)
...     label = label.rstrip("0")
...     label = label.rstrip(".")
...     return label
>>> from matplotlib.ticker import FuncFormatter
>>> formatter.scaled[1/(24.*60.)] = FuncFormatter(my_format_function)
```

Autoformat the date labels. The default format is the one to use if none of the values in `self.scaled` are greater than the unit returned by `locator._get_unit()`.

`class matplotlib.dates.DateLocator(tz=None)`

Bases: `matplotlib.ticker.Locator`

Determines the tick locations when plotting dates.

This class is subclassed by other Locators and is not meant to be used on its own.

`tz` is a `tzinfo` instance.

`datalim_to_dt()`

Convert axis data interval to datetime objects.

`hms0d = {'byhour': 0, 'byminute': 0, 'bysecond': 0}`

`nonsingular(vmin, vmax)`

Given the proposed upper and lower extent, adjust the range if it is too close to being singular (i.e. a range of ~ 0).

`set_tzinfo(tz)`

Set time zone info.

`viewlim_to_dt()`

Converts the view interval to datetime objects.

`class matplotlib.dates.RRuleLocator(o, tz=None)`

Bases: `matplotlib.dates.DateLocator`

`autoscale()`

Set the view limits to include the data range.

`static get_unit_generic(freq)`

`tick_values(vmin, vmax)`

Return the values of the located ticks given `vmin` and `vmax`.

Note: To get tick locations with the `vmin` and `vmax` values defined automatically for the associated axis simply call the Locator instance:

```
>>> print((type(loc)))
<type 'Locator'>
>>> print((loc()))
[1, 2, 3, 4]
```

`class matplotlib.dates.AutoDateLocator(tz=None, minticks=5, maxticks=None, interval_multiples=False)`

Bases: `matplotlib.dates.DateLocator`

On autoscale, this class picks the best `DateLocator` to set the view limits and the tick locations.

`minticks` is the minimum number of ticks desired, which is used to select the type of ticking (yearly, monthly, etc.).

`maxticks` is the maximum number of ticks desired, which controls any interval between ticks (ticking every other, every 3, etc.). For really fine-grained control, this can be a dictionary mapping individual rrule frequency constants (YEARLY, MONTHLY, etc.) to their own maximum number of ticks. This can be used to keep the number of ticks appropriate to the format chosen in `AutoDateFormatter`. Any frequency not specified in this dictionary is given a default value.

`tz` is a `tzinfo` instance.

`interval_multiples` is a boolean that indicates whether ticks should be chosen to be multiple of the interval. This will lock ticks to 'nicer' locations. For example, this will force the ticks to be at hours 0,6,12,18 when hourly ticking is done at 6 hour intervals.

The `AutoDateLocator` has an interval dictionary that maps the frequency of the tick (a constant from `dateutil.rrule`) and a multiple allowed for that ticking. The default looks like this:

```
self.intervald = {
    YEARLY : [1, 2, 4, 5, 10, 20, 40, 50, 100, 200, 400, 500,
              1000, 2000, 4000, 5000, 10000],
    MONTHLY : [1, 2, 3, 4, 6],
    DAILY : [1, 2, 3, 7, 14],
    HOURLY : [1, 2, 3, 4, 6, 12],
    MINUTELY: [1, 5, 10, 15, 30],
    SECONDLY: [1, 5, 10, 15, 30],
    MICROSECONDLY: [1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000,
                    5000, 10000, 20000, 50000, 100000, 200000, 500000,
                    1000000],
}
```

The interval is used to specify multiples that are appropriate for the frequency of ticking. For instance, every 7 days is sensible for daily ticks, but for minutes/seconds, 15 or 30 make sense. You can customize this dictionary by doing:

```
locator = AutoDateLocator()
locator.intervald[HOURLY] = [3] # only show every 3 hours
```

`autoscale()`

Try to choose the view limits intelligently.

`get_locator(dmin, dmax)`

Pick the best locator based on a distance.

`nonsingular(vmin, vmax)`

Given the proposed upper and lower extent, adjust the range if it is too close to being singular (i.e. a range of ~ 0).

`refresh()`

Refresh internal information based on current limits.

`set_axis(axis)`

`tick_values(vmin, vmax)`

Return the values of the located ticks given `vmin` and `vmax`.

Note: To get tick locations with the `vmin` and `vmax` values defined automatically for the associated axis simply call the `Locator` instance:

```
>>> print((type(loc)))
<type 'Locator'>
>>> print((loc()))
[1, 2, 3, 4]
```

```
class matplotlib.dates.YearLocator(base=1, month=1, day=1, tz=None)
```

Bases: matplotlib.dates.DateLocator

Make ticks on a given day of each year that is a multiple of base.

Examples:

```
# Tick every year on Jan 1st
locator = YearLocator()

# Tick every 5 years on July 4th
locator = YearLocator(5, month=7, day=4)
```

Mark years that are multiple of base on a given month and day (default jan 1).

`autoscale()`

Set the view limits to include the data range.

`tick_values(vmin, vmax)`

Return the values of the located ticks given vmin and vmax.

Note: To get tick locations with the vmin and vmax values defined automatically for the associated axis simply call the Locator instance:

```
>>> print((type(loc)))
<type 'Locator'>
>>> print((loc()))
[1, 2, 3, 4]
```

```
class matplotlib.dates.MonthLocator(bymonth=None, bymonthday=1, interval=1, tz=None)
```

Bases: matplotlib.dates.RRuleLocator

Make ticks on occurrences of each month, e.g., 1, 3, 12.

Mark every month in bymonth; bymonth can be an int or sequence. Default is range(1,13), i.e. every month.

interval is the interval between each iteration. For example, if interval=2, mark every second occurrence.

```
class matplotlib.dates.WeekdayLocator(byweekday=1, interval=1, tz=None)
```

Bases: matplotlib.dates.RRuleLocator

Make ticks on occurrences of each weekday.

Mark every weekday in byweekday; byweekday can be a number or sequence.

Elements of byweekday must be one of MO, TU, WE, TH, FR, SA, SU, the constants from `dateutil.rrule`, which have been imported into the `matplotlib.dates` namespace.

interval specifies the number of weeks to skip. For example, `interval=2` plots every second week.

```
class matplotlib.dates.DayLocator(bymonthday=None, interval=1, tz=None)
```

Bases: `matplotlib.dates.RRuleLocator`

Make ticks on occurrences of each day of the month. For example, 1, 15, 30.

Mark every day in bymonthday; bymonthday can be an int or sequence.

Default is to tick every day of the month: `bymonthday=range(1,32)`

```
class matplotlib.dates.HourLocator(byhour=None, interval=1, tz=None)
```

Bases: `matplotlib.dates.RRuleLocator`

Make ticks on occurrences of each hour.

Mark every hour in byhour; byhour can be an int or sequence. Default is to tick every hour: `byhour=range(24)`

interval is the interval between each iteration. For example, if `interval=2`, mark every second occurrence.

```
class matplotlib.dates.MinuteLocator(byminute=None, interval=1, tz=None)
```

Bases: `matplotlib.dates.RRuleLocator`

Make ticks on occurrences of each minute.

Mark every minute in byminute; byminute can be an int or sequence. Default is to tick every minute: `byminute=range(60)`

interval is the interval between each iteration. For example, if `interval=2`, mark every second occurrence.

```
class matplotlib.dates.SecondLocator(bysecond=None, interval=1, tz=None)
```

Bases: `matplotlib.dates.RRuleLocator`

Make ticks on occurrences of each second.

Mark every second in bysecond; bysecond can be an int or sequence. Default is to tick every second: `bysecond = range(60)`

interval is the interval between each iteration. For example, if `interval=2`, mark every second occurrence.

```
class matplotlib.dates.MicrosecondLocator(interval=1, tz=None)
```

Bases: `matplotlib.dates.DateLocator`

Make ticks on regular intervals of one or more microsecond(s).

Note: Due to the floating point representation of time in days since 0001-01-01 UTC (plus 1), plotting data with microsecond time resolution does not work well with current dates.

If you want microsecond resolution time plots, it is strongly recommended to use floating point seconds, not datetime-like time representation.

If you really must use `datetime.datetime()` or similar and still need microsecond precision, your only chance is to use very early years; using year 0001 is recommended.

`interval` is the interval between each iteration. For example, if `interval=2`, mark every second microsecond.

`set_axis(axis)`

`set_data_interval(vmin, vmax)`

`set_view_interval(vmin, vmax)`

`tick_values(vmin, vmax)`

Return the values of the located ticks given `vmin` and `vmax`.

Note: To get tick locations with the `vmin` and `vmax` values defined automatically for the associated axis simply call the Locator instance:

```
>>> print((type(loc)))
<type 'Locator'>
>>> print((loc()))
[1, 2, 3, 4]
```

```
class matplotlib.dates.rrule(freq, dtstart=None, interval=1, wkst=None, count=None, until=None,
                             bysetpos=None, bymonth=None, bymonthday=None,
                             byyearday=None, byeaster=None, byweekno=None, byweekday=None,
                             byhour=None, byminute=None, bysecond=None,
                             cache=False)
```

Bases: `dateutil.rrule.rrulebase`

That's the base of the `rrule` operation. It accepts all the keywords defined in the RFC as its constructor parameters (except `byday`, which was renamed to `byweekday`) and more. The constructor prototype is:

```
rrule(freq)
```

Where `freq` must be one of `YEARLY`, `MONTHLY`, `WEEKLY`, `DAILY`, `HOURLY`, `MINUTELY`, or `SECONDLY`.

Note: Per RFC section 3.3.10, recurrence instances falling on invalid dates and times are ignored rather than coerced:

Recurrence rules may generate recurrence instances with an invalid date (e.g., February 30) or nonexistent local time (e.g., 1:30 AM on a day where the local time is moved forward by an hour at 1:00 AM). Such recurrence instances **MUST** be ignored and **MUST NOT** be counted as part of the recurrence set.

This can lead to possibly surprising behavior when, for example, the start date occurs at the end of the month:

```
>>> from dateutil.rrule import rrule, MONTHLY
>>> from datetime import datetime
>>> start_date = datetime(2014, 12, 31)
>>> list(rrule(freq=MONTHLY, count=4, dtstart=start_date))
... # doctest: +NORMALIZE_WHITESPACE
[datetime.datetime(2014, 12, 31, 0, 0),
 datetime.datetime(2015, 1, 31, 0, 0),
 datetime.datetime(2015, 3, 31, 0, 0),
 datetime.datetime(2015, 5, 31, 0, 0)]
```

Additionally, it supports the following keyword arguments:

Parameters

- `dtstart` – The recurrence start. Besides being the base for the recurrence, missing parameters in the final recurrence instances will also be extracted from this date. If not given, `datetime.now()` will be used instead.
- `interval` – The interval between each `freq` iteration. For example, when using `YEARLY`, an interval of 2 means once every two years, but with `HOURLY`, it means once every two hours. The default interval is 1.
- `wkst` – The week start day. Must be one of the `MO`, `TU`, `WE` constants, or an integer, specifying the first day of the week. This will affect recurrences based on weekly periods. The default week start is got from `calendar.firstweekday()`, and may be modified by `calendar.setfirstweekday()`.
- `count` – If given, this determines how many occurrences will be generated.

Note: As of version 2.5.0, the use of the keyword `until` in conjunction with `count` is deprecated, to make sure `dateutil` is fully compliant with [RFC-5545 Sec. 3.3.10](#). Therefore, `until` and `count` must not occur in the same call to `rrule`.

- `until` – If given, this must be a `datetime` instance specifying the upper-bound limit of the recurrence. The last recurrence in the rule is the greatest `datetime` that is less than or equal to the value specified in the `until` parameter.

Note: As of version 2.5.0, the use of the keyword `until` in conjunction with

count is deprecated, to make sure `dateutil` is fully compliant with [RFC-5545 Sec. 3.3.10](#). Therefore, `until` and `count` must not occur in the same call to `rrule`.

- `bysetpos` – If given, it must be either an integer, or a sequence of integers, positive or negative. Each given integer will specify an occurrence number, corresponding to the *n*th occurrence of the rule inside the frequency period. For example, a `bysetpos` of -1 if combined with a `MONTHLY` frequency, and a `byweekday` of (MO, TU, WE, TH, FR), will result in the last work day of every month.
- `bymonth` – If given, it must be either an integer, or a sequence of integers, meaning the months to apply the recurrence to.
- `bymonthday` – If given, it must be either an integer, or a sequence of integers, meaning the month days to apply the recurrence to.
- `byyearday` – If given, it must be either an integer, or a sequence of integers, meaning the year days to apply the recurrence to.
- `byeaster` – If given, it must be either an integer, or a sequence of integers, positive or negative. Each integer will define an offset from the Easter Sunday. Passing the offset 0 to `byeaster` will yield the Easter Sunday itself. This is an extension to the RFC specification.
- `byweekno` – If given, it must be either an integer, or a sequence of integers, meaning the week numbers to apply the recurrence to. Week numbers have the meaning described in ISO8601, that is, the first week of the year is that containing at least four days of the new year.
- `byweekday` – If given, it must be either an integer (0 == MO), a sequence of integers, one of the weekday constants (MO, TU, etc), or a sequence of these constants. When given, these variables will define the weekdays where the recurrence will be applied. It's also possible to use an argument *n* for the weekday instances, which will mean the *n*th occurrence of this weekday in the period. For example, with `MONTHLY`, or with `YEARLY` and `BYMONTH`, using `FR(+1)` in `byweekday` will specify the first friday of the month where the recurrence happens. Notice that in the RFC documentation, this is specified as `BYDAY`, but was renamed to avoid the ambiguity of that keyword.
- `byhour` – If given, it must be either an integer, or a sequence of integers, meaning the hours to apply the recurrence to.
- `byminute` – If given, it must be either an integer, or a sequence of integers, meaning the minutes to apply the recurrence to.
- `bysecond` – If given, it must be either an integer, or a sequence of integers, meaning the seconds to apply the recurrence to.
- `cache` – If given, it must be a boolean value specifying to enable or disable caching of results. If you will use the same `rrule` instance multiple times,

enabling caching will improve the performance considerably.

`replace(**kwargs)`

Return new `rrule` with same attributes except for those attributes given new values by whichever keyword arguments are specified.

```
class matplotlib.dates.relativedelta(dt1=None, dt2=None, years=0, months=0, days=0,
                                     leapdays=0, weeks=0, hours=0, minutes=0, seconds=0,
                                     microseconds=0, year=None, month=None, day=None,
                                     weekday=None, yearday=None, nlyearday=None,
                                     hour=None, minute=None, second=None, microsecond=None)
```

Bases: `object`

The `relativedelta` type is designed to be applied to an existing `datetime` and can replace specific components of that `datetime`, or represents an interval of time.

It is based on the specification of the excellent work done by M.-A. Lemburg in his `mx.DateTime` extension. However, notice that this type does NOT implement the same algorithm as his work. Do NOT expect it to behave like `mx.DateTime`'s counterpart.

There are two different ways to build a `relativedelta` instance. The first one is passing it two `date/datetime` classes:

```
relativedelta(datetime1, datetime2)
```

The second one is passing it any number of the following keyword arguments:

```
relativedelta(arg1=x, arg2=y, arg3=z...)
```

`year, month, day, hour, minute, second, microsecond`:

Absolute information (argument `is` singular); adding `or` subtracting a `relativedelta` `with` absolute information does `not` perform an arithmetic operation, but rather REPLACES the corresponding value `in` the original `datetime` `with` the value(s) `in` `relativedelta`.

`years, months, weeks, days, hours, minutes, seconds, microseconds`:

Relative information, may be negative (argument `is` plural); adding `or` subtracting a `relativedelta` `with` relative information performs the corresponding arithmetic operation on the original `datetime` value `with` the information `in` the `relativedelta`.

`weekday`:

One of the `weekday` instances (`MO`, `TU`, etc) available `in` the `relativedelta` module. These instances may receive a parameter `N`, specifying the `N`th `weekday`, which could be positive `or` negative (like `MO(+1)` `or` `MO(-2)`). Not specifying it `is` the same `as` specifying `+1`. You can also use an integer, where `0=MO`. This argument `is` always relative e.g. `if` the calculated date `is` already `Monday`, using `MO(1)` `or` `MO(-1)` won't change the day. To effectively make it absolute, use

(continues on next page)

(continued from previous page)

it in combination with the day argument (e.g. day=1, MO(1) for first Monday of the month).

leapdays:

Will add given days to the date found, if year is a leap year, and the date found is post 28 of february.

yearday, nlyearday:

Set the yearday or the non-leap year day (jump leap days). These are converted to day/month/leapdays information.

There are relative and absolute forms of the keyword arguments. The plural is relative, and the singular is absolute. For each argument in the order below, the absolute form is applied first (by setting each attribute to that value) and then the relative form (by adding the value to the attribute).

The order of attributes considered when this relativedelta is added to a datetime is:

1. Year
2. Month
3. Day
4. Hours
5. Minutes
6. Seconds
7. Microseconds

Finally, weekday is applied, using the rule described above.

For example

```
>>> from datetime import datetime
>>> from dateutil.relativedelta import relativedelta, MO
>>> dt = datetime(2018, 4, 9, 13, 37, 0)
>>> delta = relativedelta(hours=25, day=1, weekday=MO(1))
>>> dt + delta
datetime.datetime(2018, 4, 2, 14, 37)
```

First, the day is set to 1 (the first of the month), then 25 hours are added, to get to the 2nd day and 14th hour, finally the weekday is applied, but since the 2nd is already a Monday there is no effect.

normalized()

Return a version of this object represented entirely using integer values for the relative attributes.

```
>>> relativedelta(days=1.5, hours=2).normalized()  
relativedelta(days=+1, hours=+14)
```

Returns Returns a `dateutil.relativedelta.relativedelta` object.

weeks

`matplotlib.dates.seconds(s)`
Return seconds as days.

`matplotlib.dates.minutes(m)`
Return minutes as days.

`matplotlib.dates.hours(h)`
Return hours as days.

`matplotlib.dates.weeks(w)`
Return weeks as days.

46.1 matplotlib.dviread

A module for reading dvi files output by TeX. Several limitations make this not (currently) useful as a general-purpose dvi preprocessor, but it is currently used by the pdf backend for processing usetex text.

Interface:

```
with Dvi(filename, 72) as dvi:
    # iterate over pages:
    for page in dvi:
        w, h, d = page.width, page.height, page.descent
        for x,y,font,glyph,width in page.text:
            fontname = font.texname
            pointsize = font.size
            ...
        for x,y,height,width in page.bboxes:
            ...
```

class matplotlib.dviread.Dvi(filename, dpi)

Bases: `object`

A reader for a dvi ("device-independent") file, as produced by TeX. The current implementation can only iterate through pages in order, and does not even attempt to verify the postamble.

This class can be used as a context manager to close the underlying file upon exit. Pages can be read via iteration. Here is an overly simple way to extract text without trying to detect whitespace:

```
>>> with matplotlib.dviread.Dvi('input.dvi', 72) as dvi:
>>>     for page in dvi:
>>>         print(''.join(unichr(t.glyph) for t in page.text))
```

Read the data from the file named filename and convert TeX's internal units to units of dpi per inch. dpi only sets the units and does not limit the resolution. Use None to return TeX's internal units.

`close()`

Close the underlying file if it is open.

`class matplotlib.dviread.DviFont(scale, tfm, texname, vf)`

Bases: `object`

Encapsulation of a font that a DVI file can refer to.

This class holds a font's texname and size, supports comparison, and knows the widths of glyphs in the same units as the AFM file. There are also internal attributes (for use by `dviread.py`) that are not used for comparison.

The size is in Adobe points (converted from TeX points).

Parameters

`scale` [float] Factor by which the font is scaled from its natural size.

`tfm` [Tfm] TeX font metrics for this font

`texname` [bytes] Name of the font as used internally by TeX and friends, as an ASCII bytestring. This is usually very different from any external font names, and `dviread.PsfontsMap` can be used to find the external name of the font.

`vf` [Vf] A TeX "virtual font" file, or None if this font is not virtual.

Attributes

`texname` [bytes]

`size` [float] Size of the font in Adobe points, converted from the slightly smaller TeX points.

`widths` [list] Widths of glyphs in glyph-space units, typically 1/1000ths of the point size.

`size`

`texname`

`widths`

`class matplotlib.dviread.Encoding(filename)`

Bases: `object`

Parses a *.enc file referenced from a psfonts.map style file. The format this class understands is a very limited subset of PostScript.

Usage (subject to change):

```
for name in Encoding(filename):
    whatever(name)
```

Parameters

filename [string or bytestring]

Attributes

encoding [list] List of character names

encoding

matplotlib.dviread.PsFont

alias of matplotlib.dviread.Font

class matplotlib.dviread.PsfontsMap(filename)

Bases: `object`

A psfonts.map formatted file, mapping TeX fonts to PS fonts.

Usage:

```
>>> map = PsfontsMap(find_tex_file('pdfTeX.map'))
>>> entry = map[b'ptmbo8r']
>>> entry.texname
b'ptmbo8r'
>>> entry.psname
b'Times-Bold'
>>> entry.encoding
'/usr/local/texlive/2008/texmf-dist/fonts/enc/dvips/base/8r.enc'
>>> entry.effects
{'slant': 0.16700000000000001}
>>> entry.filename
```

Parameters

filename [string or bytestring]

Notes

For historical reasons, TeX knows many Type-1 fonts by different names than the outside world. (For one thing, the names have to fit in eight characters.) Also, TeX's native fonts are not Type-1 but Metafont, which is nontrivial to convert to PostScript except as a bitmap. While high-quality conversions to Type-1 format exist and are shipped with modern TeX distributions, we need to know which Type-1 fonts are the counterparts of which native fonts. For these reasons a mapping is needed from internal font names to font file names.

A texmf tree typically includes mapping files called e.g. psfonts.map, pdfTeX.map, or dvipdfm.map. The file psfonts.map is used by dvips, pdfTeX.map by pdfTeX, and dvipdfm.map by dvipdfm. psfonts.map might avoid embedding the 35 PostScript fonts (i.e., have no filename for them, as in the Times-Bold example above), while the pdf-related files perhaps only avoid the "Base 14" pdf fonts. But the user may have configured these files differently.

class matplotlib.dviread.Tfm(filename)

Bases: `object`

A TeX Font Metric file.

This implementation covers only the bare minimum needed by the Dvi class.

Parameters

filename [string or bytestring]

Attributes

checksum [int] Used for verifying against the dvi file.

design_size [int] Design size of the font (unknown units)

width, height, depth [dict] Dimensions of each character, need to be scaled by the factor specified in the dvi file. These are dicts because indexing may not start from 0.

checksum

depth

design_size

height

width

class matplotlib.dviread.Vf(filename)

Bases: `matplotlib.dviread.Dvi`

A virtual font (*.vf file) containing subroutines for dvi files.

Usage:

```
vf = Vf(filename)
glyph = vf[code]
glyph.text, glyph.bboxes, glyph.width
```

Parameters

filename [string or bytestring]

Notes

The virtual font format is a derivative of dvi: <http://mirrors.ctan.org/info/knuth/virtual-fonts> This class reuses some of the machinery of Dvi but replaces the `__read` loop

and dispatch mechanism.

`matplotlib.dviread.find_tex_file(filename, format=None)`

Find a file in the texmf tree.

Calls `kpsewhich` which is an interface to the `kpathsea` library [?]. Most existing TeX distributions on Unix-like systems use `kpathsea`. It is also available as part of MikTeX, a popular distribution on Windows.

Parameters

`filename` [string or bytestring]

`format` [string or bytestring] Used as the value of the `--format` option to `kpsewhich`. Could be e.g. `'tfm'` or `'vf'` to limit the search to that type of files.

References

[?]

`matplotlib.dviread.ord(x)`

FIGURE

47.1 matplotlib.figure

The figure module provides the top-level Artist, the Figure, which contains all the plot elements. The following classes are defined

SubplotParams control the default spacing of the subplots

Figure Top level container for all plot elements.

47.1.1 Classes

AxesStack()	Specialization of the Stack to handle all tracking of Axes in a Figure.
Figure([figsize, dpi, facecolor, edgecolor, ...])	The top level container for all the plot elements.
SubplotParams([left, bottom, right, top, ...])	A class to hold the parameters for a subplot.

matplotlib.figure.AxesStack

class matplotlib.figure.AxesStack

Specialization of the Stack to handle all tracking of Axes in a Figure. This stack stores key, (ind, axes) pairs, where:

- key should be a hash of the args and kwargs used in generating the Axes.
- ind is a serial number for tracking the order in which axes were added.

The AxesStack is a callable, where `ax_stack()` returns the current axes. Alternatively the `current_key_axes()` will return the current key and associated axes.

`add(key, a)`

Add Axes a, with key key, to the stack, and return the stack.

If key is unhashable, replace it by a unique, arbitrary object.

If a is already on the stack, don't add it again, but return None.

`as_list()`

Return a list of the Axes instances that have been added to the figure

`bubble(a)`

Move the given axes, which must already exist in the stack, to the top.

`current_key_axes()`

Return a tuple of (key, axes) for the active axes.

If no axes exists on the stack, then returns (None, None).

`get(key)`

Return the Axes instance that was added with key. If it is not present, return None.

`remove(a)`

Remove the axes from the stack.

`matplotlib.figure.Figure`

```
class matplotlib.figure.Figure(figsize=None, dpi=None, facecolor=None, edgecolor=None,  
                               linewidth=0.0, frameon=None, subplotpars=None,  
                               tight_layout=None, constrained_layout=None)
```

The top level container for all the plot elements.

The Figure instance supports callbacks through a `callbacks` attribute which is a `CallbackRegistry` instance. The events you can connect to are `'dpi_changed'`, and the callback will be called with `func(fig)` where `fig` is the Figure instance.

Attributes

`patch` The Rectangle instance representing the figure patch.

`suppressComposite` For multiple figure images, the figure will make composite images depending on the renderer option `_image_nocomposite` function. If `suppressComposite` is a boolean, this will override the renderer.

Parameters

`figsize` [2-tuple of floats, default: `rcParams["figure.figsize"]`] Figure dimension (width, height) in inches.

`dpi` [float, default: `rcParams["figure.dpi"]`] Dots per inch.

`facecolor` [default: `rcParams["figure.facecolor"]`] The figure patch facecolor.

`edgecolor` [default: `rcParams["figure.edgecolor"]`] The figure patch edge color.

`linewidth` [float] The linewidth of the frame (i.e. the edge linewidth of the figure patch).

`frameon` [bool, default: `rcParams["figure.frameon"]`] If False, suppress drawing the figure frame.

`subplotpars` [SubplotParams] Subplot parameters. If not given, the default subplot parameters `rcParams["figure.subplot.*"]` are used.

`tight_layout` [bool or dict, default: `rcParams["figure.autolayout"]`] If False use `subplotspars`. If True adjust subplot parameters using `tight_layout` with default padding. When providing a dict containing the keys `pad`, `w_pad`, `h_pad`, and `rect`, the default `tight_layout` paddings will be overridden.

`constrained_layout` [bool] If True use constrained layout to adjust positioning of plot elements. Like `tight_layout`, but designed to be more flexible. See [Constrained Layout Guide](#) for examples. (Note: does not work with `subplot()` or `subplot2grid()`.) Defaults to `rcParams["figure.constrained_layout.use"]`.

`add_axes(*args, **kwargs)`

Add an axes to the figure.

Call signature:

```
add_axes(rect, projection=None, polar=False, **kwargs)
```

Parameters

`rect` [sequence of float] The dimensions [left, bottom, width, height] of the new axes. All quantities are in fractions of figure width and height.

`projection` [{None, 'aitoff', 'hammer', 'lambert', 'mollweide', 'polar', 'rectilinear'}, optional] The projection type of the axes.

`polar` [boolean, optional] If True, equivalent to `projection='polar'`.

`**kwargs` This method also takes the keyword arguments for Axes.

Returns

`axes` [Axes] The added axes.

Examples

Some simple examples:

```
rect = l, b, w, h
fig.add_axes(rect)
fig.add_axes(rect, frameon=False, facecolor='g')
fig.add_axes(rect, polar=True)
fig.add_axes(rect, projection='polar')
fig.add_axes(ax)
```

If the figure already has an axes with the same parameters, then it will simply make that axes current and return it. This behavior has been deprecated as of Matplotlib 2.1. Meanwhile, if you do not want this behavior (i.e., you want to force the creation of a new Axes), you must use a unique set of args and kwargs. The axes label attribute has been exposed for this purpose: if you want two axes that are otherwise identical to be added to the figure, make sure you give them unique labels:

```
fig.add_axes(rect, label='axes1')
fig.add_axes(rect, label='axes2')
```

In rare circumstances, `add_axes` may be called with a single argument, an `Axes` instance already created in the present figure but not in the figure's list of axes. For example, if an axis has been removed with `delaxes()`, it can be restored with:

```
fig.add_axes(ax)
```

In all cases, the `Axes` instance will be returned.

`add_axobserver(func)`

Whenever the axes state change, `func(self)` will be called.

`add_subplot(*args, **kwargs)`

Add a subplot.

Call signatures:

```
add_subplot(nrows, ncols, index, **kwargs)
add_subplot(pos, **kwargs)
```

Parameters

`*args` Either a 3-digit integer or three separate integers describing the position of the subplot. If the three integers are R, C, and P in order, the subplot will take the Pth position on a grid with R rows and C columns.

`projection` [{None, 'aitoff', 'hammer', 'lambert', 'mollweide', 'polar', 'rectilinear'}, optional] The projection type of the axes.

`polar` [boolean, optional] If True, equivalent to `projection='polar'`.

`**kwargs` This method also takes the keyword arguments for `Axes`.

Returns

`axes` [Axes] The axes of the subplot.

See also:

`matplotlib.pyplot.subplot` for an explanation of the args.

Notes

If the figure already has a subplot with key (args, kwargs) then it will simply make that subplot current and return it. This behavior is deprecated.

Examples

```
fig.add_subplot(111)

# equivalent but more general
fig.add_subplot(1, 1, 1)

# add subplot with red background
fig.add_subplot(212, facecolor='r')

# add a polar subplot
fig.add_subplot(111, projection='polar')

# add Subplot instance sub
fig.add_subplot(sub)
```

`align_labels(axes=None)`

Align the xlabels and ylabels of subplots with the same subplots row or column (respectively) if label alignment is being done automatically (i.e. the label position is not manually set).

Alignment persists for draw events after this is called.

Parameters

`axes` [list of Axes] Optional list (or ndarray) of Axes to align the labels.
Default is to align all axes on the figure.

See also:

`matplotlib.figure.Figure.align_xlabels`, `matplotlib.figure.Figure.align_ylabels`

`align_xlabels(axes=None)`

Align the ylabels of subplots in the same subplot column if label alignment is being done automatically (i.e. the label position is not manually set).

Alignment persists for draw events after this is called.

If a label is on the bottom, it is aligned with labels on axes that also have their label on the bottom and that have the same bottom-most subplot row. If the label is on the top, it is aligned with labels on axes with the same top-most row.

Parameters

`axes` [list of Axes] Optional list of (or ndarray) Axes to align the xlabels.
Default is to align all axes on the figure.

See also:

`matplotlib.figure.Figure.align_ylabels`, `matplotlib.figure.Figure.align_labels`

Notes

This assumes that `axs` are from the same `GridSpec`, so that their `SubplotSpec` positions correspond to figure positions.

Examples

Example with rotated xtick labels:

```
fig, axs = plt.subplots(1, 2)
for tick in axs[0].get_xticklabels():
    tick.set_rotation(55)
axs[0].set_xlabel('XLabel 0')
axs[1].set_xlabel('XLabel 1')
fig.align_xlabels()
```

`align_ylabels(axs=None)`

Align the ylabels of subplots in the same subplot column if label alignment is being done automatically (i.e. the label position is not manually set).

Alignment persists for draw events after this is called.

If a label is on the left, it is aligned with labels on axes that also have their label on the left and that have the same left-most subplot column. If the label is on the right, it is aligned with labels on axes with the same right-most column.

Parameters

`axs` [list of Axes] Optional list (or ndarray) of Axes to align the ylabels.
Default is to align all axes on the figure.

See also:

`matplotlib.figure.Figure.align_xlabels`, `matplotlib.figure.Figure.align_labels`

Notes

This assumes that `axs` are from the same `GridSpec`, so that their `SubplotSpec` positions correspond to figure positions.

Examples

Example with large yticks labels:

```
fig, axs = plt.subplots(2, 1)
axs[0].plot(np.arange(0, 1000, 50))
axs[0].set_ylabel('YLabel 0')
axs[1].set_ylabel('YLabel 1')
fig.align_ylabels()
```

`autofmt_xdate(bottom=0.2, rotation=30, ha='right', which=None)`

Date ticklabels often overlap, so it is useful to rotate them and right align them. Also, a common use case is a number of subplots with shared axes where the x-axis is date data. The ticklabels are often long, and it helps to rotate them on the bottom subplot and turn them off on other subplots, as well as turn off xlabels.

Parameters

`bottom` [scalar] The bottom of the subplots for `subplots_adjust()`.

`rotation` [angle in degrees] The rotation of the xtick labels.

`ha` [string] The horizontal alignment of the xticklabels.

`which` [{None, 'major', 'minor', 'both'}] Selects which ticklabels to rotate.
Default is None which works the same as major.

`axes`

List of axes in the Figure. You can access the axes in the Figure through this list. Do not modify the list itself. Instead, use `add_axes`, `subplot` or `delaxes` to add or remove an axes.

`clear(keep_observers=False)`

Clear the figure – synonym for `clf()`.

`clf(keep_observers=False)`

Clear the figure.

Set `keep_observers` to True if, for example, a gui widget is tracking the axes in the figure.

`colorbar(mappable, cax=None, ax=None, use_gridspec=True, **kw)`

Create a colorbar for a `ScalarMappable` instance, `mappable`.

Documentation for the pylab thin wrapper:

Add a colorbar to a plot.

Function signatures for the pyplot interface; all but the first are also method signatures for the `colorbar()` method:

```
colorbar(**kwargs)
colorbar(mappable, **kwargs)
colorbar(mappable, cax=cax, **kwargs)
colorbar(mappable, ax=ax, **kwargs)
```

Parameters

`mappable` : The Image, ContourSet, etc. to which the colorbar applies; this argument is mandatory for the Figure `colorbar()` method but optional for the pyplot `colorbar()` function, which sets the default to the current image.

`cax` [Axes object, optional] Axes into which the colorbar will be drawn.

`ax` [Axes, list of Axes, optional] Parent axes from which space for a new colorbar axes will be stolen. If a list of axes is given they will all be resized to make room for the colorbar axes.

`use_gridspec` [bool, optional] If `cax` is None, a new `cax` is created as an instance of Axes. If `ax` is an instance of Subplot and `use_gridspec` is True, `cax` is created as an instance of Subplot using the `grid_spec` module.

Returns

:class:~matplotlib.colorbar.Colorbar instance See also its base class, ColorbarBase. Call the `set_label()` method to label the colorbar.

Notes

Additional keyword arguments are of two kinds:

axes properties:

Property	Description
orientation	vertical or horizontal
fraction	0.15; fraction of original axes to use for colorbar
pad	0.05 if vertical, 0.15 if horizontal; fraction of original axes between colorbar and new image axes
shrink	1.0; fraction by which to multiply the size of the colorbar
aspect	20; ratio of long to short dimensions
anchor	(0.0, 0.5) if vertical; (0.5, 1.0) if horizontal; the anchor point of the colorbar axes
panchor	(1.0, 0.5) if vertical; (0.5, 0.0) if horizontal; the anchor point of the colorbar parent axes. If False, the parent axes' anchor will be unchanged

colorbar properties:

Property	Description
extend	['neither' 'both' 'min' 'max'] If not 'neither', make pointed end(s) for out-of-range values. These are set for a given colormap using the colormap set_under and set_over methods.
extendfrac	[None 'auto' length lengths] If set to None, both the minimum and maximum triangular colorbar extensions will have a length of 5% of the interior colorbar length (this is the default setting). If set to 'auto', makes the triangular colorbar extensions the same lengths as the interior boxes (when spacing is set to 'uniform') or the same lengths as the respective adjacent interior boxes (when spacing is set to 'proportional'). If a scalar, indicates the length of both the minimum and maximum triangular colorbar extensions as a fraction of the interior colorbar length. A two-element sequence of fractions may also be given, indicating the lengths of the minimum and maximum colorbar extensions respectively as a fraction of the interior colorbar length.
extendrect	bool If False the minimum and maximum colorbar extensions will be triangular (the default). If True the extensions will be rectangular.
spacing	['uniform' 'proportional'] Uniform spacing gives each discrete color the same space; proportional makes the space proportional to the data interval.
ticks	[None list of ticks Locator object] If None, ticks are determined automatically from the input.
format	[None format string Formatter object] If None, the ScalarFormatter is used. If a format string is given, e.g., '%.3f', that is used. An alternative Formatter object may be given instead.
drawedges	bool Whether to draw lines at color boundaries.

The following will probably be useful only in the context of indexed colors (that is, when the mappable has `norm=NoNorm()`), or other unusual circumstances.

Property	Description
boundaries	None or a sequence
values	None or a sequence which must be of length 1 less than the sequence of boundaries. For each region delimited by adjacent entries in boundaries, the color mapped to the corresponding value in values will be used.

If mappable is a `ContourSet`, its `extend` kwarg is included automatically.

The `shrink` kwarg provides a simple way to scale the colorbar with respect to the axes.

Note that if `cax` is specified it determines the size of the colorbar and `shrink` and `aspect` kwargs are ignored.

For more precise control, you can manually specify the positions of the axes objects in which the mappable and the colorbar are drawn. In this case, do not use any of the axes properties kwargs.

It is known that some vector graphics viewer (svg and pdf) renders white gaps between segments of the colorbar. This is due to bugs in the viewers not matplotlib. As a workaround the colorbar can be rendered with overlapping segments:

```
cbar = colorbar()
cbar.solids.set_edgecolor("face")
draw()
```

However this has negative consequences in other circumstances. Particularly with semi transparent images ($\alpha < 1$) and colorbar extensions and is not enabled by default see (issue #1188).

`contains(mouseevent)`

Test whether the mouse event occurred on the figure.

Returns

bool, {}

`delaxes(ax)`

Remove the Axes `ax` from the figure and update the current axes.

`dpi`

The resolution in dots per inch.

`draw(renderer)`

Render the figure using `matplotlib.backend_bases.RendererBase` instance `renderer`.

`draw_artist(a)`

Draw `matplotlib.artist.Artist` instance `a` only. This is available only after the figure is drawn.

`execute_constrained_layout(renderer=None)`

Use `layoutbox` to determine pos positions within axes.

See also `set_constrained_layout_pads`.

`figimage(X, xo=0, yo=0, alpha=None, norm=None, cmap=None, vmin=None, vmax=None, origin=None, resize=False, **kwargs)`

Add a non-resampled image to the figure.

The image is attached to the lower or upper left corner depending on `origin`.

Parameters

X The image data. This is an array of one of the following shapes:

- MxN: luminance (grayscale) values

- MxNx3: RGB values
- MxNx4: RGBA values

xo, yo [int] The x/y image offset in pixels.

alpha [None or float] The alpha blending value.

norm [matplotlib.colors.Normalize] A Normalize instance to map the luminance to the interval [0, 1].

cmap [str or matplotlib.colors.Colormap] The colormap to use. Default: rcParams["image.cmap"].

vmin, vmax [scalar] If norm is not given, these values set the data limits for the colormap.

origin [{'upper', 'lower'}] Indicates where the [0, 0] index of the array is in the upper left or lower left corner of the axes. Defaults to rcParams["image.origin"].

resize [bool] If True, resize the figure to match the given image size.

Returns

:class:'matplotlib.image.FigureImage'

Other Parameters

**kwargs Additional kwargs are Artist kwargs passed on to FigureImage.

Notes

figimage complements the axes image (imshow()) which will be resampled to fit the current axes. If you want a resampled image to fill the entire figure, you can define an Axes with extent [0,0,1,1].

Examples:

```
f = plt.figure()
nx = int(f.get_figwidth() * f.dpi)
ny = int(f.get_figheight() * f.dpi)
data = np.random.random((ny, nx))
f.figimage(data)
plt.show()
```

figurePatch

Deprecated since version 2.1: The figurePatch function was deprecated in version 2.1. Use Figure.patch instead.

gca(**kwargs)

Get the current axes, creating one if necessary.

The following kwargs are supported for ensuring the returned axes adheres to the given projection etc., and for axes creation if the active axes does not exist:

	Property	Description
adjustable	['box' 'datalim']	
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array	
alpha	float (0.0 transparent through 1.0 opaque)	
anchor	['C' 'SW' 'S' 'SE' 'E' 'NE' 'N' 'NW' 'W']	
animated	bool	
aspect	unknown	
autoscale_on	bool	
autoscalex_on	bool	
autoscaley_on	bool	
axes_locator	a callable object which takes an axes instance and renderer and returns a bbox.	
axisbelow	[bool 'line']	
clip_box	a Bbox instance	
clip_on	bool	
clip_path	[(Path, Transform) Patch None]	
contains	a callable function	
facecolor	color	
fc	color	
figure	Figure	
frame_on	bool	
gid	an id string	
label		object
navigate	bool	
navigate_mode	unknown	
path_effects	AbstractPathEffect	
picker	[None bool float callable]	
position	unknown	
rasterization_zorder	float or None	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
title	unknown	
transform	Transform	
url	a url string	
visible	bool	
xbound	(lower: float, upper: float)	
xlabel	unknown	
xlim	(left: float, right: float)	
xmargin	float greater than -0.5	
xscale	['linear' 'log' 'symlog' 'logit' ...]	
xticklabels	list of string labels	
xticks	list of tick locations.	
ybound	(lower: float, upper: float)	
ylabel	unknown	

Continued on next page

Table 2 – continued from previous page

	Property	Description
<code>ylim</code>		(bottom: float, top: float)
<code>ymargin</code>		float greater than -0.5
<code>yscale</code>		['linear' 'log' 'symlog' 'logit' ...]
<code>yticklabels</code>		list of string labels
<code>yticks</code>		list of tick locations.
<code>zorder</code>		float

`get_axes()`

Return a list of axes in the Figure. You can access and modify the axes in the Figure through this list.

Do not modify the list itself. Instead, use `add_axes`, `subplot` or `delaxes` to add or remove an axes.

Note: This is equivalent to the property `axes`.

`get_children()`

Get a list of artists contained in the figure.

`get_constrained_layout()`

Return a boolean: True means constrained layout is being used.

See [Constrained Layout Guide](#).

`get_constrained_layout_pads(relative=False)`

Get padding for constrained layout.

Returns a list of `w_pad`, `h_pad` in inches and `wspace` and `hspace` as fractions of the subplot.

See [Constrained Layout Guide](#).

Parameters

`relative` [boolean] If `True`, then convert from inches to figure relative.

`get_default_bbox_extra_artists()`

`get_dpi()`

Return the resolution in dots per inch as a float.

`get_edgecolor()`

Get the edge color of the Figure rectangle.

`get_facecolor()`

Get the face color of the Figure rectangle.

`get_figheight()`

Return the figure height as a float.

`get_figwidth()`

Return the figure width as a float.

`get_frameon()`

Return whether the figure frame will be drawn.

`get_size_inches()`

Returns the current size of the figure in inches.

Returns

size [ndarray] The size (width, height) of the figure in inches.

See also:

`matplotlib.Figure.set_size_inches`

`get_tight_layout()`

Return whether `tight_layout` is called when drawing.

`get_tightbbox(renderer)`

Return a (tight) bounding box of the figure in inches.

It only accounts axes title, axis labels, and axis ticklabels. Needs improvement.

`get_window_extent(*args, **kwargs)`

Return the figure bounding box in display space. Arguments are ignored.

`ginput(n=1, timeout=30, show_clicks=True, mouse_add=1, mouse_pop=3,
mouse_stop=2)`

Blocking call to interact with a figure.

Wait until the user clicks `n` times on the figure, and return the coordinates of each click in a list.

The buttons used for the various actions (adding points, removing points, terminating the inputs) can be overridden via the arguments `mouse_add`, `mouse_pop` and `mouse_stop`, that give the associated mouse button: 1 for left, 2 for middle, 3 for right.

Parameters

`n` [int, optional, default: 1] Number of mouse clicks to accumulate. If negative, accumulate clicks until the input is terminated manually.

`timeout` [scalar, optional, default: 30] Number of seconds to wait before timing out. If zero or negative will never timeout.

`show_clicks` [bool, optional, default: False] If True, show a red cross at the location of each click.

`mouse_add` [int, one of (1, 2, 3), optional, default: 1 (left click)] Mouse button used to add points.

`mouse_pop` [int, one of (1, 2, 3), optional, default: 3 (right click)] Mouse button used to remove the most recently added point.

`mouse_stop` [int, one of (1, 2, 3), optional, default: 2 (middle click)] Mouse button used to stop input.

Returns

`points` [list of tuples] A list of the clicked (x, y) coordinates.

Notes

The keyboard can also be used to select points in case your mouse does not have one or more of the buttons. The delete and backspace keys act like right clicking (i.e., remove last point), the enter key terminates input and any other key (not already used by the window manager) selects a point.

`hold(b=None)`

Deprecated since version 2.0: The hold function was deprecated in version 2.0.

Set the hold state. If hold is None (default), toggle the hold state. Else set the hold state to boolean value b.

e.g.:

```
hold()      # toggle hold
hold(True)  # hold is on
hold(False) # hold is off
```

All "hold" machinery is deprecated.

`init_layoutbox()`

Initialize the layoutbox for use in `constrained_layout`.

`legend(*args, **kwargs)`

Place a legend on the figure.

To make a legend from existing artists on every axes:

```
legend()
```

To make a legend for a list of lines and labels:

```
legend( (line1, line2, line3),
        ('label1', 'label2', 'label3'),
        loc='upper right')
```

These can also be specified by keyword:

```
legend(handles=(line1, line2, line3),
        labels=('label1', 'label2', 'label3'),
        loc='upper right')
```

Parameters

handles [sequence of Artist, optional] A list of Artists (lines, patches) to be added to the legend. Use this together with labels, if you need full control on what is shown in the legend and the automatic mechanism described above is not sufficient.

The length of handles and labels should be the same in this case. If they are not, they are truncated to the smaller length.

labels [sequence of strings, optional] A list of labels to show next to the artists. Use this together with handles, if you need full control on what is shown in the legend and the automatic mechanism described above is not sufficient.

Returns

:class:'matplotlib.legend.Legend' instance

Other Parameters

loc [int or string or pair of floats, default: 'upper right'] The location of the legend. Possible codes are:

Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

Alternatively can be a 2-tuple giving x, y of the lower-left corner of the legend in axes coordinates (in which case `bbox_to_anchor` will be ignored).

bbox_to_anchor [BboxBase or pair of floats] Specify any arbitrary location for the legend in `bbox_transform` coordinates (default Axes coordinates).

For example, to put the legend's upper right hand corner in the center of the axes the following keywords can be used:

`loc='upper right', bbox_to_anchor=(0.5, 0.5)`

ncol [integer] The number of columns that the legend has. Default is 1.

prop [None or matplotlib.font_manager.FontProperties or dict] The font

properties of the legend. If None (default), the current matplotlib.rcParams will be used.

fontsize [int or float or {'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'}] Controls the font size of the legend. If the value is numeric the size will be the absolute font size in points. String values are relative to the current default font size. This argument is only used if prop is not specified.

numpoints [None or int] The number of marker points in the legend when creating a legend entry for a Line2D (line). Default is None, which will take the value from rcParams["legend.numpoints"].

scatterpoints [None or int] The number of marker points in the legend when creating a legend entry for a PathCollection (scatter plot). Default is None, which will take the value from rcParams["legend.scatterpoints"].

scatteryoffsets [iterable of floats] The vertical offset (relative to the font size) for the markers created for a scatter plot legend entry. 0.0 is at the base the legend text, and 1.0 is at the top. To draw all markers at the same height, set to [0.5]. Default is [0.375, 0.5, 0.3125].

markerscale [None or int or float] The relative size of legend markers compared with the originally drawn ones. Default is None, which will take the value from rcParams["legend.markerscale"].

markerfirst [bool] If True, legend marker is placed to the left of the legend label. If False, legend marker is placed to the right of the legend label. Default is True.

frameon [None or bool] Control whether the legend should be drawn on a patch (frame). Default is None, which will take the value from rcParams["legend.frameon"].

fancybox [None or bool] Control whether round edges should be enabled around the FancyBboxPatch which makes up the legend's background. Default is None, which will take the value from rcParams["legend.fancybox"].

shadow [None or bool] Control whether to draw a shadow behind the legend. Default is None, which will take the value from rcParams["legend.shadow"].

framealpha [None or float] Control the alpha transparency of the legend's background. Default is None, which will take the value from rcParams["legend.framealpha"]. If shadow is activated and framealpha is None, the default value is ignored.

facecolor [None or "inherit" or a color spec] Control the legend's background color. Default is None, which will take the value from rcParams["legend.facecolor"]. If "inherit", it will take rcParams["axes.facecolor"].

`edgecolor` [None or "inherit" or a color spec] Control the legend's background patch edge color. Default is None, which will take the value from `rcParams["legend.edgecolor"]`. If "inherit", it will take `rcParams["axes.edgecolor"]`.

`mode` [{"expand", None}] If mode is set to "expand" the legend will be horizontally expanded to fill the axes area (or `bbox_to_anchor` if defines the legend's size).

`bbox_transform` [None or `matplotlib.transforms.Transform`] The transform for the bounding box (`bbox_to_anchor`). For a value of None (default) the Axes' `transAxes` transform will be used.

`title` [str or None] The legend's title. Default is no title (None).

`borderpad` [float or None] The fractional whitespace inside the legend border. Measured in font-size units. Default is None, which will take the value from `rcParams["legend.borderpad"]`.

`labelspacing` [float or None] The vertical space between the legend entries. Measured in font-size units. Default is None, which will take the value from `rcParams["legend.labelspacing"]`.

`handlelength` [float or None] The length of the legend handles. Measured in font-size units. Default is None, which will take the value from `rcParams["legend.handlelength"]`.

`handletextpad` [float or None] The pad between the legend handle and text. Measured in font-size units. Default is None, which will take the value from `rcParams["legend.handletextpad"]`.

`borderaxespad` [float or None] The pad between the axes and legend border. Measured in font-size units. Default is None, which will take the value from `rcParams["legend.borderaxespad"]`.

`columnspacing` [float or None] The spacing between columns. Measured in font-size units. Default is None, which will take the value from `rcParams["legend.columnspacing"]`.

`handler_map` [dict or None] The custom dictionary mapping instances or types to a legend handler. This `handler_map` updates the default handler map found at `matplotlib.legend.Legend.get_legend_handler_map()`.

Notes

Not all kinds of artist are supported by the legend command. See [Legend guide](#) for details.

`savefig(fname, **kwargs)`
Save the current figure.

Call signature:

```
savefig(fname, dpi=None, facecolor='w', edgecolor='w',
        orientation='portrait', papertype=None, format=None,
        transparent=False, bbox_inches=None, pad_inches=0.1,
        frameon=None)
```

The output formats available depend on the backend being used.

Parameters

fname [str or file-like object] A string containing a path to a filename, or a Python file-like object, or possibly some backend-dependent object such as PdfPages.

If format is None and fname is a string, the output format is deduced from the extension of the filename. If the filename has no extension, the value of the rc parameter `savefig.format` is used.

If fname is not a string, remember to specify format to ensure that the correct backend is used.

Other Parameters

dpi [[None | scalar > 0 | 'figure']] The resolution in dots per inch. If None it will default to the value `savefig.dpi` in the `matplotlibrc` file. If 'figure' it will set the dpi to be the value of the figure.

facecolor [color spec or None, optional] the facecolor of the figure; if None, defaults to `savefig.facecolor`

edgecolor [color spec or None, optional] the edgecolor of the figure; if None, defaults to `savefig.edgecolor`

orientation [{'landscape', 'portrait'}] not supported on all backends; currently only on postscript output

papertype [str] One of 'letter', 'legal', 'executive', 'ledger', 'a0' through 'a10', 'b0' through 'b10'. Only supported for postscript output.

format [str] One of the file extensions supported by the active backend. Most backends support png, pdf, ps, eps and svg.

transparent [bool] If True, the axes patches will all be transparent; the figure patch will also be transparent unless facecolor and/or edgecolor are specified via kwargs. This is useful, for example, for displaying a plot on top of a colored background on a web page. The transparency of these patches will be restored to their original values upon exit of this function.

frameon [bool] If True, the figure patch will be colored, if False, the figure background will be transparent. If not provided, the rcParam 'savefig.frameon' will be used.

bbox_inches [str or Bbox, optional] Bbox in inches. Only the given portion of the figure is saved. If 'tight', try to figure out the tight bbox of the figure. If None, use `savefig.bbox`

`pad_inches` [scalar, optional] Amount of padding around the figure when `bbox_inches` is 'tight'. If None, use `savefig.pad_inches`

`bbox_extra_artists` [list of Artist, optional] A list of extra artists that will be considered when the tight bbox is calculated.

`sca(a)`

Set the current axes to be `a` and return `a`.

`set_canvas(canvas)`

Set the canvas that contains the figure

ACCEPTS: a `FigureCanvas` instance

`set_constrained_layout(constrained)`

Set whether `constrained_layout` is used upon drawing. If None, the `rcParams['figure.constrained_layout.use']` value will be used.

When providing a dict containing the keys `w_pad`, `h_pad` the default `constrained_layout` paddings will be overridden. These pads are in inches and default to 3.0/72.0. `w_pad` is the width padding and `h_pad` is the height padding.

ACCEPTS: [True | False | dict | None]

See [Constrained Layout Guide](#).

`set_constrained_layout_pads(**kwargs)`

Set padding for `constrained_layout`. Note the kwargs can be passed as a dictionary `fig.set_constrained_layout(**paddict)`.

See [Constrained Layout Guide](#).

Parameters

`w_pad` [scalar] Width padding in inches. This is the pad around axes and is meant to make sure there is enough room for fonts to look good. Defaults to 3 pts = 0.04167 inches

`h_pad` [scalar] Height padding in inches. Defaults to 3 pts.

`wspace`: scalar Width padding between subplots, expressed as a fraction of the subplot width. The total padding ends up being `w_pad + wspace`.

`hspace`: scalar Height padding between subplots, expressed as a fraction of the subplot width. The total padding ends up being `h_pad + hspace`.

`set_dpi(val)`

Set the dots-per-inch of the figure.

ACCEPTS: float

`set_edgecolor(color)`

Set the edge color of the Figure rectangle.

ACCEPTS: any matplotlib color - see `help(colors)`

`set_facecolor(color)`

Set the face color of the Figure rectangle.

ACCEPTS: any matplotlib color - see help(colors)

set_figheight(val, forward=True)

Set the height of the figure in inches.

ACCEPTS: float

set_figwidth(val, forward=True)

Set the width of the figure in inches.

ACCEPTS: float

set_frameon(b)

Set whether the figure frame (background) is displayed or invisible

ACCEPTS: boolean

set_size_inches(w, h=None, forward=True)

Set the figure size in inches (1in == 2.54cm)

Usage

```
fig.set_size_inches(w, h) # OR
fig.set_size_inches((w, h))
```

optional kwarg forward=True will cause the canvas size to be automatically updated; e.g., you can resize the figure window from the shell

ACCEPTS: a w, h tuple with w, h in inches

See also:

matplotlib.Figure.get_size_inches

set_tight_layout(tight)

Set whether and how tight_layout is called when drawing.

Parameters

tight [bool or dict with keys "pad", "w_pad", "h_pad", "rect" or None]

If a bool, sets whether to call tight_layout upon drawing. If None, use the figure.autolayout rcparam instead. If a dict, pass it as kwargs to tight_layout, overriding the default paddings.

show(warn=True)

If using a GUI backend with pyplot, display the figure window.

If the figure was not created using figure(), it will lack a FigureManagerBase, and will raise an AttributeError.

Parameters

warn [bool] If True, issue warning when called on a non-GUI backend

Notes

For non-GUI backends, this does nothing, in which case a warning will be issued if `warn` is `True` (default).

`subplots(nrows=1, ncols=1, sharex=False, sharey=False, squeeze=True, subplot_kw=None, gridspec_kw=None)`
Add a set of subplots to this figure.

Parameters

`nrows, ncols` [int, default: 1] Number of rows/cols of the subplot grid.

`sharex, sharey` [bool or {'none', 'all', 'row', 'col'}, default: False] Controls sharing of properties among x (`sharex`) or y (`sharey`) axes:

- `True` or `'all'`: x- or y-axis will be shared among all subplots.
- `False` or `'none'`: each subplot x- or y-axis will be independent.
- `'row'`: each subplot row will share an x- or y-axis.
- `'col'`: each subplot column will share an x- or y-axis.

When subplots have a shared x-axis along a column, only the x tick labels of the bottom subplot are created. Similarly, when subplots have a shared y-axis along a row, only the y tick labels of the first column subplot are created. To later turn other subplots' ticklabels on, use `tick_params()`.

`squeeze` [bool, optional, default: True]

- If `True`, extra dimensions are squeezed out from the returned array of Axes:
 - if only one subplot is constructed (`nrows=ncols=1`), the resulting single Axes object is returned as a scalar.
 - for `Nx1` or `1xM` subplots, the returned object is a 1D numpy object array of Axes objects.
 - for `NxM`, subplots with `N>1` and `M>1` are returned as a 2D array.
- If `False`, no squeezing at all is done: the returned Axes object is always a 2D array containing Axes instances, even if it ends up being `1x1`.

`subplot_kw` [dict, default: {}] Dict with keywords passed to the `add_subplot()` call used to create each subplots.

`gridspec_kw` [dict, default: {}] Dict with keywords passed to the `GridSpec` constructor used to create the grid the subplots are placed on.

Returns

`ax` [single Axes object or array of Axes objects] The added axes. The dimensions of the resulting array can be controlled with the `squeeze` keyword, see above.

See also:

`pyplot.subplots` `pyplot` API; docstring includes examples.

`subplots_adjust(*args, **kwargs)`

Call signature:

```
subplots_adjust(left=None, bottom=None, right=None, top=None,
                wspace=None, hspace=None)
```

Update the SubplotParams with kwargs (defaulting to rc when None) and update the subplot locations.

`suptitle(t, **kwargs)`

Add a centered title to the figure.

kwargs are `matplotlib.text.Text` properties. Using figure coordinates, the defaults are:

`x` [0.5] The x location of the text in figure coords

`y` [0.98] The y location of the text in figure coords

`horizontalalignment` ['center'] The horizontal alignment of the text

`verticalalignment` ['top'] The vertical alignment of the text

If the `fontproperties` keyword argument is given then the rcParams defaults for font-size (`figure.titlesize`) and fontweight (`figure.titleweight`) will be ignored in favour of the `FontProperties` defaults.

A `matplotlib.text.Text` instance is returned.

Example:

```
fig.suptitle('this is the figure title', fontsize=12)
```

`text(x, y, s, *args, **kwargs)`

Add text to figure.

Call signature:

```
text(x, y, s, fontdict=None, **kwargs)
```

Add text to figure at location `x, y` (relative 0-1 coords). See `text()` for the meaning of the other arguments.

kwargs control the Text properties:

	Property	Description
<code>agg_filter</code>		a filter function, which takes a (m, n, 3) float array and a dpi value
<code>alpha</code>		float (0.0 transparent through 1.0 opaque)
<code>animated</code>		bool
<code>backgroundcolor</code>		any matplotlib color
<code>bbox</code>		FancyBboxPatch prop dict

Continued on next page

Table 3 – continued from previous page

Property	Description
clip_box	a matplotlib.transforms.Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	any matplotlib color
contains	a callable function
family or fontfamily or fontname or name	[FONTNAME 'serif' 'sans-serif' 'cursive' 'fantasy' 'monospace']
figure	a Figure instance
fontproperties or font_properties	a matplotlib.font_manager.FontProperties instance
gid	an id string
horizontalalignment or ha	['center' 'right' 'left']
label	object
linespacing	float (multiple of font size)
multialignment or ma	['left' 'right' 'center']
path_effects	AbstractPathEffect
picker	[None bool float callable]
position	(x,y)
rasterized	bool or None
rotation	[angle in degrees 'vertical' 'horizontal']
rotation_mode	[None "default" "anchor"]
size or fontsize	[size in points 'xx-small' 'x-small' 'small' 'medium' 'large' 'x-large']
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
stretch or fontstretch	[a numeric value in range 0-1000 'ultra-condensed' 'extra-condensed']
style or fontstyle	['normal' 'italic' 'oblique']
text	string or anything printable with '%s' conversion.
transform	Transform
url	a url string
usetex	bool or None
variant or fontvariant	['normal' 'small-caps']
verticalalignment or va	['center' 'top' 'bottom' 'baseline']
visible	bool
weight or fontweight	[a numeric value in range 0-1000 'ultralight' 'light' 'normal' 'bold']
wrap	bool
x	float
y	float
zorder	float

`tight_layout(renderer=None, pad=1.08, h_pad=None, w_pad=None, rect=None)`

Adjust subplot parameters to give specified padding.

Parameters

`pad` [float] padding between the figure edge and the edges of subplots, as a fraction of the font-size.

`h_pad, w_pad` [float, optional] padding (height/width) between edges of adjacent subplots. Defaults to `pad_inches`.

`rect` [tuple (left, bottom, right, top), optional] a rectangle (left, bottom, right, top) in the normalized figure coordinate that the whole subplots area (including labels) will fit into. Default is (0, 0, 1, 1).

`waitforbuttonpress(timeout=-1)`

Blocking call to interact with the figure.

This will return True is a key was pressed, False if a mouse button was pressed and None if timeout was reached without either being pressed.

If timeout is negative, does not timeout.

Examples using `matplotlib.figure.Figure`

- `sphinx_glr_gallery_api_agg_oo_sgskip.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_custom_figure_class.py`
- `sphinx_glr_gallery_images_contours_and_fields_figimage_demo.py`
- `sphinx_glr_gallery_shapes_and_collections_collections.py`
- `sphinx_glr_gallery_misc_webapp_demo_sgskip.py`
- `sphinx_glr_gallery_user_interfaces_embedding_in_gtk3_sgskip.py`
- `sphinx_glr_gallery_user_interfaces_embedding_in_gtk_sgskip.py`
- `sphinx_glr_gallery_user_interfaces_embedding_in_gtk3_panzoom_sgskip.py`
- `sphinx_glr_gallery_user_interfaces_mpl_with_glade_316_sgskip.py`
- `sphinx_glr_gallery_user_interfaces_histogram_demo_canvasagg_sgskip.py`
- `sphinx_glr_gallery_user_interfaces_embedding_in_gtk2_sgskip.py`
- `sphinx_glr_gallery_user_interfaces_embedding_in_tk_sgskip.py`
- `sphinx_glr_gallery_user_interfaces_embedding_in_wx2_sgskip.py`
- `sphinx_glr_gallery_user_interfaces_embedding_in_wx5_sgskip.py`
- `sphinx_glr_gallery_user_interfaces_embedding_in_qt_sgskip.py`
- `sphinx_glr_gallery_user_interfaces_embedding_in_tk_canvas_sgskip.py`
- `sphinx_glr_gallery_user_interfaces_wxcursor_demo_sgskip.py`
- `sphinx_glr_gallery_user_interfaces_gtk_spreadsheet_sgskip.py`
- `sphinx_glr_gallery_user_interfaces_embedding_in_wx4_sgskip.py`
- `sphinx_glr_gallery_user_interfaces_mpl_with_glade_sgskip.py`
- `sphinx_glr_gallery_user_interfaces_mathtext_wx_sgskip.py`

- sphx_glr_gallery_user_interfaces_embedding_in_wx3_sgskip.py
- sphx_glr_gallery_user_interfaces_embedding_webagg_sgskip.py
- sphx_glr_gallery_user_interfaces_fourier_demo_wx_sgskip.py

matplotlib.figure.SubplotParams

```
class matplotlib.figure.SubplotParams(left=None, bottom=None, right=None, top=None,
                                     wspace=None, hspace=None)
```

A class to hold the parameters for a subplot.

All dimensions are fractions of the figure width or height. Defaults are given by rcParams["figure.subplot.[name]"].

Parameters

left [float] The left side of the subplots of the figure.

right [float] The right side of the subplots of the figure.

bottom [float] The bottom of the subplots of the figure.

top [float] The top of the subplots of the figure.

wspace [float] The amount of width reserved for space between subplots, expressed as a fraction of the average axis width.

hspace [float] The amount of height reserved for space between subplots, expressed as a fraction of the average axis height.

```
update(left=None, bottom=None, right=None, top=None, wspace=None,
       hspace=None)
```

Update the dimensions of the passed parameters. None means unchanged.

47.1.2 Functions

<code>figaspect(arg)</code>	Create a figure with specified aspect ratio.
-----------------------------	--

matplotlib.figure.figaspect

matplotlib.figure.figaspect(arg)

Create a figure with specified aspect ratio. If arg is a number, use that aspect ratio. If arg is an array, figaspect will determine the width and height for a figure that would fit array preserving aspect ratio. The figure width, height in inches are returned. Be sure to create an axes with equal width and height, e.g.,

Example usage:

```
# make a figure twice as tall as it is wide
w, h = figaspect(2.)
```

(continues on next page)

(continued from previous page)

```
fig = Figure(figsize=(w,h))
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])
ax.imshow(A, **kwargs)

# make a figure with the proper aspect for an array
A = rand(5,3)
w, h = figaspect(A)
fig = Figure(figsize=(w,h))
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])
ax.imshow(A, **kwargs)
```

Thanks to Fernando Perez for this function

FONT_MANAGER

48.1 matplotlib.font_manager

A module for finding, managing, and using fonts across platforms.

This module provides a single `FontManager` instance that can be shared across backends and platforms. The `findfont()` function returns the best TrueType (TTF) font file in the local or system font path that matches the specified `FontProperties` instance. The `FontManager` also handles Adobe Font Metrics (AFM) font files for use by the PostScript backend.

The design is based on the [W3C Cascading Style Sheet, Level 1 \(CSS1\) font specification](#). Future versions may implement the Level 2 or 2.1 specifications.

Experimental support is included for using `fontconfig` on Unix variant platforms (Linux, OS X, Solaris). To enable it, set the constant `USE_FONTCONFIG` in this file to `True`. `Fontconfig` has the advantage that it is the standard way to look up fonts on X11 platforms, so if a font is installed, it is much more likely to be found.

```
class matplotlib.font_manager.FontEntry(fname="", name="", style='normal', variant='normal', weight='normal', stretch='normal', size='medium')
```

Bases: `object`

A class for storing Font properties. It is used when populating the font lookup dictionary.

```
class matplotlib.font_manager.FontManager(size=None, weight='normal')
```

Bases: `object`

On import, the `FontManager` singleton instance creates a list of TrueType fonts based on the font properties: name, style, variant, weight, stretch, and size. The `findfont()` method does a nearest neighbor search to find the font that most closely matches the specification. If no good enough match is found, a default font is returned.

```
findfont(prop, fontext='ttf', directory=None, fallback_to_default=True, rebuild_if_missing=True)
```

Search the font list for the font that most closely matches the `FontProperties` prop.

`findfont()` performs a nearest neighbor search. Each font is given a similarity score to the target font properties. The first font with the highest score is returned. If no matches below a certain threshold are found, the default font (usually DejaVu Sans) is returned.

directory, is specified, will only return fonts from the given directory (or subdirectory of that directory).

The result is cached, so subsequent lookups don't have to perform the $O(n)$ nearest neighbor search.

If `fallback_to_default` is `True`, will fallback to the default font family (usually "DejaVu Sans" or "Helvetica") if the first lookup hard-fails.

See the [W3C Cascading Style Sheet, Level 1](#) documentation for a description of the font finding algorithm.

`static get_default_size()`

Return the default font size.

`get_default_weight()`

Return the default font weight.

`score_family(families, family2)`

Returns a match score between the list of font families in `families` and the font family name `family2`.

An exact match at the head of the list returns 0.0.

A match further down the list will return between 0 and 1.

No match will return 1.0.

`score_size(size1, size2)`

Returns a match score between `size1` and `size2`.

If `size2` (the size specified in the font file) is 'scalable', this function always returns 0.0, since any font size can be generated.

Otherwise, the result is the absolute distance between `size1` and `size2`, normalized so that the usual range of font sizes (6pt - 72pt) will lie between 0.0 and 1.0.

`score_stretch(stretch1, stretch2)`

Returns a match score between `stretch1` and `stretch2`.

The result is the absolute value of the difference between the CSS numeric values of `stretch1` and `stretch2`, normalized between 0.0 and 1.0.

`score_style(style1, style2)`

Returns a match score between `style1` and `style2`.

An exact match returns 0.0.

A match between 'italic' and 'oblique' returns 0.1.

No match returns 1.0.

`score_variant(variant1, variant2)`

Returns a match score between `variant1` and `variant2`.

An exact match returns 0.0, otherwise 1.0.

`score_weight(weight1, weight2)`

Returns a match score between `weight1` and `weight2`.

The result is 0.0 if both `weight1` and `weight2` are given as strings and have the same value.

Otherwise, the result is the absolute value of the difference between the CSS numeric values of `weight1` and `weight2`, normalized between 0.05 and 1.0.

`set_default_weight(weight)`

Set the default font weight. The initial value is 'normal'.

`update_fonts(filenames)`

Update the font dictionary with new font files. Currently not implemented.

```
class matplotlib.font_manager.FontProperties(family=None, style=None, variant=None,
                                             weight=None, stretch=None, size=None,
                                             fname=None, __init=None)
```

Bases: `object`

A class for storing and manipulating font properties.

The font properties are those described in the [W3C Cascading Style Sheet, Level 1](#) font specification. The six properties are:

- **family:** A list of font names in decreasing order of priority. The items may include a generic font family name, either 'serif', 'sans-serif', 'cursive', 'fantasy', or 'monospace'. In that case, the actual font to be used will be looked up from the associated rcParam in `matplotlibrc`.
- **style:** Either 'normal', 'italic' or 'oblique'.
- **variant:** Either 'normal' or 'small-caps'.
- **stretch:** A numeric value in the range 0-1000 or one of 'ultra-condensed', 'extra-condensed', 'condensed', 'semi-condensed', 'normal', 'semi-expanded', 'expanded', 'extra-expanded' or 'ultra-expanded'.
- **weight:** A numeric value in the range 0-1000 or one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.
- **size:** Either an relative value of 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large' or an absolute font size, e.g., 12.

The default font property for TrueType fonts (as specified in the default `matplotlibrc` file) is:

sans-serif, normal, normal, normal, normal, scalable.

Alternatively, a font may be specified using an absolute path to a .ttf file, by using the `fname` kwarg.

The preferred usage of font sizes is to use the relative values, e.g., 'large', instead of absolute font sizes, e.g., 12. This approach allows all text sizes to be made larger or smaller based on the font manager's default font size.

This class will also accept a [fontconfig](#) pattern, if it is the only argument provided. See the documentation on [fontconfig patterns](#). This support does not require fontconfig to be installed. We are merely borrowing its pattern syntax for use here.

Note that matplotlib's internal font manager and fontconfig use a different algorithm to lookup fonts, so the results of the same pattern may be different in matplotlib than in other applications that use fontconfig.

`copy()`

Return a deep copy of self

`get_family()`

Return a list of font names that comprise the font family.

`get_file()`

Return the filename of the associated font.

`get_fontconfig_pattern()`

Get a fontconfig pattern suitable for looking up the font as specified with fontconfig's fc-match utility.

See the documentation on [fontconfig patterns](#).

This support does not require fontconfig to be installed or support for it to be enabled. We are merely borrowing its pattern syntax for use here.

`get_name()`

Return the name of the font that best matches the font properties.

`get_size()`

Return the font size.

`get_size_in_points()`

`get_slant()`

Return the font style. Values are: 'normal', 'italic' or 'oblique'.

`get_stretch()`

Return the font stretch or width. Options are: 'ultra-condensed', 'extra-condensed', 'condensed', 'semi-condensed', 'normal', 'semi-expanded', 'expanded', 'extra-expanded', 'ultra-expanded'.

`get_style()`

Return the font style. Values are: 'normal', 'italic' or 'oblique'.

`get_variant()`

Return the font variant. Values are: 'normal' or 'small-caps'.

`get_weight()`

Set the font weight. Options are: A numeric value in the range 0-1000 or one of 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'

`set_family(family)`

Change the font family. May be either an alias (generic name is CSS parlance), such

as: 'serif', 'sans-serif', 'cursive', 'fantasy', or 'monospace', a real font name or a list of real font names. Real font names are not supported when `text.usetex` is `True`.

`set_file(file)`

Set the filename of the fontfile to use. In this case, all other properties will be ignored.

`set_fontconfig_pattern(pattern)`

Set the properties by parsing a fontconfig pattern.

See the documentation on [fontconfig patterns](#).

This support does not require fontconfig to be installed or support for it to be enabled. We are merely borrowing its pattern syntax for use here.

`set_name(family)`

Change the font family. May be either an alias (generic name is CSS parlance), such as: 'serif', 'sans-serif', 'cursive', 'fantasy', or 'monospace', a real font name or a list of real font names. Real font names are not supported when `text.usetex` is `True`.

`set_size(size)`

Set the font size. Either an relative value of 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large' or an absolute font size, e.g., 12.

`set_slant(style)`

Set the font style. Values are: 'normal', 'italic' or 'oblique'.

`set_stretch(stretch)`

Set the font stretch or width. Options are: 'ultra-condensed', 'extra-condensed', 'condensed', 'semi-condensed', 'normal', 'semi-expanded', 'expanded', 'extra-expanded' or 'ultra-expanded', or a numeric value in the range 0-1000.

`set_style(style)`

Set the font style. Values are: 'normal', 'italic' or 'oblique'.

`set_variant(variant)`

Set the font variant. Values are: 'normal' or 'small-caps'.

`set_weight(weight)`

Set the font weight. May be either a numeric value in the range 0-1000 or one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'

```
class matplotlib.font_manager.JSONEncoder(*, skipkeys=False, ensure_ascii=True,
                                           check_circular=True, allow_nan=True,
                                           sort_keys=False, indent=None, separators=None, default=None)
```

Bases: `json.encoder.JSONEncoder`

Constructor for JSONEncoder, with sensible defaults.

If `skipkeys` is false, then it is a `TypeError` to attempt encoding of keys that are not str, int, float or None. If `skipkeys` is True, such items are simply skipped.

If `ensure_ascii` is true, the output is guaranteed to be str objects with all incoming non-ASCII characters escaped. If `ensure_ascii` is false, the output can contain non-ASCII characters.

If `check_circular` is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an `OverflowError`). Otherwise, no such check takes place.

If `allow_nan` is true, then NaN, Infinity, and -Infinity will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a `ValueError` to encode such floats.

If `sort_keys` is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if `indent` is None and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

If specified, default is a function that gets called for objects that can't otherwise be serialized. It should return a JSON encodable version of the object or raise a `TypeError`.

`default(o)`

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement `default` like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

`matplotlib.font_manager.OSXInstalledFonts(directories=None, fonttext='ttf')`

Get list of font files on OS X - ignores font suffix by default.

`class matplotlib.font_manager.TempCache`

Bases: `object`

A class to store temporary caches that are (a) not saved to disk and (b) invalidated whenever certain font-related rcParams—namely the family lookup lists—are changed or the font cache is reloaded. This avoids the expensive linear search through all fonts every time a font is looked up.

`get(prop)`

`invalidating_rcparams = ('font.serif', 'font.sans-serif', 'font.cursive', 'font.fantasy', 'font.monospace')`

`make_rcparams_key()`

`set(prop, value)`

`matplotlib.font_manager.afmFontProperty(fontpath, font)`

Extract information from an AFM font file.

Parameters

`font` [AFM] The AFM font file from which information will be extracted.

Returns

‘FontEntry’ The extracted font properties.

`matplotlib.font_manager.createFontList(fontfiles, fonttext='ttf')`

A function to create a font lookup list. The default is to create a list of TrueType fonts. An AFM font list can optionally be created.

`matplotlib.font_manager.findSystemFonts(fontpaths=None, fonttext='ttf')`

Search for fonts in the specified font paths. If no paths are given, will use a standard set of system paths, as well as the list of fonts tracked by fontconfig if fontconfig is installed and available. A list of TrueType fonts are returned by default with AFM fonts as an option.

`matplotlib.font_manager.findfont(prop, **kw)`

`matplotlib.font_manager.get_font(filename, hinting_factor=None)`

`matplotlib.font_manager.get_fontconfig_fonts(fonttext='ttf')`

List the font filenames known to fc-list having the given extension.

`matplotlib.font_manager.get_fonttext_synonyms(fonttext)`

Return a list of file extensions extensions that are synonyms for the given file extension fileext.

`matplotlib.font_manager.is_opentype_cff_font(filename)`

Returns True if the given font is a Postscript Compact Font Format Font embedded in an OpenType wrapper. Used by the PostScript and PDF backends that can not subset these fonts.

`matplotlib.font_manager.json_dump(data, filename)`

Dumps a data structure as JSON in the named file. Handles FontManager and its fields.

`matplotlib.font_manager.json_load(filename)`

Loads a data structure as JSON from the named file. Handles FontManager and its fields.

`matplotlib.font_manager.list_fonts(directory, extensions)`

Return a list of all fonts matching any of the extensions, possibly upper-cased, found recursively under the directory.

`matplotlib.font_manager.ttfFontProperty(font)`

Extract information from a TrueType font file.

Parameters

`font` [FT2Font] The TrueType font file from which information will be extracted.

Returns

‘FontEntry’ The extracted font properties.

`matplotlib.font_manager.ttfdict_to_fnames(d)`

Deprecated since version 2.1: The `ttfdict_to_fnames` function was deprecated in version 2.1.

flatten a `ttfdict` to all the filenames it contains

`matplotlib.font_manager.weight_as_number(weight)`

Deprecated since version 2.1: The `weight_as_number` function was deprecated in version 2.1.

Return the weight property as a numeric value. String values are converted to their corresponding numeric value.

`matplotlib.font_manager.win32FontDirectory()`

Return the user-specified font directory for Win32. This is looked up from the registry key:

```
\\HKEY_CURRENT_
USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\Fonts
```

If the key is not found, `$WINDIR/Fonts` will be returned.

`matplotlib.font_manager.win32InstalledFonts(directory=None, fonttext='ttf')`

Search for fonts in the specified font directory, or use the system directories if none given. A list of TrueType font filenames are returned by default, or AFM fonts if `fonttext == 'afm'`.

48.2 matplotlib.fontconfig_pattern

A module for parsing and generating fontconfig patterns.

See the [fontconfig pattern specification](#) for more information.

`class matplotlib.fontconfig_pattern.FontconfigPatternParser`

Bases: `object`

A simple pyparsing-based parser for fontconfig-style patterns.

See the [fontconfig pattern specification](#) for more information.

`parse(pattern)`

Parse the given fontconfig pattern and return a dictionary of key/value pairs useful for initializing a `font_manager.FontProperties` object.

`matplotlib.fontconfig_pattern.family_escape($self, /, repl, string, count=0)`

Return the string obtained by replacing the leftmost non-overlapping occurrences of pattern in string by the replacement repl.

`matplotlib.fontconfig_pattern.family_unescape($self, /, repl, string, count=0)`

Return the string obtained by replacing the leftmost non-overlapping occurrences of pattern in string by the replacement repl.

`matplotlib.fontconfig_pattern.generate_fontconfig_pattern(d)`

Given a dictionary of key/value pairs, generates a fontconfig pattern string.

`matplotlib.fontconfig_pattern.parse_fontconfig_pattern`

Parse the given fontconfig pattern and return a dictionary of key/value pairs useful for initializing a `font_manager.FontProperties` object.

`matplotlib.fontconfig_pattern.value_escape($self, /, repl, string, count=0)`

Return the string obtained by replacing the leftmost non-overlapping occurrences of pattern in string by the replacement repl.

`matplotlib.fontconfig_pattern.value_unescape($self, /, repl, string, count=0)`

Return the string obtained by replacing the leftmost non-overlapping occurrences of pattern in string by the replacement repl.

GRIDSPEC

49.1 matplotlib.gridspec

gridspec is a module which specifies the location of the subplot in the figure.

`GridSpec` specifies the geometry of the grid that a subplot will be placed. The number of rows and number of columns of the grid need to be set. Optionally, the subplot layout parameters (e.g., left, right, etc.) can be tuned.

`SubplotSpec` specifies the location of the subplot in the given `GridSpec`.

49.1.1 Classes

<code>GridSpec(nrows, ncols[, figure, left, ...])</code>	A class that specifies the geometry of the grid that a subplot will be placed.
<code>SubplotSpec(gridspec, num1[, num2])</code>	Specifies the location of the subplot in the given <code>GridSpec</code> .
<code>GridSpecBase(nrows, ncols[, height_ratios, ...])</code>	A base class of <code>GridSpec</code> that specifies the geometry of the grid that a subplot will be placed.
<code>GridSpecFromSubplotSpec(nrows, ncols, ...[, ...])</code>	<code>GridSpec</code> whose subplot layout parameters are inherited from the location specified by a given <code>SubplotSpec</code> .

matplotlib.gridspec.GridSpec

```
class matplotlib.gridspec.GridSpec(nrows, ncols, figure=None, left=None, bottom=None,
                                   right=None, top=None, wspace=None, hspace=None,
                                   width_ratios=None, height_ratios=None)
```

A class that specifies the geometry of the grid that a subplot will be placed. The location of grid is determined by similar way as the `SubplotParams`.

The number of rows and number of columns of the grid need to be set. Optionally, the subplot layout parameters (e.g., left, right, etc.) can be tuned.

Parameters

`nrows` [int] Number of rows in grid.

`ncols` [int] Number or columns in grid.

Notes

See `SubplotParams` for descriptions of the layout parameters.

`get_subplot_params(figure=None, fig=None)`

Return a dictionary of subplot layout parameters. The default parameters are from `rcParams` unless a figure attribute is set.

`locally_modified_subplot_params()`

`tight_layout(figure, renderer=None, pad=1.08, h_pad=None, w_pad=None, rect=None)`

Adjust subplot parameters to give specified padding.

Parameters

`pad` [float] Padding between the figure edge and the edges of subplots, as a fraction of the font-size.

`h_pad, w_pad` [float, optional] Padding (height/width) between edges of adjacent subplots. Defaults to `pad_inches`.

`rect` [tuple of 4 floats, optional] (left, bottom, right, top) rectangle in normalized figure coordinates that the whole subplots area (including labels) will fit into. Default is (0, 0, 1, 1).

`update(**kwargs)`

Update the current values. If any kwarg is `None`, default to the current value, if set, otherwise to `rc`.

Examples using `matplotlib.gridspec.GridSpec`

- `sphx_glr_gallery_subplots_axes_and_figures_align_labels_demo.py`
- `sphx_glr_gallery_subplots_axes_and_figures_demo_tight_layout.py`
- `sphx_glr_gallery_lines_bars_and_markers_markevery_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_plot_streamplot.py`
- `sphx_glr_gallery_pie_and_polar_charts_pie_demo2.py`
- `sphx_glr_gallery_userdemo_demo_gridspec05.py`
- `sphx_glr_gallery_userdemo_demo_gridspec02.py`
- `sphx_glr_gallery_userdemo_demo_gridspec03.py`
- `sphx_glr_gallery_userdemo_demo_gridspec04.py`
- `sphx_glr_gallery_userdemo_demo_gridspec06.py`

- [Customizing Figure Layouts Using GridSpec and Other Functions](#)
- [Tight Layout guide](#)
- [origin and extent in imshow](#)
- [Constrained Layout Guide](#)

matplotlib.gridspec.SubplotSpec

class matplotlib.gridspec.SubplotSpec(gridspec, num1, num2=None)

Specifies the location of the subplot in the given GridSpec.

The subplot will occupy the num1-th cell of the given gridspec. If num2 is provided, the subplot will span between num1-th cell and num2-th cell.

The index starts from 0.

get_geometry()

Get the subplot geometry (n_rows, n_cols, start, stop).

start and stop are the index of the start and stop of the subplot.

get_gridspec()

get_position(figure, return_all=False)

Update the subplot position from figure.subplotspars.

get_rows_columns()

Get the subplot row and column numbers: (n_rows, n_cols, row_start, row_stop, col_start, col_stop)

get_topmost_subplotspec()

get the topmost SubplotSpec instance associated with the subplot

matplotlib.gridspec.GridSpecBase

class matplotlib.gridspec.GridSpecBase(nrows, ncols, height_ratios=None, width_ratios=None)

A base class of GridSpec that specifies the geometry of the grid that a subplot will be placed.

The number of rows and number of columns of the grid need to be set. Optionally, the ratio of heights and widths of rows and columns can be specified.

get_geometry()

get the geometry of the grid, e.g., 2,3

get_grid_positions(fig, raw=False)

return lists of bottom and top position of rows, left and right positions of columns.

If raw=True, then these are all in units relative to the container with no margins. (used for constrained_layout).

`get_height_ratios()`

`get_subplot_params(figure=None, fig=None)`

`get_width_ratios()`

`new_subplotspec(loc, rowspan=1, colspan=1)`
create and return a SubplotSpec instance.

`set_height_ratios(height_ratios)`

`set_width_ratios(width_ratios)`

`matplotlib.gridspec.GridSpecFromSubplotSpec`

```
class matplotlib.gridspec.GridSpecFromSubplotSpec(nrows,      ncols,      subplot_spec,
                                                    wspace=None,      hspace=None,
                                                    height_ratios=None,
                                                    width_ratios=None)
```

GridSpec whose subplot layout parameters are inherited from the location specified by a given SubplotSpec.

The number of rows and number of columns of the grid need to be set. An instance of SubplotSpec is also needed to be set from which the layout parameters will be inherited. The wspace and hspace of the layout can be optionally specified or the default values (from the figure or rcParams) will be used.

`get_subplot_params(figure=None, fig=None)`
Return a dictionary of subplot layout parameters.

`get_topmost_subplotspec()`
Get the topmost SubplotSpec instance associated with the subplot.

Examples using `matplotlib.gridspec.GridSpecFromSubplotSpec`

- [sphx_glr_gallery_userdemo_demo_gridspec04.py](#)
- [sphx_glr_gallery_userdemo_demo_gridspec06.py](#)
- [Customizing Figure Layouts Using GridSpec and Other Functions](#)
- [Constrained Layout Guide](#)

50.1 matplotlib.image

The image module supports basic image loading, rescaling and display operations.

```
class matplotlib.image.AxesImage(ax, cmap=None, norm=None, interpolation=None, ori-
                                gin=None, extent=None, filternorm=1, filterrad=4.0, re-
                                sample=False, **kwargs)
```

Bases: matplotlib.image._ImageBase

interpolation and cmap default to their rc settings

cmap is a colors.Colormap instance norm is a colors.Normalize instance to map luminance to 0-1

extent is data axes (left, right, bottom, top) for making image plots registered with data plots. Default is to label the pixel centers with the zero-based row and column indices.

Additional kwargs are matplotlib.artist properties

```
get_cursor_data(event)
```

Get the cursor data for a given event

```
get_extent()
```

Get the image extent: left, right, bottom, top

```
get_window_extent(renderer=None)
```

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

```
make_image(renderer, magnification=1.0, unsampled=False)
```

```
set_extent(extent)
```

extent is data axes (left, right, bottom, top) for making image plots

This updates `ax.dataLim`, and, if autoscaling, sets `viewLim` to tightly fit the image, regardless of `dataLim`. Autoscaling state is not changed, so following this with `ax.autoscale_view` will redo the autoscaling in accord with `dataLim`.

```
class matplotlib.image.BboxImage(bbox, cmap=None, norm=None, interpolation=None,
                                origin=None, filtnorm=1, filterrad=4.0, resam-
                                ple=False, interp_at__native=True, **kwargs)
Bases: matplotlib.image._ImageBase
```

The Image class whose size is determined by the given `bbox`.

`cmap` is a `colors.Colormap` instance `norm` is a `colors.Normalize` instance to map luminance to 0-1

`interp_at__native` is a flag that determines whether or not interpolation should still be applied when the image is displayed at its native resolution. A common use case for this is when displaying an image for annotational purposes; it is treated similarly to Photoshop (interpolation is only used when displaying the image at non-native resolutions).

`kwargs` are an optional list of Artist keyword args

`contains(mouseevent)`

Test whether the mouse event occurred within the image.

`get__transform()`

Return the Transform instance used by this artist.

`get__window__extent(renderer=None)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`make__image(renderer, magnification=1.0, unsampled=False)`

```
class matplotlib.image.FigureImage(fig, cmap=None, norm=None, offsetx=0, offsety=0, ori-
                                gin=None, **kwargs)
Bases: matplotlib.image._ImageBase
```

`cmap` is a `colors.Colormap` instance `norm` is a `colors.Normalize` instance to map luminance to 0-1

`kwargs` are an optional list of Artist keyword args

`get__extent()`

Get the image extent: left, right, bottom, top

`make__image(renderer, magnification=1.0, unsampled=False)`

```
set_data(A)
    Set the image array.

zorder = 0
```

```
class matplotlib.image.NonUniformImage(ax, **kwargs)
```

Bases: `matplotlib.image.AxesImage`

kwargs are identical to those for `AxesImage`, except that 'nearest' and 'bilinear' are the only supported 'interpolation' options.

```
get_extent()
    Get the image extent: left, right, bottom, top

make_image(renderer, magnification=1.0, unsampled=False)
```

```
set_array(*args)
    Retained for backwards compatibility - use set_data instead
```

ACCEPTS: numpy array A or PIL Image

```
set_cmap(cmap)
    set the colormap for luminance data

    ACCEPTS: a colormap or registered colormap name
```

```
set_data(x, y, A)
    Set the grid for the pixel centers, and the pixel values.

    x and y are monotonic 1-D ndarrays of lengths N and M, respectively, specifying pixel centers

    A is an (M,N) ndarray or masked array of values to be colormapped, or a (M,N,3) RGB array, or a (M,N,4) RGBA array.
```

```
set_filtnorm(s)
    Set whether the resize filter norms the weights – see help for imshow

    ACCEPTS: 0 or 1
```

```
set_filtrrad(s)
    Set the resize filter radius only applicable to some interpolation schemes – see help for imshow

    ACCEPTS: positive float
```

```
set_interpolation(s)
```

Parameters

s [str, None] Either 'nearest', 'bilinear', or None.

```
set_norm(norm)
    Set the normalization instance.
```

Parameters

norm [Normalize]

class matplotlib.image.PcolorImage(ax, x=None, y=None, A=None, cmap=None,
norm=None, **kwargs)

Bases: matplotlib.image.AxesImage

Make a pcolor-style plot with an irregular rectangular grid.

This uses a variation of the original irregular image code, and it is used by pcolorfast for the corresponding grid type.

cmap defaults to its rc setting

cmap is a colors.Colormap instance norm is a colors.Normalize instance to map luminance to 0-1

Additional kwargs are matplotlib.artist properties

get_cursor_data(event)

Get the cursor data for a given event

make_image(renderer, magnification=1.0, unsampled=False)

set_array(*args)

Retained for backwards compatibility - use set_data instead

ACCEPTS: numpy array A or PIL Image

set_data(x, y, A)

Set the grid for the rectangle boundaries, and the data values.

x and y are monotonic 1-D ndarrays of lengths N+1 and M+1, respectively, specifying rectangle boundaries. If None, they will be created as uniform arrays from 0 through N and 0 through M, respectively.

A is an (M,N) ndarray or masked array of values to be colormapped, or a (M,N,3) RGB array, or a (M,N,4) RGBA array.

matplotlib.image.composite_images(images, renderer, magnification=1.0)

Composite a number of RGBA images into one. The images are composited in the order in which they appear in the images list.

Parameters

images [list of Images] Each must have a make_image method. For each image, can_composite should return `True`, though this is not enforced by this function. Each image must have a purely affine transformation with no shear.

renderer [RendererBase instance]

magnification [float] The additional magnification to apply for the renderer in use.

Returns

tuple [image, offset_x, offset_y] Returns the tuple:

- image: A numpy array of the same type as the input images.
- offset_x, offset_y: The offset of the image (left, bottom) in the output figure.

`matplotlib.image.imread(fname, format=None)`

Read an image from a file into an array.

fname may be a string path, a valid URL, or a Python file-like object. If using a file object, it must be opened in binary mode.

If format is provided, will try to read file of that type, otherwise the format is deduced from the filename. If nothing can be deduced, PNG is tried.

Return value is a `numpy.array`. For grayscale images, the return array is $M \times N$. For RGB images, the return value is $M \times N \times 3$. For RGBA images the return value is $M \times N \times 4$.

matplotlib can only read PNGs natively, but if [PIL](#) is installed, it will use it to load the image and return an array (if possible) which can be used with `imshow()`. Note, URL strings may not be compatible with PIL. Check the PIL documentation for more information.

`matplotlib.image.imsave(fname, arr, vmin=None, vmax=None, cmap=None, format=None, origin=None, dpi=100)`

Save an array as in image file.

The output formats available depend on the backend being used.

Parameters

fname [str or file-like] Path string to a filename, or a Python file-like object. If format is None and fname is a string, the output format is deduced from the extension of the filename.

arr [array-like] An $M \times N$ (luminance), $M \times N \times 3$ (RGB) or $M \times N \times 4$ (RGBA) array.

vmin, vmax: [None | scalar] vmin and vmax set the color scaling for the image by fixing the values that map to the colormap color limits. If either vmin or vmax is None, that limit is determined from the arr min/max value.

cmap [matplotlib.colors.Colormap, optional] For example, `cm.viridis`. If None, defaults to the `image.cmap` rcParam.

format [str] One of the file extensions supported by the active backend. Most backends support png, pdf, ps, eps and svg.

origin [['upper' | 'lower']] Indicates whether the (0, 0) index of the array is in the upper left or lower left corner of the axes. Defaults to the `image.origin` rcParam.

dpi [int] The DPI to store in the metadata of the file. This does not affect the resolution of the output image.

`matplotlib.image.pil_to_array(pilImage)`

Load a PIL image and return it as a numpy array.

Grayscale images are returned as (M, N) arrays. RGB images are returned as (M, N, 3) arrays. RGBA images are returned as (M, N, 4) arrays.

`matplotlib.image.thumbnail(infile, thumbfile, scale=0.1, interpolation='bilinear', preview=False)`

make a thumbnail of image in infile with output filename thumbfile.

infile the image file – must be PNG or Pillow-readable if you have `Pillow` installed

thumbfile the thumbnail filename

scale the scale factor for the thumbnail

interpolation the interpolation scheme used in the resampling

preview if True, the default backend (presumably a user interface backend) will be used which will cause a figure to be raised if `show()` is called. If it is False, a pure image backend will be used depending on the extension, 'png' -> `FigureCanvasAgg`, 'pdf' -> `FigureCanvasPdf`, 'svg' -> `FigureCanvasSVG`

See `examples/misc/image_thumbnail.py`.

Return value is the figure instance containing the thumbnail

LEGEND AND LEGEND_HANDLER

51.1 matplotlib.legend

The legend module defines the Legend class, which is responsible for drawing legends associated with axes and/or figures.

Important: It is unlikely that you would ever create a Legend instance manually. Most users would normally create a legend via the `legend()` function. For more details on legends there is also a [legend guide](#).

The Legend class can be considered as a container of legend handles and legend texts. Creation of corresponding legend handles from the plot elements in the axes or figures (e.g., lines, patches, etc.) are specified by the handler map, which defines the mapping between the plot elements and the legend handlers to be used (the default legend handlers are defined in the `legend_handler` module). Note that not all kinds of artist are supported by the legend yet by default but it is possible to extend the legend handler's capabilities to support arbitrary objects. See the [legend guide](#) for more information.

```
class matplotlib.legend.DraggableLegend(legend, use_blit=False, update='loc')
```

```
    Bases: matplotlib.offsetbox.DraggableOffsetBox
```

Parameters

update [string] If "loc", update loc parameter of legend upon finalizing. If "bbox", update bbox_to_anchor parameter.

```
artist_picker(legend, evt)
```

```
finalize_offset()
```

```
class matplotlib.legend.Legend(parent, handles, labels, loc=None, numpoints=None,
                               markerscale=None, markerfirst=True, scatterpoints=None,
                               scatteryoffsets=None, prop=None, fontsize=None, borderpad=None,
                               labelspacing=None, handlelength=None, handleheight=None,
                               handletextpad=None, borderaxespad=None, columnspacing=None,
                               ncol=1, mode=None, fancybox=None, shadow=None, title=None,
                               framealpha=None, edgecolor=None, facecolor=None,
                               bbox_to_anchor=None, bbox_transform=None, frameon=None,
                               handler_map=None)
```

Bases: `matplotlib.artist.Artist`

Place a legend on the axes at location `loc`.

Parameters

`parent` [Axes or Figure] The artist that contains the legend.

`handles` [sequence of Artist] A list of Artists (lines, patches) to be added to the legend.

`labels` [sequence of strings] A list of labels to show next to the artists. The length of handles and labels should be the same. If they are not, they are truncated to the smaller of both lengths.

Other Parameters

`loc` [int or string or pair of floats, default: 'upper right'] The location of the legend. Possible codes are:

Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

Alternatively can be a 2-tuple giving `x`, `y` of the lower-left corner of the legend in axes coordinates (in which case `bbox_to_anchor` will be ignored).

`bbox_to_anchor` [BboxBase or pair of floats] Specify any arbitrary location for the legend in `bbox_transform` coordinates (default Axes coordinates).

For example, to put the legend's upper right hand corner in the center of the axes the following keywords can be used:

`loc='upper right', bbox_to_anchor=(0.5, 0.5)`

- `ncol` [integer] The number of columns that the legend has. Default is 1.
- `prop` [None or `matplotlib.font_manager.FontProperties` or dict] The font properties of the legend. If None (default), the current `matplotlib.rcParams` will be used.
- `fontsize` [int or float or {'xx-small', 'x-small', 'small', 'medium', 'large', 'xx-large', 'xx-large'}] Controls the font size of the legend. If the value is numeric the size will be the absolute font size in points. String values are relative to the current default font size. This argument is only used if `prop` is not specified.
- `numpoints` [None or int] The number of marker points in the legend when creating a legend entry for a `Line2D` (line). Default is None, which will take the value from `rcParams["legend.numpoints"]`.
- `scatterpoints` [None or int] The number of marker points in the legend when creating a legend entry for a `PathCollection` (scatter plot). Default is None, which will take the value from `rcParams["legend.scatterpoints"]`.
- `scatteryoffsets` [iterable of floats] The vertical offset (relative to the font size) for the markers created for a scatter plot legend entry. 0.0 is at the base the legend text, and 1.0 is at the top. To draw all markers at the same height, set to [0.5]. Default is [0.375, 0.5, 0.3125].
- `markerscale` [None or int or float] The relative size of legend markers compared with the originally drawn ones. Default is None, which will take the value from `rcParams["legend.markerscale"]`.
- `markerfirst` [bool] If True, legend marker is placed to the left of the legend label. If False, legend marker is placed to the right of the legend label. Default is True.
- `frameon` [None or bool] Control whether the legend should be drawn on a patch (frame). Default is None, which will take the value from `rcParams["legend.frameon"]`.
- `fancybox` [None or bool] Control whether round edges should be enabled around the `FancyBboxPatch` which makes up the legend's background. Default is None, which will take the value from `rcParams["legend.fancybox"]`.
- `shadow` [None or bool] Control whether to draw a shadow behind the legend. Default is None, which will take the value from `rcParams["legend.shadow"]`.
- `framealpha` [None or float] Control the alpha transparency of the legend's background. Default is None, which will take the value from `rcParams["legend.framealpha"]`. If shadow is activated and `framealpha` is None, the default value is ignored.
- `facecolor` [None or "inherit" or a color spec] Control the legend's background

- color. Default is None, which will take the value from rcParams["legend.facecolor"]. If "inherit", it will take rcParams["axes.facecolor"].
- edgecolor [None or "inherit" or a color spec] Control the legend's background patch edge color. Default is None, which will take the value from rcParams["legend.edgecolor"] If "inherit", it will take rcParams["axes.edgecolor"].
- mode [{ "expand", None}] If mode is set to "expand" the legend will be horizontally expanded to fill the axes area (or bbox_to_anchor if defines the legend's size).
- bbox_transform [None or matplotlib.transforms.Transform] The transform for the bounding box (bbox_to_anchor). For a value of None (default) the Axes' transAxes transform will be used.
- title [str or None] The legend's title. Default is no title (None).
- borderpad [float or None] The fractional whitespace inside the legend border. Measured in font-size units. Default is None, which will take the value from rcParams["legend.borderpad"].
- labelspacing [float or None] The vertical space between the legend entries. Measured in font-size units. Default is None, which will take the value from rcParams["legend.labelspacing"].
- handlelength [float or None] The length of the legend handles. Measured in font-size units. Default is None, which will take the value from rcParams["legend.handlelength"].
- handletextpad [float or None] The pad between the legend handle and text. Measured in font-size units. Default is None, which will take the value from rcParams["legend.handletextpad"].
- borderaxespad [float or None] The pad between the axes and legend border. Measured in font-size units. Default is None, which will take the value from rcParams["legend.borderaxespad"].
- columnspacing [float or None] The spacing between columns. Measured in font-size units. Default is None, which will take the value from rcParams["legend.columnspacing"].
- handler_map [dict or None] The custom dictionary mapping instances or types to a legend handler. This handler_map updates the default handler map found at matplotlib.legend.Legend.get_legend_handler_map().

Notes

Users can specify any arbitrary location for the legend using the `bbox_to_anchor` keyword argument. `bbox_to_anchor` can be an instance of `BboxBase`(or its derivatives) or a tuple of 2 or 4 floats. See `set_bbox_to_anchor()` for more detail.

The legend location can be specified by setting `loc` with a tuple of 2 floats, which is interpreted as the lower-left corner of the legend in the normalized axes coordinate.

`codes = {'best': 0, 'center': 10, 'center left': 6, 'center right': 7, 'lower center': 8, 'lower left': 3, 'lower right': 4}`

`contains(event)`

Test whether the artist contains the mouse event.

Returns the truth value and a dictionary of artist specific details of selection, such as which points are contained in the pick radius. See individual artists for details.

`draggable(state=None, use_blit=False, update='loc')`

Set the draggable state – if state is

- None : toggle the current state
- True : turn draggable on
- False : turn draggable off

If draggable is on, you can drag the legend on the canvas with the mouse. The `DraggableLegend` helper instance is returned if draggable is on.

The update parameter control which parameter of the legend changes when dragged. If update is "loc", the `loc` parameter of the legend is changed. If "bbox", the `bbox_to_anchor` parameter is changed.

`draw(renderer)`

Draw everything that belongs to the legend.

`draw_frame(b)`

Set draw frame to b.

Parameters

b [bool]

`get_bbox_to_anchor()`

Return the bbox that the legend will be anchored to.

`get_children()`

Return a list of child artists.

`classmethod get_default_handler_map()`

A class method that returns the default handler map.

`get_frame()`

Return the `Rectangle` instances used to frame the legend.

`get_frame_on()`

Get whether the legend box patch is drawn.

`static get_legend_handler(legend_handler_map, orig_handle)`

Return a legend handler from `legend_handler_map` that corresponds to `orig_handler`.

`legend_handler_map` should be a dictionary object (that is returned by the `get_legend_handler_map` method).

It first checks if the `orig_handle` itself is a key in the `legend_handler_map` and return the associated value. Otherwise, it checks for each of the classes in its method-resolution-order. If no matching key is found, it returns `None`.

`get_legend_handler_map()`

Return the handler map.

`get_lines()`

Return a list of `Line2D` instances in the legend.

`get_patches()`

Return a list of `Patch` instances in the legend.

`get_texts()`

Return a list of `Text` instances in the legend.

`get_title()`

Return the `Text` instance for the legend title.

`get_window_extent(*args, **kwargs)`

Return extent of the legend.

`set_bbox_to_anchor(bbox, transform=None)`

Set the `bbox` that the legend will be anchored to.

`bbox` can be

- A `BboxBase` instance
- A tuple of (left, bottom, width, height) in the given transform (normalized axes coordinate if `None`)
- A tuple of (left, bottom) where the width and height will be assumed to be zero.

classmethod `set_default_handler_map(handler_map)`

A class method to set the default handler map.

`set_frame_on(b)`

Set whether the legend box patch is drawn.

Parameters

`b` [bool]

`set_title(title, prop=None)`

Set the legend title. Fontproperties can be optionally set with `prop` parameter.

classmethod `update_default_handler_map(handler_map)`

A class method to update the default handler map.

`zorder = 5`

51.2 matplotlib.legend_handler

This module defines default legend handlers.

It is strongly encouraged to have read the [legend guide](#) before this documentation.

Legend handlers are expected to be a callable object with a following signature.

```
legend_handler(legend, orig_handle, fontsize, handlebox)
```

Where `legend` is the legend itself, `orig_handle` is the original plot, `fontsize` is the fontsize in pixels, and `handlebox` is a `OffsetBox` instance. Within the call, you should create relevant artists (using relevant properties from the legend and/or `orig_handle`) and add them into the `handlebox`. The artists needs to be scaled according to the fontsize (note that the size is in pixel, i.e., this is dpi-scaled value).

This module includes definition of several legend handler classes derived from the base class (`HandlerBase`) with the following method:

```
def legend_artist(self, legend, orig_handle, fontsize, handlebox):
```

```
class matplotlib.legend_handler.HandlerBase(xpad=0.0, ypad=0.0, update_func=None)
```

A Base class for default legend handlers.

The derived classes are meant to override `create_artists` method, which has a following signature.:

```
def create_artists(self, legend, orig_handle,
                  xdescent, ydescent, width, height, fontsize,
                  trans):
```

The overridden method needs to create artists of the given transform that fits in the given dimension (`xdescent`, `ydescent`, `width`, `height`) that are scaled by `fontsize` if necessary.

```
adjust_drawing_area(legend, orig_handle, xdescent, ydescent, width, height, fontsize)
```

```
create_artists(legend, orig_handle, xdescent, ydescent, width, height, fontsize, trans)
```

```
legend_artist(legend, orig_handle, fontsize, handlebox)
```

Return the artist that this `HandlerBase` generates for the given original artist/handle.

Parameters

`legend` [`matplotlib.legend.Legend` instance] The legend for which these legend artists are being created.

`orig_handle` [`matplotlib.artist.Artist` or similar] The object for which these legend artists are being created.

`fontsize` [float or int] The fontsize in pixels. The artists being created should be scaled according to the given fontsize.

handlebox [matplotlib.offsetbox.OffsetBox instance] The box which has been created to hold this legend entry's artists. Artists created in the legend_artist method must be added to this handlebox inside this method.

update_prop(legend_handle, orig_handle, legend)

class matplotlib.legend_handler.HandlerCircleCollection(yoffsets=None, sizes=None, **kw)
Handler for CircleCollections.

create_collection(orig_handle, sizes, offsets, transOffset)

class matplotlib.legend_handler.HandlerErrorbar(xerr_size=0.5, yerr_size=None, marker_pad=0.3, numpoints=None, **kw)
Handler for Errorbars.

create_artists(legend, orig_handle, xdescent, ydescent, width, height, fontsize, trans)

get_err_size(legend, xdescent, ydescent, width, height, fontsize)

class matplotlib.legend_handler.HandlerLine2D(marker_pad=0.3, numpoints=None, **kw)
Handler for Line2D instances.

Parameters

marker_pad [float] Padding between points in legend entry.

numpoints [int] Number of points to show in legend entry.

Notes

Any other keyword arguments are given to HandlerNpoints.

create_artists(legend, orig_handle, xdescent, ydescent, width, height, fontsize, trans)

class matplotlib.legend_handler.HandlerLineCollection(marker_pad=0.3, numpoints=None, **kw)
Handler for LineCollection instances.

Parameters

marker_pad [float] Padding between points in legend entry.

numpoints [int] Number of points to show in legend entry.

Notes

Any other keyword arguments are given to HandlerNpoints.

```
create_artists(legend, orig_handle, xdescent, ydescent, width, height, fontsize, trans)
```

```
get_numpoints(legend)
```

```
class matplotlib.legend_handler.HandlerNpoints(marker_pad=0.3, numpoints=None, **kw)
```

A legend handler that shows numpoints points in the legend entry.

Parameters

marker_pad [float] Padding between points in legend entry.

numpoints [int] Number of points to show in legend entry.

Notes

Any other keyword arguments are given to HandlerBase.

```
get_numpoints(legend)
```

```
get_xdata(legend, xdescent, ydescent, width, height, fontsize)
```

```
class matplotlib.legend_handler.HandlerNpointsYoffsets(numpoints=None, yoffsets=None,
                                                         **kw)
```

A legend handler that shows numpoints in the legend, and allows them to be individually offset in the y-direction.

Parameters

numpoints [int] Number of points to show in legend entry.

yoffsets [array of floats] Length numpoints list of y offsets for each point in legend entry.

Notes

Any other keyword arguments are given to HandlerNpoints.

```
get_ydata(legend, xdescent, ydescent, width, height, fontsize)
```

```
class matplotlib.legend_handler.HandlerPatch(patch_func=None, **kw)
```

Handler for Patch instances.

Parameters

patch_func [callable, optional] The function that creates the legend key artist.
patch_func should have the signature:

```
def patch_func(legend=legend, orig_handle=orig_handle,
               xdescent=xdescent, ydescent=ydescent,
               width=width, height=height, fontsize=fontsize)
```

Subsequently the created artist will have its `update_prop` method called and the appropriate transform will be applied.

Notes

Any other keyword arguments are given to `HandlerBase`.

```
create_artists(legend, orig_handle, xdescent, ydescent, width, height, fontsize, trans)
```

```
class matplotlib.legend_handler.HandlerPathCollection(yoffsets=None, sizes=None, **kw)
    Handler for PathCollections, which are used by scatter.
```

```
create_collection(orig_handle, sizes, offsets, transOffset)
```

```
class matplotlib.legend_handler.HandlerPolyCollection(xpad=0.0, ypad=0.0, update_func=None)
    Handler for PolyCollection used in fill_between and stackplot.
```

```
create_artists(legend, orig_handle, xdescent, ydescent, width, height, fontsize, trans)
```

```
class matplotlib.legend_handler.HandlerRegularPolyCollection(yoffsets=None, sizes=None, **kw)
    Handler for RegularPolyCollections.
```

```
create_artists(legend, orig_handle, xdescent, ydescent, width, height, fontsize, trans)
```

```
create_collection(orig_handle, sizes, offsets, transOffset)
```

```
get_numpoints(legend)
```

```
get_sizes(legend, orig_handle, xdescent, ydescent, width, height, fontsize)
```

```
update_prop(legend_handle, orig_handle, legend)
```

```
class matplotlib.legend_handler.HandlerStem(marker_pad=0.3, numpoints=None, bottom=None, yoffsets=None, **kw)
    Handler for plots produced by stem.
```

Parameters

`marker_pad` [float] Padding between points in legend entry. Default is 0.3.

`numpoints` [int, optional] Number of points to show in legend entry.

`bottom` [float, optional]

`yoffsets` [array of floats, optional] Length numpoints list of y offsets for each point in legend entry.

Notes

Any other keyword arguments are given to `HandlerNpointsYoffsets`.

`create_artists(legend, orig_handle, xdescent, ydescent, width, height, fontsize, trans)`

`get_ydata(legend, xdescent, ydescent, width, height, fontsize)`

`class matplotlib.legend_handler.HandlerTuple(ndivide=1, pad=None, **kwargs)`
Handler for Tuple.

Additional kwargs are passed through to `HandlerBase`.

Parameters

`ndivide` [int, optional] The number of sections to divide the legend area into. If None, use the length of the input tuple. Default is 1.

`pad` [float, optional] If None, fall back to `legend.borderpad` as the default. In units of fraction of font size. Default is None.

`create_artists(legend, orig_handle, xdescent, ydescent, width, height, fontsize, trans)`

`matplotlib.legend_handler.update_from_first_child(tgt, src)`

LINES

52.1 matplotlib.lines

This module contains all the 2D line class which can draw with a variety of line styles, markers and colors.

52.1.1 Classes

Line2D(xdata, ydata[, linewidth, linestyle, ...])	A line - the line can have both a solid linestyle connecting all the vertices, and a marker at each vertex.
VertexSelector(line)	Manage the callbacks to maintain a list of selected vertices for matplotlib.lines.Line2D.

matplotlib.lines.Line2D

```
class matplotlib.lines.Line2D(xdata, ydata, linewidth=None, linestyle=None,
                              color=None, marker=None, markersize=None, mark-
                              edgedwidth=None, markeredgewidth=None, markerface-
                              color=None, markerfacecoloralt='none', fillstyle=None, an-
                              tialiased=None, dash_capstyle=None, solid_capstyle=None,
                              dash_joinstyle=None, solid_joinstyle=None, pickradius=5,
                              drawstyle=None, markevery=None, **kwargs)
```

A line - the line can have both a solid linestyle connecting all the vertices, and a marker at each vertex. Additionally, the drawing of the solid line is influenced by the drawstyle, e.g., one can create "stepped" lines in various styles.

Create a Line2D instance with x and y data in sequences xdata, ydata.

The kwargs are Line2D properties:

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)
alpha		float (0.0 transparent through 1.0 opaque)

Continued on next page

Table 2 – continued from previous page

	Property	Description
animated	bool	
antialiased or aa	bool	
clip_box	a Bbox instance	
clip_on	bool	
clip_path	[(Path, Transform) Patch None]	
color or c	any matplotlib color	
contains	a callable function	
dash_capstyle	['butt' 'round' 'projecting']	
dash_joinstyle	['miter' 'round' 'bevel']	
dashes	sequence of on/off ink in points	
drawstyle	['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']	
figure	a Figure instance	
fillstyle	['full' 'left' 'right' 'bottom' 'top' 'none']	
gid	an id string	
label	object	
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']	
linewidth or lw	float value in points	
marker	A valid marker style	
markeredgcolor or mec	any matplotlib color	
markeredgewidth or mew	float value in points	
markerfacecolor or mfc	any matplotlib color	
markerfacecoloralt or mfcalt	any matplotlib color	
markersize or ms	float	
markevery	[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]	
path_effects	AbstractPathEffect	
picker	float distance in points or callable pick function fn(artist, event)	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
solid_capstyle	['butt' 'round' 'projecting']	
solid_joinstyle	['miter' 'round' 'bevel']	
transform	a matplotlib.transforms.Transform instance	
url	a url string	
visible	bool	
xdata	1D array	
ydata	1D array	
zorder	float	

See `set_linestyle()` for a description of the line styles, `set_marker()` for a description of the markers, and `set_drawstyle()` for a description of the draw styles.

axes

The Axes instance the artist resides in, or None.

`contains(mouseevent)`

Test whether the mouse event occurred on the line. The pick radius determines the precision of the location test (usually within five points of the value). Use `get_pickradius()` or `set_pickradius()` to view or modify it.

Returns True if any values are within the radius along with `{'ind': pointlist}`, where pointlist is the set of points within the radius.

TODO: sort returned indices by distance

`draw(renderer)`

draw the Line with renderer unless visibility is False

`drawStyleKeys = ['default', 'steps-mid', 'steps-pre', 'steps-post', 'steps']`

`drawStyles = {'default': '_draw_lines', 'steps': '_draw_steps_pre', 'steps-mid': '_draw_steps_mid', 'steps-post': '_draw_steps_post'}`

`fillStyles = ('full', 'left', 'right', 'bottom', 'top', 'none')`

`filled_markers = ('o', 'v', '^', '<', '>', '8', 's', 'p', '*', 'h', 'H', 'D', 'd', 'P', 'X')`

`get_aa()`

alias for `get_antialiased`

`get_antialiased()`

`get_c()`

alias for `get_color`

`get_color()`

`get_dash_capstyle()`

Get the cap style for dashed linestyles

`get_dash_joinstyle()`

Get the join style for dashed linestyles

`get_data(orig=True)`

Return the xdata, ydata.

If orig is True, return the original data.

`get_drawstyle()`

`get_fillstyle()`

return the marker fillstyle

`get_linestyle()`

`get_linewidth()`

`get_ls()`
alias for `get_linestyle`

`get_lw()`
alias for `get_linewidth`

`get_marker()`

`get_markeredgecolor()`

`get_markerewidth()`

`get_markerfacecolor()`

`get_markerfacecoloralt()`

`get_markersize()`

`get_markevery()`
return the markevery setting

`get_mec()`
alias for `get_markeredgecolor`

`get_mew()`
alias for `get_markerewidth`

`get_mfc()`
alias for `get_markerfacecolor`

`get_mfcalt(alt=False)`
alias for `get_markerfacecoloralt`

`get_ms()`
alias for `get_markersize`

`get_path()`
Return the Path object associated with this line.

`get_pickradius()`
return the pick radius used for containment tests

`get_solid_capstyle()`
Get the cap style for solid linestyles

`get_solid_joinstyle()`

Get the join style for solid linestyles

`get_window_extent(renderer)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`get_xdata(orig=True)`

Return the xdata.

If orig is True, return the original data, else the processed data.

`get_xydata()`

Return the xy data as a Nx2 numpy array.

`get_ydata(orig=True)`

Return the ydata.

If orig is True, return the original data, else the processed data.

`is_dashed()`

return True if line is dashstyle

`lineStyles = {'': '_draw_nothing', ' ': '_draw_nothing', '-': '_draw_solid', '--': '_draw_dashed', '-.': '_draw_dashdot'}`

`markers = {'.': 'point', ',': 'pixel', 'o': 'circle', 'v': 'triangle_down', '^': 'triangle_up', '<': 'triangle_left', '>': 'triangle_right'}`

`recache(always=False)`

`recache_always()`

`set_aa(val)`

alias for `set_antialiased`

`set_antialiased(b)`

Set whether to use antialiased rendering.

Parameters

b [bool]

`set_c(val)`

alias for `set_color`

`set_color(color)`

Set the color of the line

ACCEPTS: any matplotlib color

`set_dash_capstyle(s)`

Set the cap style for dashed linestyles

ACCEPTS: ['butt' | 'round' | 'projecting']

`set_dash_joinstyle(s)`

Set the join style for dashed linestyles ACCEPTS: ['miter' | 'round' | 'bevel']

`set_dashes(seq)`

Set the dash sequence, sequence of dashes with on off ink in points. If seq is empty or if seq = (None, None), the linestyle will be set to solid.

ACCEPTS: sequence of on/off ink in points

`set_data(*args)`

Set the x and y data

ACCEPTS: 2D array (rows are x, y) or two 1D arrays

`set_drawstyle(drawstyle)`

Set the drawstyle of the plot

'default' connects the points with lines. The steps variants produce step-plots. 'steps' is equivalent to 'steps-pre' and is maintained for backward-compatibility.

ACCEPTS: ['default' | 'steps' | 'steps-pre' | 'steps-mid' | 'steps-post']

`set_fillstyle(fs)`

Set the marker fill style; 'full' means fill the whole marker. 'none' means no filling; other options are for half-filled markers.

ACCEPTS: ['full' | 'left' | 'right' | 'bottom' | 'top' | 'none']

`set_linestyle(ls)`

Set the linestyle of the line (also accepts drawstyles, e.g., 'steps--')

linestyle	description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line
'None'	draw nothing
' '	draw nothing
' '	draw nothing

'steps' is equivalent to 'steps-pre' and is maintained for backward-compatibility.

Alternatively a dash tuple of the following form can be provided:

(offset, onoffseq),

where onoffseq is an even length tuple of on and off ink in points.

ACCEPTS: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '--' |
 '-.' | ':' | 'None' | '' | '']

See also:

`set_drawstyle()` To set the drawing style (stepping) of the plot.

Parameters

`ls` [{ '-' | '--', '-.', ':'} and more see description] The line style.

`set_linewidth(w)`

Set the line width in points

ACCEPTS: float value in points

`set_ls(val)`

alias for `set_linestyle`

`set_lw(val)`

alias for `set_linewidth`

`set_marker(marker)`

Set the line marker

ACCEPTS: A valid marker style

Parameters

`marker`: marker style See markers for full description of possible argument

`set_markedgcolor(ec)`

Set the marker edge color

ACCEPTS: any matplotlib color

`set_markedgwidth(ew)`

Set the marker edge width in points

ACCEPTS: float value in points

`set_markerfacecolor(fc)`

Set the marker face color.

ACCEPTS: any matplotlib color

`set_markerfacecoloralt(fc)`

Set the alternate marker face color.

ACCEPTS: any matplotlib color

`set_markersize(sz)`

Set the marker size in points

ACCEPTS: float

`set_markevery(every)`

Set the `markevery` property to subsample the plot when using markers.

e.g., if `every=5`, every 5-th marker will be plotted.

ACCEPTS: [None | int | length-2 tuple of int | slice | list/array of int | float | length-2 tuple of float]

Parameters

`every`: None | int | length-2 tuple of int | slice | list/array of int | float | length-2 tuple of float
Which markers to plot.

- `every=None`, every point will be plotted.
- `every=N`, every N-th marker will be plotted starting with marker 0.
- `every=(start, N)`, every N-th marker, starting at point `start`, will be plotted.
- `every=slice(start, end, N)`, every N-th marker, starting at point `start`, upto but not including point `end`, will be plotted.
- `every=[i, j, m, n]`, only markers at points `i`, `j`, `m`, and `n` will be plotted.
- `every=0.1`, (i.e. a float) then markers will be spaced at approximately equal distances along the line; the distance along the line between markers is determined by multiplying the display-coordinate distance of the axes bounding-box diagonal by the value of `every`.
- `every=(0.5, 0.1)` (i.e. a length-2 tuple of float), the same functionality as `every=0.1` is exhibited but the first marker will be 0.5 multiplied by the display-coordinate-diagonal-distance along the line.

Notes

Setting the `markevery` property will only show markers at actual data points. When using float arguments to set the `markevery` property on irregularly spaced data, the markers will likely not appear evenly spaced because the actual data points do not coincide with the theoretical spacing between markers.

When using a start offset to specify the first marker, the offset will be from the first data point which may be different from the first the visible data point if the plot is zoomed in.

If zooming in on a plot when using float arguments then the actual data points that have markers will change because the distance between markers is always determined from the display-coordinates axes-bounding-box-diagonal regardless of the actual axes data limits.

`set_mec(val)`

alias for `set_markeredgcolor`

`set__mew(val)`
 alias for `set__markeredgewidth`

`set__mfc(val)`
 alias for `set__markerfacecolor`

`set__mfcalt(val)`
 alias for `set__markerfacecoloralt`

`set__ms(val)`
 alias for `set__markersize`

`set__picker(p)`
 Sets the event picker details for the line.
 ACCEPTS: float distance in points or callable pick function `fn(artist, event)`

`set__pickradius(d)`
 Set the pick radius used for containment tests.
 Parameters
 `d` [float] Pick radius, in points.

`set__solid__capstyle(s)`
 Set the cap style for solid linestyles
 ACCEPTS: ['butt' | 'round' | 'projecting']

`set__solid__joinstyle(s)`
 Set the join style for solid linestyles ACCEPTS: ['miter' | 'round' | 'bevel']

`set__transform(t)`
 set the Transformation instance used by this artist
 ACCEPTS: a `matplotlib.transforms.Transform` instance

`set__xdata(x)`
 Set the data `np.array` for `x`
 ACCEPTS: 1D array

`set__ydata(y)`
 Set the data `np.array` for `y`
 ACCEPTS: 1D array

`update__from(other)`
 copy properties from other to self

`validCap = ('butt', 'round', 'projecting')`

`validJoin = ('miter', 'round', 'bevel')`

`zorder = 2`

Examples using `matplotlib.lines.Line2D`

- `sphinx_glr_gallery_api_line_with_text.py`
- `sphinx_glr_gallery_pyplots_fig_x.py`
- `sphinx_glr_gallery_shapes_and_collections_artist_reference.py`
- `sphinx_glr_gallery_text_labels_and_annotations_custom_legends.py`
- `sphinx_glr_gallery_text_labels_and_annotations_legend_demo.py`
- Oscilloscope
- `sphinx_glr_gallery_event_handling_pick_event_demo.py`
- `sphinx_glr_gallery_event_handling_poly_editor.py`
- `sphinx_glr_gallery_units_artist_tests.py`
- [Legend guide](#)
- [Artist tutorial](#)

`matplotlib.lines.VertexSelector`

`class matplotlib.lines.VertexSelector(line)`

Manage the callbacks to maintain a list of selected vertices for `matplotlib.lines.Line2D`. Derived classes should override `process_selected()` to do something with the picks.

Here is an example which highlights the selected verts with red circles:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.lines as lines

class HighlightSelected(lines.VertexSelector):
    def __init__(self, line, fmt='ro', **kwargs):
        lines.VertexSelector.__init__(self, line)
        self.markers, = self.axes.plot([], [], fmt, **kwargs)

    def process_selected(self, ind, xs, ys):
        self.markers.set_data(xs, ys)
        self.canvas.draw()

fig, ax = plt.subplots()
x, y = np.random.rand(2, 30)
line, = ax.plot(x, y, 'bs-', picker=5)

selector = HighlightSelected(line)
plt.show()
```

Initialize the class with a `matplotlib.lines.Line2D` instance. The line should already be added to some `matplotlib.axes.Axes` instance and should have the `picker` property set.

`onpick(event)`

When the line is picked, update the set of selected indices.

`process_selected(ind, xs, ys)`

Default "do nothing" implementation of the `process_selected()` method.

`ind` are the indices of the selected vertices. `xs` and `ys` are the coordinates of the selected vertices.

52.1.2 Functions

`segment_hits(cx, cy, x, y, radius)`

Determine if any line segments are within radius of a point.

`matplotlib.lines.segment_hits`

`matplotlib.lines.segment_hits(cx, cy, x, y, radius)`

Determine if any line segments are within radius of a point. Returns the list of line segments that are within that radius.

MARKERS

53.1 matplotlib.markers

This module contains functions to handle markers. Used by both the marker functionality of plot and scatter.

All possible markers are defined here:

marker	description
","	point
" "	pixel
"o"	circle
"v"	triangle_down
"^"	triangle_up
"<"	triangle_left
">"	triangle_right
"1"	tri_down
"2"	tri_up
"3"	tri_left
"4"	tri_right
"8"	octagon
"s"	square
"p"	pentagon
"P"	plus (filled)
"*"	star
"h"	hexagon1
"H"	hexagon2
"+"	plus
"x"	x
"X"	x (filled)
"D"	diamond
"d"	thin_diamond
" "	vline
"_"	hline

Continued on next page

Table 1 – continued from previous page

marker	description
TICKLEFT	tickleft
TICKRIGHT	tickright
TICKUP	tickup
TICKDOWN	tickdown
CARETLEFT	caretleft (centered at tip)
CARETRIGHT	caretright (centered at tip)
CARETUP	caretup (centered at tip)
CARETDOWN	caretdown (centered at tip)
CARETLEFTBASE	caretleft (centered at base)
CARETRIGHTBASE	caretright (centered at base)
CARETUPBASE	caretup (centered at base)
"None", " " or ""	nothing
'\$...\$'	render the string using mathtext.
verts	a list of (x, y) pairs used for Path vertices. The center of the marker is located at (0,0) and the size is normalized.
path	a Path instance.
(numsides, style, angle)	<p>The marker can also be a tuple (numsides, style, angle), which will create a custom, regular symbol.</p> <p>numsides: the number of sides</p> <p>style: the style of the regular symbol:</p> <ul style="list-style-type: none">0 a regular polygon1 a star-like symbol2 an asterisk3 a circle (numsides and angle is ignored) <p>angle: the angle of rotation of the symbol</p>

For backward compatibility, the form (verts, 0) is also accepted, but it is equivalent to just verts for giving a raw set of vertices that define the shape.

`None` is the default which means 'nothing', however this table is referred to from other docs for the valid inputs from marker inputs and in those cases `None` still means 'default'.

53.1.1 Classes

MarkerStyle([marker, fillstyle])

Parameters

matplotlib.markers.MarkerStyle

class matplotlib.markers.MarkerStyle(marker=None, fillstyle=None)

Parameters

marker [string or array_like, optional, default: None] See the descriptions of possible markers in the module docstring.

fillstyle [string, optional, default: 'full'] 'full', 'left', 'right', 'bottom', 'top', 'none'

Attributes

markers [list of known marks]

fillstyles [list of known fillstyles]

filled_markers [list of known filled markers.]

filled_markers = ('o', 'v', '^', '<', '>', '8', 's', 'p', '*', 'h', 'H', 'D', 'd', 'P', 'X')

fillstyles = ('full', 'left', 'right', 'bottom', 'top', 'none')

get_alt_path()

get_alt_transform()

get_capstyle()

get_fillstyle()

get_joinstyle()

get_marker()

get_path()

get_snap_threshold()

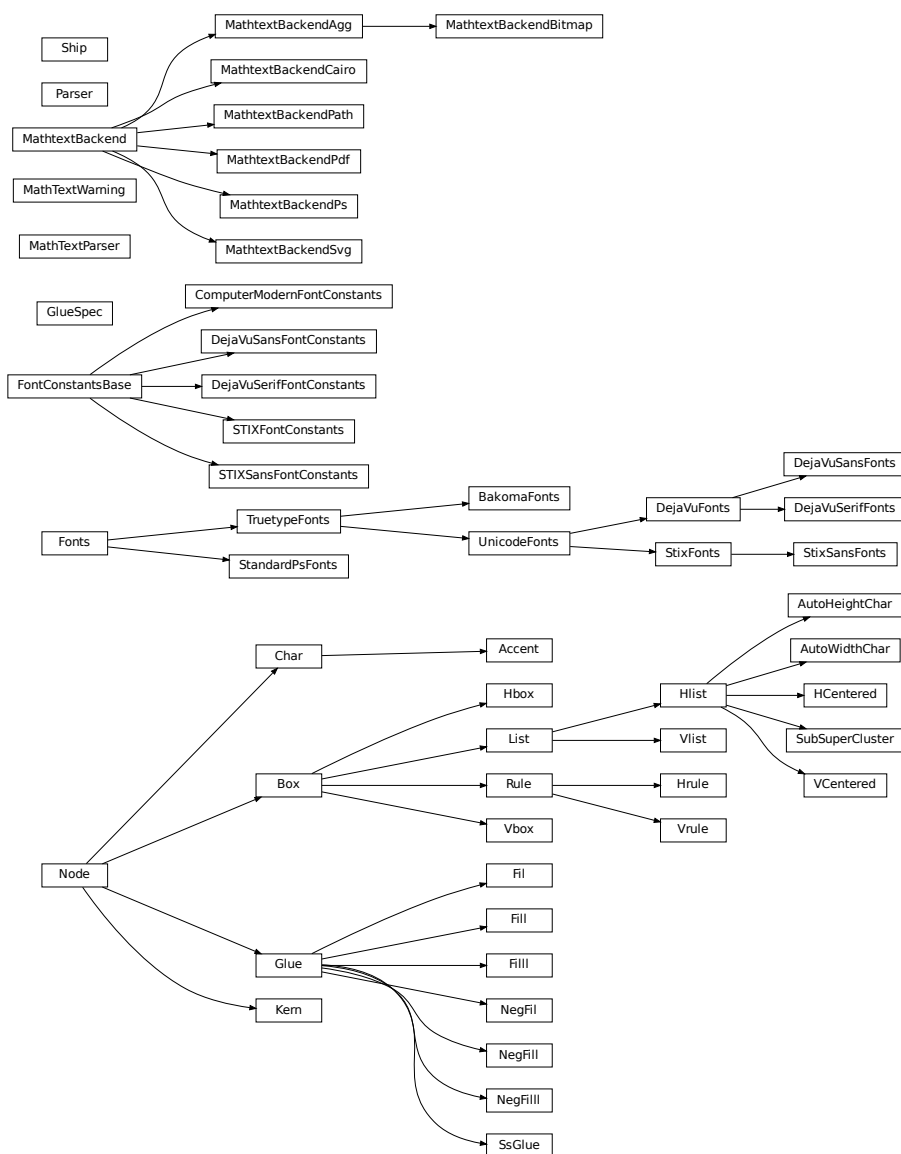
get_transform()

is_filled()

Chapter 53. markers

```
set_marker(marker)
```


MATHTEXT



54.1 matplotlib.mathtext

`mathtext` is a module for parsing a subset of the TeX math syntax and drawing them to a matplotlib backend.

For a tutorial of its usage see [Writing mathematical expressions](#). This document is primarily concerned with implementation details.

The module uses `pyarsing` to parse the TeX expression.

The Bakoma distribution of the TeX Computer Modern fonts, and STIX fonts are supported. There is experimental support for using arbitrary fonts, but results may vary without proper tweaking and metrics for those fonts.

```
class matplotlib.mathtext.Accent(c, state, math=True)
```

Bases: `matplotlib.mathtext.Char`

The font metrics need to be dealt with differently for accents, since they are already offset correctly from the baseline in TrueType fonts.

```
grow()
```

Grows one level larger. There is no limit to how big something can get.

```
render(x, y)
```

Render the character to the canvas.

```
shrink()
```

Shrinks one level smaller. There are only three levels of sizes, after which things will no longer get smaller.

```
class matplotlib.mathtext.AutoHeightChar(c, height, depth, state, always=False, factor=None)
```

Bases: `matplotlib.mathtext.Hlist`

`AutoHeightChar` will create a character as close to the given height and depth as possible. When using a font with multiple height versions of some characters (such as the BaKoMa fonts), the correct glyph will be selected, otherwise this will always just return a scaled version of the glyph.

```
class matplotlib.mathtext.AutoWidthChar(c, width, state, always=False, char_class=<class 'matplotlib.mathtext.Char'>)
```

Bases: `matplotlib.mathtext.Hlist`

`AutoWidthChar` will create a character as close to the given width as possible. When using a font with multiple width versions of some characters (such as the BaKoMa fonts), the correct glyph will be selected, otherwise this will always just return a scaled version of the glyph.

```
class matplotlib.mathtext.BakomaFonts(*args, **kwargs)
```

Bases: `matplotlib.mathtext.TruetypeFonts`

Use the Bakoma TrueType fonts for rendering.

Symbols are strewn about a number of font files, each of which has its own proprietary 8-bit encoding.


```
alias = '\\|'
```

```
get_sized_alternatives_for_symbol(fontname, sym)
```

Override if your font provides multiple sizes of the same symbol. Should return a list of symbols matching sym in various sizes. The expression renderer will select the most appropriate size for a given situation from this list.

```
target = '|'
```

```
class matplotlib.mathtext.Box(width, height, depth)
```

Bases: matplotlib.mathtext.Node

Represents any node with a physical location.

```
grow()
```

Grows one level larger. There is no limit to how big something can get.

```
render(x1, y1, x2, y2)
```

```
shrink()
```

Shrinks one level smaller. There are only three levels of sizes, after which things will no longer get smaller.

```
class matplotlib.mathtext.Char(c, state, math=True)
```

Bases: matplotlib.mathtext.Node

Represents a single character. Unlike TeX, the font information and metrics are stored with each Char to make it easier to lookup the font metrics when needed. Note that TeX boxes have a width, height, and depth, unlike Type1 and Truetype which use a full bounding box and an advance in the x-direction. The metrics must be converted to the TeX way, and the advance (if different from width) must be converted into a Kern node when the Char is added to its parent Hlist.

```
get_kerning(next)
```

Return the amount of kerning between this and the given character. Called when characters are strung together into Hlist to create Kern nodes.

```
grow()
```

Grows one level larger. There is no limit to how big something can get.

```
is_slanted()
```

```
render(x, y)
```

Render the character to the canvas

```
shrink()
```

Shrinks one level smaller. There are only three levels of sizes, after which things will no longer get smaller.

```
class matplotlib.mathtext.ComputerModernFontConstants
```

Bases: matplotlib.mathtext.FontConstantsBase

`delta = 0.075`

`delta_integral = 0.3`

`delta_slanted = 0.3`

`script_space = 0.075`

`sub1 = 0.2`

`sub2 = 0.3`

`subdrop = 0.2`

`sup1 = 0.45`

```
class matplotlib.mathtext.DejaVuFonts(*args, **kwargs)
    Bases: matplotlib.mathtext.UnicodeFonts
    use_cmex = False
```

```
class matplotlib.mathtext.DejaVuSansFontConstants
    Bases: matplotlib.mathtext.FontConstantsBase
```

```
class matplotlib.mathtext.DejaVuSansFonts(*args, **kwargs)
    Bases: matplotlib.mathtext.DejaVuFonts

    A font handling class for the DejaVu Sans fonts

    If a glyph is not found it will fallback to Stix Sans
```

```
class matplotlib.mathtext.DejaVuSerifFontConstants
    Bases: matplotlib.mathtext.FontConstantsBase
```

```
class matplotlib.mathtext.DejaVuSerifFonts(*args, **kwargs)
    Bases: matplotlib.mathtext.DejaVuFonts

    A font handling class for the DejaVu Serif fonts

    If a glyph is not found it will fallback to Stix Serif
```

```
matplotlib.mathtext.Error(msg)
    Helper class to raise parser errors.
```

```
class matplotlib.mathtext.Fil
    Bases: matplotlib.mathtext.Glue
```

```
class matplotlib.mathtext.Fill
    Bases: matplotlib.mathtext.Glue
```

```
class matplotlib.mathtext.Filll
```

```
    Bases: matplotlib.mathtext.Glue
```

```
class matplotlib.mathtext.FontConstantsBase
```

```
    Bases: object
```

A set of constants that controls how certain things, such as sub- and superscripts are laid out. These are all metrics that can't be reliably retrieved from the font metrics in the font itself.

```
delta = 0.025
```

```
delta_integral = 0.1
```

```
delta_slanted = 0.2
```

```
script_space = 0.05
```

```
sub1 = 0.3
```

```
sub2 = 0.5
```

```
subdrop = 0.4
```

```
sup1 = 0.7
```

```
class matplotlib.mathtext.Fonts(default_font_prop, mathtext_backend)
```

```
    Bases: object
```

An abstract base class for a system of fonts to use for mathtext.

The class must be able to take symbol keys and font file names and return the character metrics. It also delegates to a backend class to do the actual drawing.

`default_font_prop`: A `FontProperties` object to use for the default non-math font, or the base font for Unicode (generic) font rendering.

`mathtext_backend`: A subclass of `MathTextBackend` used to delegate the actual rendering.

```
destroy()
```

Fix any cyclical references before the object is about to be destroyed.

```
get_kern(font1, fontclass1, sym1, fontsize1, font2, fontclass2, sym2, fontsize2, dpi)
```

Get the kerning distance for font between `sym1` and `sym2`.

`fontX`: one of the TeX font names:

tt, it, rm, cal, sf, bf or default/regular (non-math)

fontclassX: TODO

symX: a symbol in raw TeX form. e.g., '1', 'x' or 'sigma'

fontsizeX: the fontsize in points

dpi: the current dots-per-inch

get_metrics(font, font_class, sym, fontsize, dpi, math=True)

font: one of the TeX font names:

tt, it, rm, cal, sf, bf or default/regular (non-math)

font_class: TODO

sym: a symbol in raw TeX form. e.g., '1', 'x' or 'sigma'

fontsize: font size in points

dpi: current dots-per-inch

math: whether sym is a math character

Returns an object with the following attributes:

- advance: The advance distance (in points) of the glyph.
- height: The height of the glyph in points.
- width: The width of the glyph in points.
- xmin, xmax, ymin, ymax - the ink rectangle of the glyph
- iceberg - the distance from the baseline to the top of the glyph. This corresponds to TeX's definition of "height".

get_results(box)

Get the data needed by the backend to render the math expression. The return value is backend-specific.

get_sized_alternatives_for_symbol(fontname, sym)

Override if your font provides multiple sizes of the same symbol. Should return a list of symbols matching sym in various sizes. The expression renderer will select the most appropriate size for a given situation from this list.

get_underline_thickness(font, fontsize, dpi)

Get the line thickness that matches the given font. Used as a base unit for drawing lines such as in a fraction or radical.

get_used_characters()

Get the set of characters that were used in the math expression. Used by backends that need to subset fonts so they know which glyphs to include.

get_xheight(font, fontsize, dpi)

Get the xheight for the given font and fontsize.

render_glyph(ox, oy, facename, font_class, sym, fontsize, dpi)

Draw a glyph at

- `ox, oy`: position
- `facename`: One of the TeX face names
- `font_class`:
- `sym`: TeX symbol name or single character
- `fontsize`: fontsize in points
- `dpi`: The dpi to draw at.

`render_rect_filled(x1, y1, x2, y2)`

Draw a filled rectangle from $(x1, y1)$ to $(x2, y2)$.

`set_canvas_size(w, h, d)`

Set the size of the buffer used to render the math expression. Only really necessary for the bitmap backends.

`class matplotlib.mathtext.Glue(glue_type, copy=False)`

Bases: `matplotlib.mathtext.Node`

Most of the information in this object is stored in the underlying `GlueSpec` class, which is shared between multiple glue objects. (This is a memory optimization which probably doesn't matter anymore, but it's easier to stick to what TeX does.)

`grow()`

Grows one level larger. There is no limit to how big something can get.

`shrink()`

Shrinks one level smaller. There are only three levels of sizes, after which things will no longer get smaller.

`class matplotlib.mathtext.GlueSpec(width=0.0, stretch=0.0, stretch_order=0, shrink=0.0, shrink_order=0)`

Bases: `object`

See `Glue`.

`copy()`

`classmethod factory(glue_type)`

`class matplotlib.mathtext.HCentered(elements)`

Bases: `matplotlib.mathtext.Hlist`

A convenience class to create an `Hlist` whose contents are centered within its enclosing box.

`class matplotlib.mathtext.Hbox(width)`

Bases: `matplotlib.mathtext.Box`

A box with only width (zero height and depth).

`class matplotlib.mathtext.Hlist(elements, w=0.0, m='additional', do_kern=True)`

Bases: `matplotlib.mathtext.List`

A horizontal list of boxes.

`hpack(w=0.0, m='additional')`

The main duty of `hpack()` is to compute the dimensions of the resulting boxes, and to adjust the glue if one of those dimensions is pre-specified. The computed sizes normally enclose all of the material inside the new box; but some items may stick out if negative glue is used, if the box is overfull, or if a `\vbox` includes other boxes that have been shifted left.

- `w`: specifies a width
- `m`: is either `'exactly'` or `'additional'`.

Thus, `hpack(w, 'exactly')` produces a box whose width is exactly `w`, while `hpack(w, 'additional')` yields a box whose width is the natural width plus `w`. The default values produce a box with the natural width.

`kern()`

Insert Kern nodes between Char nodes to set kerning. The Char nodes themselves determine the amount of kerning they need (in `get_kerning()`), and this function just creates the linked list in the correct way.

`class matplotlib.mathtext.Hrule(state, thickness=None)`

Bases: `matplotlib.mathtext.Rule`

Convenience class to create a horizontal rule.

`class matplotlib.mathtext.Kern(width)`

Bases: `matplotlib.mathtext.Node`

A Kern node has a width field to specify a (normally negative) amount of spacing. This spacing correction appears in horizontal lists between letters like A and V when the font designer said that it looks better to move them closer together or further apart. A kern node can also appear in a vertical list, when its width denotes additional spacing in the vertical direction.

`depth = 0`

`grow()`

Grows one level larger. There is no limit to how big something can get.

`height = 0`

`shrink()`

Shrinks one level smaller. There are only three levels of sizes, after which things will no longer get smaller.

`class matplotlib.mathtext.List(elements)`

Bases: `matplotlib.mathtext.Box`

A list of nodes (either horizontal or vertical).

`grow()`

Grows one level larger. There is no limit to how big something can get.

`shrink()`

Shrinks one level smaller. There are only three levels of sizes, after which things will no longer get smaller.

`class matplotlib.mathtext.MathTextParser(output)`

Bases: `object`

Create a `MathTextParser` for the given backend output.

`get_depth(texstr, dpi=120, fontsize=14)`

Returns the offset of the baseline from the bottom of the image in pixels.

`texstr` A valid `mathtext` string, e.g., `r'IQ: \sigma_i=15$'`

`dpi` The dots-per-inch to render the text

`fontsize` The font size in points

`parse(s, dpi=72, prop=None)`

Parse the given math expression `s` at the given `dpi`. If `prop` is provided, it is a `Font-Properties` object specifying the "default" font to use in the math expression, used for all non-math text.

The results are cached, so multiple calls to `parse()` with the same expression should be fast.

`to_mask(texstr, dpi=120, fontsize=14)`

`texstr` A valid `mathtext` string, e.g., `r'IQ: \sigma_i=15$'`

`dpi` The dots-per-inch to render the text

`fontsize` The font size in points

Returns a tuple (array, depth)

- `array` is an `NxM uint8` alpha ubyte mask array of rasterized tex.
- `depth` is the offset of the baseline from the bottom of the image in pixels.

`to_png(filename, texstr, color='black', dpi=120, fontsize=14)`

Writes a tex expression to a PNG file.

Returns the offset of the baseline from the bottom of the image in pixels.

`filename` A writable filename or fileobject

`texstr` A valid `mathtext` string, e.g., `r'IQ: \sigma_i=15$'`

`color` A valid matplotlib color argument

`dpi` The dots-per-inch to render the text

`fontsize` The font size in points

Returns the offset of the baseline from the bottom of the image in pixels.

`to_rgba(texstr, color='black', dpi=120, fontsize=14)`

`texstr` A valid mathtext string, e.g., `r'IQ: $\sigma_i=15$'`

`color` Any matplotlib color argument

`dpi` The dots-per-inch to render the text

`fontsize` The font size in points

Returns a tuple (array, depth)

- array is an NxM uint8 alpha ubyte mask array of rasterized tex.
- depth is the offset of the baseline from the bottom of the image in pixels.

exception `matplotlib.mathtext.MathTextWarning`

Bases: [Warning](#)

class `matplotlib.mathtext.MathtextBackend`

Bases: [object](#)

The base class for the mathtext backend-specific code. The purpose of `MathtextBackend` subclasses is to interface between mathtext and a specific matplotlib graphics backend.

Subclasses need to override the following:

- `render_glyph()`
- `render_rect_filled()`
- `get_results()`

And optionally, if you need to use a FreeType hinting style:

- `get_hinting_type()`

`get_hinting_type()`

Get the FreeType hinting type to use with this particular backend.

`get_results(box)`

Return a backend-specific tuple to return to the backend after all processing is done.

`render_glyph(ox, oy, info)`

Draw a glyph described by info to the reference point (ox, oy).

`render_rect_filled(x1, y1, x2, y2)`

Draw a filled black rectangle from (x1, y1) to (x2, y2).

`set_canvas_size(w, h, d)`

Dimension the drawing canvas

class `matplotlib.mathtext.MathtextBackendAgg`

Bases: `matplotlib.mathtext.MathtextBackend`

Render glyphs and rectangles to an FTImage buffer, which is later transferred to the Agg image by the Agg backend.

`get_hinting_type()`

Get the FreeType hinting type to use with this particular backend.

`get_results(box, used_characters)`

Return a backend-specific tuple to return to the backend after all processing is done.

`render_glyph(ox, oy, info)`

Draw a glyph described by info to the reference point (ox, oy).

`render_rect_filled(x1, y1, x2, y2)`

Draw a filled black rectangle from (x1, y1) to (x2, y2).

`set_canvas_size(w, h, d)`

Dimension the drawing canvas

`class matplotlib.mathtext.MathtextBackendBitmap`

Bases: `matplotlib.mathtext.MathtextBackendAgg`

`get_results(box, used_characters)`

Return a backend-specific tuple to return to the backend after all processing is done.

`class matplotlib.mathtext.MathtextBackendCairo`

Bases: `matplotlib.mathtext.MathtextBackend`

Store information to write a mathtext rendering to the Cairo backend.

`get_results(box, used_characters)`

Return a backend-specific tuple to return to the backend after all processing is done.

`render_glyph(ox, oy, info)`

Draw a glyph described by info to the reference point (ox, oy).

`render_rect_filled(x1, y1, x2, y2)`

Draw a filled black rectangle from (x1, y1) to (x2, y2).

`class matplotlib.mathtext.MathtextBackendPath`

Bases: `matplotlib.mathtext.MathtextBackend`

Store information to write a mathtext rendering to the text path machinery.

`get_results(box, used_characters)`

Return a backend-specific tuple to return to the backend after all processing is done.

`render_glyph(ox, oy, info)`

Draw a glyph described by info to the reference point (ox, oy).

`render_rect_filled(x1, y1, x2, y2)`

Draw a filled black rectangle from (x1, y1) to (x2, y2).

`class matplotlib.mathtext.MathtextBackendPdf`

Bases: `matplotlib.mathtext.MathtextBackend`

Store information to write a mathtext rendering to the PDF backend.

`get_results(box, used_characters)`

Return a backend-specific tuple to return to the backend after all processing is done.

`render_glyph(ox, oy, info)`

Draw a glyph described by info to the reference point (ox, oy).

```

render_rect_filled(x1, y1, x2, y2)
    Draw a filled black rectangle from (x1, y1) to (x2, y2).

class matplotlib.mathtext.MathtextBackendPs
    Bases: matplotlib.mathtext.MathtextBackend

    Store information to write a mathtext rendering to the PostScript backend.

    get_results(box, used_characters)
        Return a backend-specific tuple to return to the backend after all processing is done.

    render_glyph(ox, oy, info)
        Draw a glyph described by info to the reference point (ox, oy).

    render_rect_filled(x1, y1, x2, y2)
        Draw a filled black rectangle from (x1, y1) to (x2, y2).

class matplotlib.mathtext.MathtextBackendSvg
    Bases: matplotlib.mathtext.MathtextBackend

    Store information to write a mathtext rendering to the SVG backend.

    get_results(box, used_characters)
        Return a backend-specific tuple to return to the backend after all processing is done.

    render_glyph(ox, oy, info)
        Draw a glyph described by info to the reference point (ox, oy).

    render_rect_filled(x1, y1, x2, y2)
        Draw a filled black rectangle from (x1, y1) to (x2, y2).

class matplotlib.mathtext.NegFil
    Bases: matplotlib.mathtext.Glue

class matplotlib.mathtext.NegFill
    Bases: matplotlib.mathtext.Glue

class matplotlib.mathtext.NegFilll
    Bases: matplotlib.mathtext.Glue

class matplotlib.mathtext.Node
    Bases: object

    A node in the TeX box model

    get_kerning(next)

    grow()
        Grows one level larger. There is no limit to how big something can get.

    render(x, y)

    shrink()
        Shrinks one level smaller. There are only three levels of sizes, after which things will no
        longer get smaller.

```

class matplotlib.mathtext.Parser

Bases: [object](#)

This is the pyparsing-based parser for math expressions. It actually parses full strings containing math expressions, in that raw text may also appear outside of pairs of \$.

The grammar is based directly on that in TeX, though it cuts a few corners.

class State(font_output, font, font_class, fontsize, dpi)

Bases: [object](#)

Stores the state of the parser.

States are pushed and popped from a stack as necessary, and the "current" state is always at the top of the stack.

copy()

font

accent(s, loc, toks)

auto_delim(s, loc, toks)

binom(s, loc, toks)

c_over_c(s, loc, toks)

customspace(s, loc, toks)

dfrac(s, loc, toks)

end_group(s, loc, toks)

font(s, loc, toks)

frac(s, loc, toks)

function(s, loc, toks)

genfrac(s, loc, toks)

get_state()

Get the current State of the parser.

`group(s, loc, toks)`

`is_between_brackets(s, loc)`

`is_dropsup(nucleus)`

`is_overunder(nucleus)`

`is_slanted(nucleus)`

`main(s, loc, toks)`

`math(s, loc, toks)`

`math_string(s, loc, toks)`

`non_math(s, loc, toks)`

`operatorname(s, loc, toks)`

`overline(s, loc, toks)`

`parse(s, fonts_object, fontsize, dpi)`

Parse expression `s` using the given `fonts_object` for output, at the given `fontsize` and `dpi`.

Returns the parse tree of Node instances.

`pop_state()`

Pop a State off of the stack.

`push_state()`

Push a new State onto the stack which is just a copy of the current state.

`required_group(s, loc, toks)`

`simple_group(s, loc, toks)`

`snowflake(s, loc, toks)`

`space(s, loc, toks)`

`sqrt(s, loc, toks)`

`stackrel(s, loc, toks)`

`start__group(s, loc, toks)`

`subsuper(s, loc, toks)`

`symbol(s, loc, toks)`

`unknown__symbol(s, loc, toks)`

`class matplotlib.mathtext.Rule(width, height, depth, state)`

Bases: `matplotlib.mathtext.Box`

A Rule node stands for a solid black rectangle; it has width, depth, and height fields just as in an Hlist. However, if any of these dimensions is `inf`, the actual value will be determined by running the rule up to the boundary of the innermost enclosing box. This is called a "running dimension." The width is never running in an Hlist; the height and depth are never running in a Vlist.

`render(x, y, w, h)`

`class matplotlib.mathtext.STIXFontConstants`

Bases: `matplotlib.mathtext.FontConstantsBase`

`delta = 0.05`

`delta__integral = 0.3`

`delta__slanted = 0.3`

`script__space = 0.1`

`sub2 = 0.6`

`sup1 = 0.8`

`class matplotlib.mathtext.STIXSansFontConstants`

Bases: `matplotlib.mathtext.FontConstantsBase`

`delta__integral = 0.3`

```
delta_slanted = 0.6
```

```
script_space = 0.05
```

```
sup1 = 0.8
```

```
class matplotlib.mathtext.Ship
```

```
    Bases: object
```

Once the boxes have been set up, this sends them to output. Since boxes can be inside of boxes inside of boxes, the main work of Ship is done by two mutually recursive routines, `hlist_out()` and `vlist_out()`, which traverse the Hlist nodes and Vlist nodes inside of horizontal and vertical boxes. The global variables used in TeX to store state as it processes have become member variables here.

```
    static clamp(value)
```

```
    hlist_out(box)
```

```
    vlist_out(box)
```

```
class matplotlib.mathtext.SsGlue
```

```
    Bases: matplotlib.mathtext.Glue
```

```
class matplotlib.mathtext.StandardPsFonts(default_font_prop)
```

```
    Bases: matplotlib.mathtext.Fonts
```

Use the standard postscript fonts for rendering to backend_ps

Unlike the other font classes, `BakomaFont` and `UnicodeFont`, this one requires the Ps backend.

```
    basepath = '/home/tcaswell/.virtualenvs/sys37/lib/python3.7/site-packages/matplotlib/mpl-data/fonts/af
```

```
    fontmap = {'cal': 'pzcmi8a', 'rm': 'pncr8a', 'tt': 'pcrr8a', 'it': 'pncri8a', 'sf': 'phvr8a', 'bf': 'pncb8a', None:
```

```
    get_kern(font1, fontclass1, sym1, fontsize1, font2, fontclass2, sym2, fontsize2, dpi)
```

Get the kerning distance for font between sym1 and sym2.

fontX: one of the TeX font names:

```
tt, it, rm, cal, sf, bf or default/regular (non-math)
```

fontclassX: TODO

symX: a symbol in raw TeX form. e.g., '1', 'x' or 'sigma'

fontsizeX: the fontsize in points

dpi: the current dots-per-inch

`get_underline_thickness(font, fontsize, dpi)`

Get the line thickness that matches the given font. Used as a base unit for drawing lines such as in a fraction or radical.

`get_xheight(font, fontsize, dpi)`

Get the xheight for the given font and fontsize.

`class matplotlib.mathtext.StixFonts(*args, **kwargs)`

Bases: `matplotlib.mathtext.UnicodeFonts`

A font handling class for the STIX fonts.

In addition to what `UnicodeFonts` provides, this class:

- supports "virtual fonts" which are complete alpha numeric character sets with different font styles at special Unicode code points, such as "Blackboard".
- handles sized alternative characters for the `STIXSizeX` fonts.

`cm_fallback = False`

`get_sized_alternatives_for_symbol(fontname, sym)`

Override if your font provides multiple sizes of the same symbol. Should return a list of symbols matching `sym` in various sizes. The expression renderer will select the most appropriate size for a given situation from this list.

`use_cmex = False`

`class matplotlib.mathtext.StixSansFonts(*args, **kwargs)`

Bases: `matplotlib.mathtext.StixFonts`

A font handling class for the STIX fonts (that uses sans-serif characters by default).

`class matplotlib.mathtext.SubSuperCluster`

Bases: `matplotlib.mathtext.Hlist`

`SubSuperCluster` is a sort of hack to get around that fact that this code do a two-pass parse like TeX. This lets us store enough information in the `hlist` itself, namely the nucleus, sub- and super-script, such that if another script follows that needs to be attached, it can be reconfigured on the fly.

`class matplotlib.mathtext.TruetypeFonts(default_font_prop, mathtext_backend)`

Bases: `matplotlib.mathtext.Fonts`

A generic base class for all font setups that use Truetype fonts (through `FT2Font`).

`destroy()`

Fix any cyclical references before the object is about to be destroyed.

`get_kern(font1, fontclass1, sym1, fontsize1, font2, fontclass2, sym2, fontsize2, dpi)`

Get the kerning distance for font between `sym1` and `sym2`.

`fontX`: one of the TeX font names:

tt, it, rm, cal, sf, bf or default/regular (non-math)

fontclassX: TODO

symX: a symbol in raw TeX form. e.g., '1', 'x' or 'sigma'

fontsizeX: the fontsize in points

dpi: the current dots-per-inch

get_underline_thickness(font, fontsize, dpi)

Get the line thickness that matches the given font. Used as a base unit for drawing lines such as in a fraction or radical.

get_xheight(fontname, fontsize, dpi)

Get the xheight for the given font and fontsize.

class matplotlib.mathtext.UnicodeFonts(*args, **kwargs)

Bases: matplotlib.mathtext.TruetypeFonts

An abstract base class for handling Unicode fonts.

While some reasonably complete Unicode fonts (such as DejaVu) may work in some situations, the only Unicode font I'm aware of with a complete set of math symbols is STIX.

This class will "fallback" on the Bakoma fonts when a required symbol can not be found in the font.

get_sized_alternatives_for_symbol(fontname, sym)

Override if your font provides multiple sizes of the same symbol. Should return a list of symbols matching sym in various sizes. The expression renderer will select the most appropriate size for a given situation from this list.

use_cmex = True

class matplotlib.mathtext.VCentered(elements)

Bases: matplotlib.mathtext.Hlist

A convenience class to create a Vlist whose contents are centered within its enclosing box.

class matplotlib.mathtext.Vbox(height, depth)

Bases: matplotlib.mathtext.Box

A box with only height (zero width).

class matplotlib.mathtext.Vlist(elements, h=0.0, m='additional')

Bases: matplotlib.mathtext.List

A vertical list of boxes.

vpack(h=0.0, m='additional', l=inf)

The main duty of vpack() is to compute the dimensions of the resulting boxes, and to adjust the glue if one of those dimensions is pre-specified.

- h: specifies a height

- `m`: is either 'exactly' or 'additional'.
- `l`: a maximum height

Thus, `vpack(h, 'exactly')` produces a box whose height is exactly `h`, while `vpack(h, 'additional')` yields a box whose height is the natural height plus `h`. The default values produce a box with the natural width.

`class matplotlib.mathtext.Vrule(state)`

Bases: `matplotlib.mathtext.Rule`

Convenience class to create a vertical rule.

`matplotlib.mathtext.get_unicode_index(symbol[, bool])` → integer

Return the integer index (from the Unicode table) of `symbol`. `symbol` can be a single unicode character, a TeX command (i.e. `r'pi'`), or a Type1 symbol name (i.e. `'phi'`). If `math` is `False`, the current symbol should be treated as a non-math symbol.

`matplotlib.mathtext.math_to_image(s, filename_or_obj, prop=None, dpi=None, format=None)`

Given a math expression, renders it in a closely-clipped bounding box to an image file.

`s` A math expression. The math portion should be enclosed in dollar signs.

`filename_or_obj` A filepath or writable file-like object to write the image data to.

`prop` If provided, a `FontProperties()` object describing the size and style of the text.

`dpi` Override the output dpi, otherwise use the default associated with the output format.

`format` The output format, e.g., `'svg'`, `'pdf'`, `'ps'` or `'png'`. If not provided, will be deduced from the filename.

`matplotlib.mathtext.unichr_safe(index)`

Return the Unicode character corresponding to the `index`, or the replacement character if this is a narrow build of Python and the requested character is outside the BMP.

55.1 matplotlib.mlab

Numerical python functions written for compatibility with MATLAB commands with the same names.

55.1.1 MATLAB compatible functions

`cohere()` Coherence (normalized cross spectral density)
`csd()` Cross spectral density using Welch's average periodogram
`detrend()` Remove the mean or best fit line from an array
`find()` Return the indices where some condition is true; `numpy.nonzero` is similar but more general.
`griddata()` Interpolate irregularly distributed data to a regular grid.
`prtile()` Find the percentiles of a sequence
`prepca()` Principal Component Analysis
`psd()` Power spectral density using Welch's average periodogram
`rk4()` A 4th order runge kutta integrator for 1D or ND systems
`specgram()` Spectrogram (spectrum over segments of time)

55.1.2 Miscellaneous functions

Functions that don't exist in MATLAB, but are useful anyway:

`cohere_pairs()` Coherence over all pairs. This is not a MATLAB function, but we compute coherence a lot in my lab, and we compute it for a lot of pairs. This function is optimized to do this efficiently by caching the direct FFTs.
`rk4()` A 4th order Runge-Kutta ODE integrator in case you ever find yourself stranded without `scipy` (and the far superior `scipy.integrate` tools)
`contiguous_regions()` Return the indices of the regions spanned by some logical mask

`cross_from_below()` Return the indices where a 1D array crosses a threshold from below

`cross_from_above()` Return the indices where a 1D array crosses a threshold from above

`complex_spectrum()` Return the complex-valued frequency spectrum of a signal

`magnitude_spectrum()` Return the magnitude of the frequency spectrum of a signal

`angle_spectrum()` Return the angle (wrapped phase) of the frequency spectrum of a signal

`phase_spectrum()` Return the phase (unwrapped angle) of the frequency spectrum of a signal

`detrend_mean()` Remove the mean from a line.

`demean()` Remove the mean from a line. This function is the same as `detrend_mean()` except for the default axis.

`detrend_linear()` Remove the best fit line from a line.

`detrend_none()` Return the original line.

`stride_windows()` Get all windows in an array in a memory-efficient manner

`stride_repeat()` Repeat an array in a memory-efficient manner

`apply_window()` Apply a window along a given axis

55.1.3 record array helper functions

A collection of helper methods for `numpyrecord` arrays

See [misc-examples-index](#)

`rec2txt()` Pretty print a record array

`rec2csv()` Store record array in CSV file

`csv2rec()` Import record array from CSV file with type inspection

`rec_append_fields()` Adds field(s)/array(s) to record array

`rec_drop_fields()` Drop fields from record array

`rec_join()` Join two record arrays on sequence of fields

`recs_join()` A simple join of multiple recarrays using a single column as a key

`rec_groupby()` Summarize data by groups (similar to SQL GROUP BY)

`rec_summarize()` Helper code to filter rec array fields into new fields

For the rec viewer functions (e.g. `rec2csv`), there are a bunch of Format objects you can pass into the functions that will do things like color negative values red, set percent formatting and scaling, etc.

Example usage:

```
r = csv2rec('somefile.csv', checkrows=0)
```

```
formatd = dict(
    weight = FormatFloat(2),
    change = FormatPercent(2),
    cost   = FormatThousands(2),
)
```

```
rec2excel(r, 'test.xls', formatd=formatd)
rec2csv(r, 'test.csv', formatd=formatd)
scroll = rec2gtk(r, formatd=formatd)
```

```
win = gtk.Window()
win.set_size_request(600,800)
win.add(scroll)
win.show_all()
gtk.main()
```

```
class matplotlib.mlab.FormatBool(**kwargs)
```

Bases: matplotlib.mlab.FormatObj

Deprecated since version 2.2: The FormatBool class was deprecated in version 2.2.

fromstr(s)

toval(x)

```
class matplotlib.mlab.FormatDate(**kwargs)
```

Bases: matplotlib.mlab.FormatObj

Deprecated since version 2.2: The FormatDate class was deprecated in version 2.2. Use date.strftime instead.

fromstr(x)

toval(x)

```
class matplotlib.mlab.FormatDatetime(**kwargs)
```

Bases: matplotlib.mlab.FormatDate

Deprecated since version 2.2: The FormatDatetime class was deprecated in version 2.2. Use datetime.strftime instead.

fromstr(x)

```
class matplotlib.mlab.FormatFloat(**kwargs)
    Bases: matplotlib.mlab.FormatFormatStr

    Deprecated since version 2.2: The FormatFloat class was deprecated in version 2.2.

    fromstr(s)

    toval(x)
```

```
class matplotlib.mlab.FormatFormatStr(**kwargs)
    Bases: matplotlib.mlab.FormatObj

    Deprecated since version 2.2: The FormatFormatStr class was deprecated in version 2.2.

    tostr(x)
```

```
class matplotlib.mlab.FormatInt(**kwargs)
    Bases: matplotlib.mlab.FormatObj

    Deprecated since version 2.2: The FormatInt class was deprecated in version 2.2.

    fromstr(s)

    tostr(x)

    toval(x)
```

```
class matplotlib.mlab.FormatMillions(**kwargs)
    Bases: matplotlib.mlab.FormatFloat

    Deprecated since version 2.2: The FormatMillions class was deprecated in version 2.2.
```

```
class matplotlib.mlab.FormatObj(**kwargs)
    Bases: object

    Deprecated since version 2.2: The FormatObj class was deprecated in version 2.2.

    fromstr(s)

    tostr(x)

    toval(x)
```

```
class matplotlib.mlab.FormatPercent(**kwargs)
    Bases: matplotlib.mlab.FormatFloat

    Deprecated since version 2.2: The FormatPercent class was deprecated in version 2.2.
```

class matplotlib.mlab.FormatString(**kwargs)

Bases: matplotlib.mlab.FormatObj

Deprecated since version 2.2: The FormatString class was deprecated in version 2.2.

tostr(x)

class matplotlib.mlab.FormatThousands(**kwargs)

Bases: matplotlib.mlab.FormatFloat

Deprecated since version 2.2: The FormatThousands class was deprecated in version 2.2.

class matplotlib.mlab.GaussianKDE(dataset, bw_method=None)

Bases: [object](#)

Representation of a kernel-density estimate using Gaussian kernels.

Parameters

dataset [array_like] Datapoints to estimate from. In case of univariate data this is a 1-D array, otherwise a 2-D array with shape (# of dims, # of data).

bw_method [str, scalar or callable, optional] The method used to calculate the estimator bandwidth. This can be 'scott', 'silverman', a scalar constant or a callable. If a scalar, this will be used directly as kde.factor. If a callable, it should take a GaussianKDE instance as only parameter and return a scalar. If None (default), 'scott' is used.

Attributes

dataset [ndarray] The dataset with which gaussian_kde was initialized.

dim [int] Number of dimensions.

num_dp [int] Number of datapoints.

factor [float] The bandwidth factor, obtained from kde.covariance_factor, with which the covariance matrix is multiplied.

covariance [ndarray] The covariance matrix of dataset, scaled by the calculated bandwidth (kde.factor).

inv_cov [ndarray] The inverse of covariance.

Methods

kde.evaluate(points)	(ndarray) Evaluate the estimated pdf on a provided set of points.
kde(points)	(ndarray) Same as kde.evaluate(points)

covariance_factor()

`evaluate(points)`

Evaluate the estimated pdf on a set of points.

Parameters

`points` [(# of dimensions, # of points)-array] Alternatively, a (# of dimensions,) vector can be passed in and treated as a single point.

Returns

`values` [(# of points,)-array] The values at each point.

Raises

`ValueError` [if the dimensionality of the input points is different] than the dimensionality of the KDE.

`scotts_factor()`

`silverman_factor()`

`class matplotlib.mlab.PCA(**kwargs)`

Bases: `object`

Deprecated since version 2.2: The PCA class was deprecated in version 2.2.

`center(x)`

center and optionally standardize the data using the mean and sigma from training set `a`

`project(x, minfrac=0.0)`

project `x` onto the principle axes, dropping any axes where fraction of variance < minfrac

`matplotlib.mlab.amap(fn, *args)`

Deprecated since version 2.2: The `amap` function was deprecated in version 2.2. Use `numpy.array(list(map(...)))` instead.

`amap(function, sequence[, sequence, ...]) -> array.`

Works like `map()`, but it returns an array. This is just a convenient shorthand for `numpy.array(map(...))`.

`matplotlib.mlab.angle_spectrum(x, Fs=None, window=None, pad_to=None, sides=None)`

Compute the angle of the frequency spectrum (wrapped phase spectrum) of `x`. Data is padded to a length of `pad_to` and the windowing function `window` is applied to the signal.

Parameters

`x` [1-D array or sequence] Array or sequence containing the data

`Fs` [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.

`window` [callable or ndarray] A function or a vector of length NFFT. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.

`sides` [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.

`pad_to` [integer] The number of points to which the data segment is padded when performing the FFT. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` equal to the length of the input signal (i.e. no padding).

Returns

`spectrum` [1-D array] The values for the angle spectrum in radians (real valued)

`freqs` [1-D array] The frequencies corresponding to the elements in `spectrum`

See also:

`complex_spectrum()` This function returns the angle value of `complex_spectrum()`.

`magnitude_spectrum()` `angle_spectrum()` returns the magnitudes of the corresponding frequencies.

`phase_spectrum()` `phase_spectrum()` returns the unwrapped version of this function.

`specgram()` `specgram()` can return the angle spectrum of segments within the signal.

`matplotlib.mlab.apply_window(x, window, axis=0, return_window=None)`

Apply the given window to the given 1D or 2D array along the given axis.

Parameters

`x` [1D or 2D array or sequence] Array or sequence containing the data.

`window` [function or array.] Either a function to generate a window or an array with length `x.shape[axis]`

`axis` [integer] The axis over which to do the repetition. Must be 0 or 1. The default is 0

`return_window` [bool] If true, also return the 1D values of the window that was applied

`matplotlib.mlab.base_repr(number, base=2, padding=0)`

Deprecated since version 2.2: The `base_repr` function was deprecated in version 2.2.

Return the representation of a number in any given base.

`matplotlib.mlab.binary_repr(number, max_length=1025)`

Deprecated since version 2.2: The `binary_repr` function was deprecated in version 2.2.

Return the binary representation of the input number as a string.

This is more efficient than using `base_repr()` with base 2.

Increase the value of `max_length` for very large numbers. Note that on 32-bit machines, `2**1023` is the largest integer power of 2 which can be converted to a Python float.

`matplotlib.mlab.bivariate_normal(X, Y, sigmax=1.0, sigmay=1.0, mux=0.0, muy=0.0, sigmaxy=0.0)`

Deprecated since version 2.2: The `bivariate_normal` function was deprecated in version 2.2.

Bivariate Gaussian distribution for equal shape X, Y.

See [bivariate normal](#) at mathworld.

`matplotlib.mlab.center_matrix(M, dim=0)`

Deprecated since version 2.2: The `center_matrix` function was deprecated in version 2.2.

Return the matrix M with each row having zero mean and unit std.

If `dim = 1` operate on columns instead of rows. (`dim` is opposite to the numpy axis kwarg.)

`matplotlib.mlab.cohere(x, y, NFFT=256, Fs=2, detrend=<function detrend_none>, window=<function window_hanning>, noverlap=0, pad_to=None, sides='default', scale_by_freq=None)`

The coherence between x and y. Coherence is the normalized cross spectral density:

$$C_{xy} = \frac{|P_{xy}|^2}{P_{xx}P_{yy}}$$

Parameters

`x, y` Array or sequence containing the data

`Fs` [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.

`window` [callable or ndarray] A function or a vector of length `NFFT`. To create window vectors see `window_hanning()`, `window_none()`, [numpy.blackman\(\)](#), [numpy.hamming\(\)](#), [numpy.bartlett\(\)](#), [scipy.signal\(\)](#), [scipy.signal.get_window\(\)](#), etc. The default is `window_hanning()`. If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.

`sides` [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.

`pad_to` [integer] The number of points to which the data segment is padded when performing the FFT. This can be different from `NFFT`, which specifies the number of data points used. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` equal to `NFFT`

`NFFT` [integer] The number of data points used in each block for the FFT. A power 2 is most efficient. The default value is 256. This should NOT be used to get zero padding, or the scaling of the result will be incorrect. Use `pad_to` for this instead.

`detrend` [{`'default'`, `'constant'`, `'mean'`, `'linear'`, `'none'`} or callable] The function applied to each segment before `fft`-ing, designed to remove the mean or linear trend. Unlike in MATLAB, where the `detrend` parameter is a vector, in matplotlib it is a function. The `pylab` module defines `detrend_none()`, `detrend_mean()`, and `detrend_linear()`, but you can use a custom function as well. You can also use a string to choose one of the functions. `'default'`, `'constant'`, and `'mean'` call `detrend_mean()`. `'linear'` calls `detrend_linear()`. `'none'` calls `detrend_none()`.

`scale_by_freq` [boolean, optional] Specifies whether the resulting density values should be scaled by the scaling frequency, which gives density in units of Hz^{-1} . This allows for integration over the returned frequency values. The default is `True` for MATLAB compatibility.

`noverlap` [integer] The number of points of overlap between blocks. The default value is 0 (no overlap).

Returns

The return value is the tuple `(*Cxy*, *f*)`, where `*f*` are the frequencies of the coherence vector. For `cohere`, scaling the individual densities by the sampling frequency has no effect, since the factors cancel out.

See also:

`psd()`, `csd()`

```
matplotlib.mlab.cohere_pairs(X, ij, NFFT=256, Fs=2, detrend=<function detrend_none>,
                             window=<function window_hanning>, noverlap=0, prefer-
                             SpeedOverMemory=True, progressCallback=<function
                             donothing_callback>, returnPxx=False)
```

Deprecated since version 2.2: `scipy.signal.coherence`

Compute the coherence and phase for all pairs `ij`, in `X`.

`X` is a `numSamples * numCols` array

ij is a list of tuples. Each tuple is a pair of indexes into the columns of X for which you want to compute coherence. For example, if X has 64 columns, and you want to compute all nonredundant pairs, define ij as:

```
ij = []
for i in range(64):
    for j in range(i+1,64):
        ij.append( (i,j) )
```

preferSpeedOverMemory is an optional bool. Defaults to true. If False, limits the caching by only making one, rather than two, complex cache arrays. This is useful if memory becomes critical. Even when preferSpeedOverMemory is False, cohere_pairs() will still give significant performance gains over calling cohere() for each pair, and will use substantially less memory than if preferSpeedOverMemory is True. In my tests with a 43000,64 array over all nonredundant pairs, preferSpeedOverMemory = True delivered a 33% performance boost on a 1.7GHZ Athlon with 512MB RAM compared with preferSpeedOverMemory = False. But both solutions were more than 10x faster than naively crunching all possible pairs through cohere().

Returns

Cxy [dictionary of (i, j) tuples -> coherence vector for] that pair. i.e., Cxy[(i,j)] = cohere(X[:,i], X[:,j]). Number of dictionary keys is len(ij).

Phase [dictionary of phases of the cross spectral density at] each frequency for each pair. Keys are (i, j).

freqs [vector of frequencies, equal in length to either the] coherence or phase vectors for any (i, j) key.

e.g., to make a coherence Bode plot:: subplot(211) plot(freqs, Cxy[(12,19)])
subplot(212) plot(freqs, Phase[(12,19)])

For a large number of pairs, :func:'cohere_pairs' can be much more

efficient than just calling :func:'cohere' for each pair, because

it caches most of the intensive computations. If :math:'N' is the

number of pairs, this function is :math:'O(N)' for most of the

heavy lifting, whereas calling cohere for each pair is

:math:'O(N^2)'. However, because of the caching, it is also more

memory intensive, making 2 additional complex arrays with

approximately the same number of elements as *X*.

See :file:'test/cohere_pairs_test.py' in the src tree for an

example script that shows that this :func:'cohere_pairs' and

:func:'cohere' give the same results for a given pair.

See also:

`psd()` For information about the methods used to compute P_{xy} , P_{xx} and P_{yy} .

`matplotlib.mlab.complex_spectrum(x, Fs=None, window=None, pad_to=None, sides=None)`

Compute the complex-valued frequency spectrum of `x`. Data is padded to a length of `pad_to` and the windowing function `window` is applied to the signal.

Parameters

`x` [1-D array or sequence] Array or sequence containing the data

`Fs` [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.

`window` [callable or ndarray] A function or a vector of length `NFFT`. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.

`sides` [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.

`pad_to` [integer] The number of points to which the data segment is padded when performing the FFT. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` equal to the length of the input signal (i.e. no padding).

Returns

`spectrum` [1-D array] The values for the complex spectrum (complex valued)

`freqs` [1-D array] The frequencies corresponding to the elements in `spectrum`

See also:

`magnitude_spectrum()` `magnitude_spectrum()` returns the absolute value of this function.

`angle_spectrum()` `angle_spectrum()` returns the angle of this function.

`phase_spectrum()` `phase_spectrum()` returns the phase (unwrapped angle) of this function.

`specgram()` `specgram()` can return the complex spectrum of segments within the signal.

`matplotlib.mlab.contiguous_regions(mask)`

Deprecated since version 2.2: Moved to `matplotlib.cbook`

return a list of (`ind0`, `ind1`) such that `mask[ind0:ind1].all()` is `True` and we cover all such regions

matplotlib.mlab.cross_from_above(x, threshold)

Deprecated since version 2.2: The cross_from_above function was deprecated in version 2.2.

return the indices into x where x crosses some threshold from below, e.g., the i's where:

```
x[i-1]>threshold and x[i]<=threshold
```

See also:

cross_from_below()

matplotlib.mlab.cross_from_below(x, threshold)

Deprecated since version 2.2: The cross_from_below function was deprecated in version 2.2.

return the indices into x where x crosses some threshold from below, e.g., the i's where:

```
x[i-1]<threshold and x[i]>=threshold
```

Example code:

```
import matplotlib.pyplot as plt

t = np.arange(0.0, 2.0, 0.1)
s = np.sin(2*np.pi*t)

fig, ax = plt.subplots()
ax.plot(t, s, '-o')
ax.axhline(0.5)
ax.axhline(-0.5)

ind = cross_from_below(s, 0.5)
ax.vlines(t[ind], -1, 1)

ind = cross_from_above(s, -0.5)
ax.vlines(t[ind], -1, 1)

plt.show()
```

See also:

cross_from_above()

matplotlib.mlab.csd(x, y, NFFT=None, Fs=None, detrend=None, window=None, noverlap=None, pad_to=None, sides=None, scale_by_freq=None)

Compute the cross-spectral density.

Call signature:

```
csd(x, y, NFFT=256, Fs=2, detrend=mlab.detrend_none,
    window=mlab.window_hanning, noverlap=0, pad_to=None,
    sides='default', scale_by_freq=None)
```

The cross spectral density P_{xy} by Welch's average periodogram method. The vectors x and y are divided into NFFT length segments. Each segment is detrended by function `detrend` and windowed by function `window`. `noverlap` gives the length of the overlap between segments. The product of the direct FFTs of x and y are averaged over each segment to compute P_{xy} , with a scaling to correct for power loss due to windowing.

If `len(x) < NFFT` or `len(y) < NFFT`, they will be zero padded to NFFT.

Parameters

- `x, y` [1-D arrays or sequences] Arrays or sequences containing the data
- `Fs` [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.
- `window` [callable or ndarray] A function or a vector of length NFFT. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.
- `sides` [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.
- `pad_to` [integer] The number of points to which the data segment is padded when performing the FFT. This can be different from NFFT, which specifies the number of data points used. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is None, which sets `pad_to` equal to NFFT
- `NFFT` [integer] The number of data points used in each block for the FFT. A power 2 is most efficient. The default value is 256. This should NOT be used to get zero padding, or the scaling of the result will be incorrect. Use `pad_to` for this instead.
- `detrend` [{'default', 'constant', 'mean', 'linear', 'none'} or callable] The function applied to each segment before fft-ing, designed to remove the mean or linear trend. Unlike in MATLAB, where the `detrend` parameter is a vector, in matplotlib is it a function. The `pylab` module defines `detrend_none()`, `detrend_mean()`, and `detrend_linear()`, but you can use a custom function as well. You can also use a string to choose one of the functions. 'default', 'constant', and 'mean' call `detrend_mean()`. 'linear' calls `detrend_linear()`. 'none' calls `detrend_none()`.
- `scale_by_freq` [boolean, optional] Specifies whether the resulting density values should be scaled by the scaling frequency, which gives density in units

of Hz^{-1} . This allows for integration over the returned frequency values. The default is True for MATLAB compatibility.

`noverlap` [integer] The number of points of overlap between segments. The default value is 0 (no overlap).

Returns

`Pxy` [1-D array] The values for the cross spectrum $P_{\{xy\}}$ before scaling (real valued)

`freqs` [1-D array] The frequencies corresponding to the elements in `Pxy`

See also:

`psd()` `psd()` is the equivalent to setting `y=x`.

References

Bendat & Piersol – Random Data: Analysis and Measurement Procedures, John Wiley & Sons (1986)

`matplotlib.mlab.csv2rec(fname, comments='#', skiprows=0, checkrows=0, delimiter=',',
, converterd=None, names=None, missing='', missingd=None,
use_mrecords=False, dayfirst=False, yearfirst=False)`

Deprecated since version 2.2: The `csv2rec` function was deprecated in version 2.2.

Load data from comma/space/tab delimited file in `fname` into a numpy record array and return the record array.

If `names` is None, a header row is required to automatically assign the recarray names. The headers will be lower cased, spaces will be converted to underscores, and illegal attribute name characters removed. If `names` is not None, it is a sequence of names to use for the column names. In this case, it is assumed there is no header row.

- `fname`: can be a filename or a file handle. Support for gzipped files is automatic, if the filename ends in `'.gz'`
- `comments`: the character used to indicate the start of a comment in the file, or None to switch off the removal of comments
- `skiprows`: is the number of rows from the top to skip
- `checkrows`: is the number of rows to check to validate the column data type. When set to zero all rows are validated.
- `converterd`: if not None, is a dictionary mapping column number or munged column name to a converter function.
- `names`: if not None, is a list of header names. In this case, no header will be read from the file
- `missingd` is a dictionary mapping munged column names to field values which signify that the field does not contain actual data and should be masked, e.g., `'0000-00-00'` or `'unused'`

- `missing`: a string whose value signals a missing field regardless of the column it appears in
- `use_mrecords`: if `True`, return an `mrecords.fromrecords` record array if any of the data are missing
- `dayfirst`: default is `False` so that `MM-DD-YY` has precedence over `DD-MM-YY`. See <http://labix.org/python-dateutil#head-b95ce2094d189a89f80f5ae52a05b4ab7b41af47> for further information.
- `yearfirst`: default is `False` so that `MM-DD-YY` has precedence over `YY-MM-DD`. See <http://labix.org/python-dateutil#head-b95ce2094d189a89f80f5ae52a05b4ab7b41af47> for further information.

If no rows are found, `None` is returned

`matplotlib.mlab.csvformat_factory(format)`

Deprecated since version 2.2: The `csvformat_factory` function was deprecated in version 2.2.

`matplotlib.mlab.demean(x, axis=0)`

Return `x` minus its mean along the specified axis.

Parameters

`x` [array or sequence] Array or sequence containing the data Can have any dimensionality

`axis` [integer] The axis along which to take the mean. See `numpy.mean` for a description of this argument.

See also:

`delinear()`

`denone()` `delinear()` and `denone()` are other detrend algorithms.

`detrend_mean()` This function is the same as `detrend_mean()` except for the default axis.

`matplotlib.mlab.detrend(x, key=None, axis=None)`

Return `x` with its trend removed.

Parameters

`x` [array or sequence] Array or sequence containing the data.

`key` [['default' | 'constant' | 'mean' | 'linear' | 'none'] or function] Specifies the detrend algorithm to use. 'default' is 'mean', which is the same as `detrend_mean()`. 'constant' is the same. 'linear' is the same as `detrend_linear()`. 'none' is the same as `detrend_none()`. The default is 'mean'. See the corresponding functions for more details regarding the algorithms. Can also be a function that carries out the detrend operation.

`axis` [integer] The axis along which to do the detrending.

See also:

`detrend_mean()` `detrend_mean()` implements the 'mean' algorithm.

`detrend_linear()` `detrend_linear()` implements the 'linear' algorithm.

`detrend_none()` `detrend_none()` implements the 'none' algorithm.

`matplotlib.mlab.detrend_linear(y)`

Return x minus best fit line; 'linear' detrending.

Parameters

`y` [0-D or 1-D array or sequence] Array or sequence containing the data

`axis` [integer] The axis along which to take the mean. See `numpy.mean` for a description of this argument.

See also:

`delinear()` This function is the same as `delinear()` except for the default axis.

`detrend_mean()`

`detrend_none()` `detrend_mean()` and `detrend_none()` are other detrend algorithms.

`detrend()` `detrend()` is a wrapper around all the detrend algorithms.

`matplotlib.mlab.detrend_mean(x, axis=None)`

Return x minus the mean(x).

Parameters

`x` [array or sequence] Array or sequence containing the data Can have any dimensionality

`axis` [integer] The axis along which to take the mean. See `numpy.mean` for a description of this argument.

See also:

`demean()` This function is the same as `demean()` except for the default axis.

`detrend_linear()`

`detrend_none()` `detrend_linear()` and `detrend_none()` are other detrend algorithms.

`detrend()` `detrend()` is a wrapper around all the detrend algorithms.

`matplotlib.mlab.detrend_none(x, axis=None)`

Return x: no detrending.

Parameters

`x` [any object] An object containing the data

`axis` [integer] This parameter is ignored. It is included for compatibility with `detrend_mean`

See also:

`denone()` This function is the same as `denone()` except for the default axis, which has no effect.

`detrend__mean()`

`detrend__linear()` `detrend__mean()` and `detrend__linear()` are other detrend algorithms.

`detrend()` `detrend()` is a wrapper around all the detrend algorithms.

`matplotlib.mlab.dist(x, y)`

Deprecated since version 2.2: `numpy.hypot`

Return the distance between two points.

`matplotlib.mlab.dist__point__to__segment(p, s0, s1)`

Deprecated since version 2.2: The `dist__point__to__segment` function was deprecated in version 2.2.

Get the distance of a point to a segment.

`p, s0, s1` are xy sequences

This algorithm from http://geomalgorithms.com/a02-_lines.html

`matplotlib.mlab.distances__along__curve(X)`

Deprecated since version 2.2: The `distances__along__curve` function was deprecated in version 2.2.

Computes the distance between a set of successive points in N dimensions.

Where X is an M x N array or matrix. The distances between successive rows is computed. Distance is the standard Euclidean distance.

`matplotlib.mlab.donothing__callback(*args)`

Deprecated since version 2.2: The `donothing__callback` function was deprecated in version 2.2.

`matplotlib.mlab.entropy(y, bins)`

Deprecated since version 2.2: `scipy.stats.entropy`

Return the entropy of the data in y in units of nat.

$$-\sum p_i \ln(p_i)$$

where p_i is the probability of observing y in the i^{th} bin of bins. bins can be a number of bins or a range of bins; see [numpy.histogram\(\)](#).

Compare S with analytic calculation for a Gaussian:

```
x = mu + sigma * randn(200000)
Sanalytic = 0.5 * ( 1.0 + log(2*pi*sigma**2.0) )
```

`matplotlib.mlab.exp__safe(x)`

Deprecated since version 2.2: `numpy.exp`

Compute exponentials which safely underflow to zero.

Slow, but convenient to use. Note that numpy provides proper floating point exception handling with access to the underlying hardware.

`matplotlib.mlab.fftsurr(x, detrend=<function detrend_none>, window=<function window_none>)`

Deprecated since version 2.2: The `fftsurr` function was deprecated in version 2.2.

Compute an FFT phase randomized surrogate of `x`.

`matplotlib.mlab.find(condition)`

Deprecated since version 2.2: The `find` function was deprecated in version 2.2.

Return the indices where `ravel(condition)` is true

`matplotlib.mlab.frange(xini, xfin=None, delta=None, **kw)`

Deprecated since version 2.2: `numpy.arange`

`frange([start,] stop[, step, keywords])` -> array of floats

Return a numpy ndarray containing a progression of floats. Similar to `numpy.arange()`, but defaults to a closed interval.

`frange(x0, x1)` returns `[x0, x0+1, x0+2, ..., x1]`; start defaults to 0, and the endpoint is included. This behavior is different from that of `range()` and `numpy.arange()`. This is deliberate, since `frange()` will probably be more useful for generating lists of points for function evaluation, and endpoints are often desired in this use. The usual behavior of `range()` can be obtained by setting the keyword `closed = 0`, in this case, `frange()` basically becomes `:func:numpy.arange`.

When `step` is given, it specifies the increment (or decrement). All arguments can be floating point numbers.

`frange(x0,x1,d)` returns `[x0,x0+d,x0+2d,...,xfin]` where `xfin <= x1`.

`frange()` can also be called with the keyword `npts`. This sets the number of points the list should contain (and overrides the value `step` might have been given). `numpy.arange()` doesn't offer this option.

Examples:

```
>>> frange(3)
array([ 0.,  1.,  2.,  3.])
>>> frange(3,closed=0)
array([ 0.,  1.,  2.])
>>> frange(1,6,2)
array([1, 3, 5]) or 1,3,5,7, depending on floating point vagueries
>>> frange(1,6.5,npts=5)
array([ 1. ,  2.375,  3.75 ,  5.125,  6.5  ])
```

`matplotlib.mlab.get_formatd(r, formatd=None)`

Deprecated since version 2.2: The `get_formatd` function was deprecated in version 2.2.

build a `formatd` guaranteed to have a key for every dtype name

`matplotlib.mlab.get_sparse_matrix(M, N, frac=0.1)`

Deprecated since version 2.2: The `get_sparse_matrix` function was deprecated in version

2.2.

Return a $M \times N$ sparse matrix with `frac` elements randomly filled.

`matplotlib.mlab.get_xyz_where(Z, Cond)`

Deprecated since version 2.2: The `get_xyz_where` function was deprecated in version 2.2.

`Z` and `Cond` are $M \times N$ matrices. `Z` are data and `Cond` is a boolean matrix where some condition is satisfied. Return value is `(x, y, z)` where `x` and `y` are the indices into `Z` and `z` are the values of `Z` at those indices. `x`, `y`, and `z` are 1D arrays.

`matplotlib.mlab.griddata(x, y, z, xi, yi, interp='nn')`

Deprecated since version 2.2: The `griddata` function was deprecated in version 2.2. Use `scipy.interpolate.griddata` instead.

Interpolates from a nonuniformly spaced grid to some other grid.

Fits a surface of the form $z = f(x, y)$ to the data in the (usually) nonuniformly spaced vectors `(x, y, z)`, then interpolates this surface at the points specified by `(xi, yi)` to produce `zi`.

Parameters

`x, y, z` [1d array_like] Coordinates of grid points to interpolate from.

`xi, yi` [1d or 2d array_like] Coordinates of grid points to interpolate to.

`interp` [string key from {'nn', 'linear'}] Interpolation algorithm, either 'nn' for natural neighbor, or 'linear' for linear interpolation.

Returns

2d float array Array of values interpolated at `(xi, yi)` points. Array will be masked is any of `(xi, yi)` are outside the convex hull of `(x, y)`.

Notes

If `interp` is 'nn' (the default), uses natural neighbor interpolation based on Delaunay triangulation. This option is only available if the `mpl_toolkits.natgrid` module is installed. This can be downloaded from <https://github.com/matplotlib/natgrid>. The `(xi, yi)` grid must be regular and monotonically increasing in this case.

If `interp` is 'linear', linear interpolation is used via `matplotlib.tri.LinearTriInterpolator`.

Instead of using `griddata`, more flexible functionality and other interpolation options are available using a `matplotlib.tri.Triangulation` and a `matplotlib.tri.TriInterpolator`.

`matplotlib.mlab.identity(n, rank=2, dtype='l', typecode=None)`

Deprecated since version 2.2: `numpy.identity`

Returns the identity matrix of shape `(n, n, ..., n)` (rank `r`).

For ranks higher than 2, this object is simply a multi-index Kronecker delta:

$$\text{id}[i_0, i_1, \dots, i_R] = \begin{cases} 1 & \text{if } i_0 = i_1 = \dots = i_R, \\ 0 & \text{otherwise.} \end{cases}$$

Optionally a dtype (or typecode) may be given (it defaults to 'l').

Since rank defaults to 2, this function behaves in the default case (when only *n* is given) like `numpy.identity(n)` – but surprisingly, it is much faster.

`matplotlib.mlab.inside_poly(points, verts)`

Deprecated since version 2.2: The `inside_poly` function was deprecated in version 2.2.

points is a sequence of x, y points. *verts* is a sequence of x, y vertices of a polygon.

Return value is a sequence of indices into *points* for the points that are inside the polygon.

`matplotlib.mlab.is_closed_polygon(X)`

Deprecated since version 2.2: The `is_closed_polygon` function was deprecated in version 2.2.

Tests whether first and last object in a sequence are the same. These are presumably coordinates on a polygonal curve, in which case this function tests if that curve is closed.

`matplotlib.mlab.ispower2(n)`

Deprecated since version 2.2: The `ispower2` function was deprecated in version 2.2.

Returns the log base 2 of *n* if *n* is a power of 2, zero otherwise.

Note the potential ambiguity if *n* == 1: `2**0 == 1`, interpret accordingly.

`matplotlib.mlab.isvector(X)`

Deprecated since version 2.2: The `isvector` function was deprecated in version 2.2.

Like the MATLAB function with the same name, returns True if the supplied numpy array or matrix *X* looks like a vector, meaning it has a one non-singleton axis (i.e., it can have multiple axes, but all must have length 1, except for one of them).

If you just want to see if the array has 1 axis, use `X.ndim == 1`.

`matplotlib.mlab.l1norm(a)`

Deprecated since version 2.2: The `l1norm` function was deprecated in version 2.2. Use `numpy.linalg.norm(a, ord=1)` instead.

Return the l1 norm of *a*, flattened out.

Implemented as a separate function (not a call to `norm()` for speed).

`matplotlib.mlab.l2norm(a)`

Deprecated since version 2.2: The `l2norm` function was deprecated in version 2.2. Use `numpy.linalg.norm(a, ord=2)` instead.

Return the l2 norm of *a*, flattened out.

Implemented as a separate function (not a call to `norm()` for speed).

`matplotlib.mlab.less_simple_linear_interpolation(x, y, xi, extrap=False)`

Deprecated since version 2.2: The `less_simple_linear_interpolation` function was deprecated in version 2.2. Use `numpy.interp` instead.

This function provides simple (but somewhat less so than `cbook.simple_linear_interpolation()`) linear interpolation. `simple_linear_interpolation()` will give a list of point between a start and an end, while this does true linear interpolation at an arbitrary set of points.

This is very inefficient linear interpolation meant to be used only for a small number of points in relatively non-intensive use cases. For real linear interpolation, use `scipy`.

`matplotlib.mlab.log2(x, ln2=0.6931471805599453)`

Deprecated since version 2.2: `numpy.log2`

Return the $\log(x)$ in base 2.

This is a `_slow_` function but which is guaranteed to return the correct integer value if the input is an integer exact power of 2.

`matplotlib.mlab.logspace(xmin, xmax, N)`

Deprecated since version 2.2: The `logspace` function was deprecated in version 2.2. Use `numpy.logspace` or `numpy.geomspace` instead.

Return N values logarithmically spaced between `xmin` and `xmax`.

`matplotlib.mlab.longest_contiguous_ones(x)`

Deprecated since version 2.2: The `longest_contiguous_ones` function was deprecated in version 2.2.

Return the indices of the longest stretch of contiguous ones in `x`, assuming `x` is a vector of zeros and ones. If there are two equally long stretches, pick the first.

`matplotlib.mlab.longest_ones(x)`

Deprecated since version 2.2: The `longest_ones` function was deprecated in version 2.2.

alias for `longest_contiguous_ones`

`matplotlib.mlab.magnitude_spectrum(x, Fs=None, window=None, pad_to=None, sides=None)`

Compute the magnitude (absolute value) of the frequency spectrum of `x`. Data is padded to a length of `pad_to` and the windowing function `window` is applied to the signal.

Parameters

`x` [1-D array or sequence] Array or sequence containing the data

`Fs` [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.

`window` [callable or ndarray] A function or a vector of length `NFFT`. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function

is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.

`sides` `[['default' | 'onesided' | 'twosided']]` Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.

`pad_to` `[integer]` The number of points to which the data segment is padded when performing the FFT. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` equal to the length of the input signal (i.e. no padding).

Returns

`spectrum` `[1-D array]` The values for the magnitude spectrum (real valued)

`freqs` `[1-D array]` The frequencies corresponding to the elements in `spectrum`

See also:

`psd()` `psd()` returns the power spectral density.

`complex_spectrum()` This function returns the absolute value of `complex_spectrum()`.

`angle_spectrum()` `angle_spectrum()` returns the angles of the corresponding frequencies.

`phase_spectrum()` `phase_spectrum()` returns the phase (unwrapped angle) of the corresponding frequencies.

`specgram()` `specgram()` can return the magnitude spectrum of segments within the signal.

`matplotlib.mlab.movavg(x, n)`

Deprecated since version 2.2: The `movavg` function was deprecated in version 2.2.

Compute the `len(n)` moving average of `x`.

`matplotlib.mlab.norm_flat(a, p=2)`

Deprecated since version 2.2: The `norm_flat` function was deprecated in version 2.2. Use `numpy.linalg.norm(a.flat, ord=p)` instead.

`norm(a,p=2) -> l-p norm of a.flat`

Return the `l-p` norm of `a`, considered as a flat array. This is NOT a true matrix norm, since arrays of arbitrary rank are always flattened.

`p` can be a number or the string 'Infinity' to get the L-infinity norm.

`matplotlib.mlab.normpdf(x, *args)`

Deprecated since version 2.2: `scipy.stats.norm.pdf`

Return the normal pdf evaluated at `x`; `args` provides `mu`, `sigma`

matplotlib.mlab.offset_line(y, yerr)

Deprecated since version 2.2: The offset_line function was deprecated in version 2.2.

Offsets an array y by +/- an error and returns a tuple (y - err, y + err).

The error term can be:

- A scalar. In this case, the returned tuple is obvious.
- A vector of the same length as y. The quantities y +/- err are computed component-wise.
- A tuple of length 2. In this case, yerr[0] is the error below y and yerr[1] is error above y. For example:

```
from pylab import *
x = linspace(0, 2*pi, num=100, endpoint=True)
y = sin(x)
y_minus, y_plus = mlab.offset_line(y, 0.1)
plot(x, y)
fill_between(x, ym, y2=yp)
show()
```

matplotlib.mlab.path_length(X)

Deprecated since version 2.2: The path_length function was deprecated in version 2.2.

Computes the distance travelled along a polygonal curve in N dimensions.

Where X is an M x N array or matrix. Returns an array of length M consisting of the distance along the curve at each point (i.e., the rows of X).

matplotlib.mlab.phase_spectrum(x, Fs=None, window=None, pad_to=None, sides=None)

Compute the phase of the frequency spectrum (unwrapped angle spectrum) of x. Data is padded to a length of pad_to and the windowing function window is applied to the signal.

Parameters

x [1-D array or sequence] Array or sequence containing the data

Fs [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, freqs, in cycles per time unit. The default value is 2.

window [callable or ndarray] A function or a vector of length NFFT. To create window vectors see window_hanning(), window_none(), [numpy.blackman\(\)](#), [numpy.hamming\(\)](#), [numpy.bartlett\(\)](#), [scipy.signal\(\)](#), [scipy.signal.get_window\(\)](#), etc. The default is window_hanning(). If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.

sides [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.

`pad_to` [integer] The number of points to which the data segment is padded when performing the FFT. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` equal to the length of the input signal (i.e. no padding).

Returns

`spectrum` [1-D array] The values for the phase spectrum in radians (real valued)

`freqs` [1-D array] The frequencies corresponding to the elements in `spectrum`

See also:

`complex_spectrum()` This function returns the angle value of `complex_spectrum()`.

`magnitude_spectrum()` `magnitude_spectrum()` returns the magnitudes of the corresponding frequencies.

`angle_spectrum()` `angle_spectrum()` returns the wrapped version of this function.

`specgram()` `specgram()` can return the phase spectrum of segments within the signal.

`matplotlib.mlab.poly_below(xmin, xs, ys)`

Deprecated since version 2.2: The `poly_below` function was deprecated in version 2.2.

Given a sequence of `xs` and `ys`, return the vertices of a polygon that has a horizontal base at `xmin` and an upper bound at the `ys`. `xmin` is a scalar.

Intended for use with `matplotlib.axes.Axes.fill()`, e.g.,:

```
xv, yv = poly_below(0, x, y)
ax.fill(xv, yv)
```

`matplotlib.mlab.poly_between(x, ylower, yupper)`

Deprecated since version 2.2: The `poly_between` function was deprecated in version 2.2.

Given a sequence of `x`, `ylower` and `yupper`, return the polygon that fills the regions between them. `ylower` or `yupper` can be scalar or iterable. If they are iterable, they must be equal in length to `x`.

Return value is `x`, `y` arrays for use with `matplotlib.axes.Axes.fill()`.

`matplotlib.mlab.prctile(x, p=(0.0, 25.0, 50.0, 75.0, 100.0))`

Deprecated since version 2.2: `numpy.percentile`

Return the percentiles of `x`. `p` can either be a sequence of percentile values or a scalar. If `p` is a sequence, the `i`th element of the return sequence is the `p*(i)`-th percentile of `*x`. If `p` is a scalar, the largest value of `x` less than or equal to the `p` percentage point in the sequence is returned.

`matplotlib.mlab.prctile_rank(x, p)`

Deprecated since version 2.2: The `prctile_rank` function was deprecated in version 2.2.

Return the rank for each element in `x`, return the rank `0..len(p)`. e.g., if `p = (25, 50, 75)`, the return value will be a `len(x)` array with values in `[0,1,2,3]` where 0 indicates the value is less than the 25th percentile, 1 indicates the value is \geq the 25th and $<$ 50th percentile, ... and 3 indicates the value is above the 75th percentile cutoff.

`p` is either an array of percentiles in `[0..100]` or a scalar which indicates how many quantiles of data you want ranked.

`matplotlib.mlab.psd(x, NFFT=None, Fs=None, detrend=None, window=None, noverlap=None, pad_to=None, sides=None, scale_by_freq=None)`
 Compute the power spectral density.

Call signature:

```
psd(x, NFFT=256, Fs=2, detrend=mlab.detrend_none,
    window=mlab.window_hanning, noverlap=0, pad_to=None,
    sides='default', scale_by_freq=None)
```

The power spectral density P_{xx} by Welch's average periodogram method. The vector `x` is divided into `NFFT` length segments. Each segment is detrended by function `detrend` and windowed by function `window`. `noverlap` gives the length of the overlap between segments. The $|\text{fft}(i)|^2$ of each segment i are averaged to compute P_{xx} .

If `len(x) < NFFT`, it will be zero padded to `NFFT`.

Parameters

- `x` [1-D array or sequence] Array or sequence containing the data
- `Fs` [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.
- `window` [callable or ndarray] A function or a vector of length `NFFT`. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.
- `sides` [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.
- `pad_to` [integer] The number of points to which the data segment is padded when performing the FFT. This can be different from `NFFT`, which specifies the number of data points used. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` to equal to `NFFT`

NFFT [integer] The number of data points used in each block for the FFT. A power 2 is most efficient. The default value is 256. This should NOT be used to get zero padding, or the scaling of the result will be incorrect. Use `pad_to` for this instead.

detrend [{‘default’, ‘constant’, ‘mean’, ‘linear’, ‘none’} or callable] The function applied to each segment before fft-ing, designed to remove the mean or linear trend. Unlike in MATLAB, where the detrend parameter is a vector, in matplotlib it is a function. The pylab module defines `detrend_none()`, `detrend_mean()`, and `detrend_linear()`, but you can use a custom function as well. You can also use a string to choose one of the functions. ‘default’, ‘constant’, and ‘mean’ call `detrend_mean()`. ‘linear’ calls `detrend_linear()`. ‘none’ calls `detrend_none()`.

scale_by_freq [boolean, optional] Specifies whether the resulting density values should be scaled by the scaling frequency, which gives density in units of Hz^{-1} . This allows for integration over the returned frequency values. The default is True for MATLAB compatibility.

noverlap [integer] The number of points of overlap between segments. The default value is 0 (no overlap).

Returns

Pxx [1-D array] The values for the power spectrum $P_{\{xx\}}$ (real valued)

freqs [1-D array] The frequencies corresponding to the elements in Pxx

See also:

`specgram()` `specgram()` differs in the default overlap; in not returning the mean of the segment periodograms; and in returning the times of the segments.

`magnitude_spectrum()` `magnitude_spectrum()` returns the magnitude spectrum.

`csd()` `csd()` returns the spectral density between two signals.

References

Bendat & Piersol – Random Data: Analysis and Measurement Procedures, John Wiley & Sons (1986)

`matplotlib.mlab.quad2cubic(q0x, q0y, q1x, q1y, q2x, q2y)`

Deprecated since version 2.2: The `quad2cubic` function was deprecated in version 2.2.

Converts a quadratic Bezier curve to a cubic approximation.

The inputs are the x and y coordinates of the three control points of a quadratic curve, and the output is a tuple of x and y coordinates of the four control points of the cubic curve.

`matplotlib.mlab.rec2csv(r, fname, delimiter=',', formatd=None, missing="", missingd=None, withheader=True)`

Deprecated since version 2.2: The `rec2csv` function was deprecated in version 2.2. Use `numpy.recarray.tofile` instead.

Save the data from numpy recarray `r` into a comma-/space-/tab-delimited file. The record array dtype names will be used for column headers.

`fname`: can be a filename or a file handle. Support for gzipped files is automatic, if the file-name ends in `'gz'`

`withheader`: if `withheader` is `False`, do not write the attribute names in the first row

for `formatd` type `FormatFloat`, we override the precision to store full precision floats in the CSV file

See also:

`csv2rec()` For information about `missing` and `missingd`, which can be used to fill in masked values into your CSV file.

`matplotlib.mlab.rec2txt(r, header=None, padding=3, precision=3, fields=None)`

Deprecated since version 2.2: The `rec2txt` function was deprecated in version 2.2. Use `numpy.recarray.tofile` instead.

Returns a textual representation of a record array.

Parameters

`r`: numpy recarray

`header`: list column headers

`padding`: space between each column

`precision`: number of decimal places to use for floats. Set to an integer to apply to all floats. Set to a list of integers to apply precision individually. Precision for non-floats is simply ignored.

`fields` [list] If not `None`, a list of field names to print. `fields` can be a list of strings like `['field1', 'field2']` or a single comma separated string like `'field1,field2'`

Examples

For `precision=[0,2,3]`, the output is

ID	Price	Return
ABC	12.54	0.234
XYZ	6.32	-0.076

`matplotlib.mlab.rec_append_fields(rec, names, arrs, dtypes=None)`

Deprecated since version 2.2: The `rec_append_fields` function was deprecated in version 2.2.

Return a new record array with field names populated with data from arrays in `arrs`. If appending a single field, then `names`, `arrs` and `dtypes` do not have to be lists. They can just be the values themselves.

`matplotlib.mlab.rec_drop_fields(rec, names)`

Deprecated since version 2.2: The `rec_drop_fields` function was deprecated in version 2.2.

Return a new numpy record array with fields in names dropped.

`matplotlib.mlab.rec_groupby(r, groupby, stats)`

Deprecated since version 2.2: The `rec_groupby` function was deprecated in version 2.2.

`r` is a numpy record array

`groupby` is a sequence of record array attribute names that together form the grouping key. e.g., ('date', 'productcode')

`stats` is a sequence of (attr, func, outname) tuples which will call `x = func(attr)` and assign `x` to the record array output with attribute `outname`. For example:

```
stats = ( ('sales', len, 'numsales'), ('sales', np.mean, 'avgsale') )
```

Return record array has dtype names for each attribute name in the `groupby` argument, with the associated group values, and for each `outname` name in the `stats` argument, with the associated stat summary output.

`matplotlib.mlab.rec_join(key, r1, r2, jointype='inner', defaults=None, r1postfix='1', r2postfix='2')`

Deprecated since version 2.2: The `rec_join` function was deprecated in version 2.2.

Join record arrays `r1` and `r2` on `key`; `key` is a tuple of field names – if `key` is a string it is assumed to be a single attribute name. If `r1` and `r2` have equal values on all the keys in the `key` tuple, then their fields will be merged into a new record array containing the intersection of the fields of `r1` and `r2`.

`r1` (also `r2`) must not have any duplicate keys.

The `jointype` keyword can be 'inner', 'outer', 'leftouter'. To do a rightouter join just reverse `r1` and `r2`.

The `defaults` keyword is a dictionary filled with {column_name:default_value} pairs.

The keywords `r1postfix` and `r2postfix` are postfixed to column names (other than keys) that are both in `r1` and `r2`.

`matplotlib.mlab.rec_keep_fields(rec, names)`

Deprecated since version 2.2: The `rec_keep_fields` function was deprecated in version 2.2.

Return a new numpy record array with only fields listed in names

`matplotlib.mlab.rec_summarize(r, summaryfuncs)`

Deprecated since version 2.2: The `rec_summarize` function was deprecated in version 2.2.

`r` is a numpy record array

`summaryfuncs` is a list of (attr, func, outname) tuples which will apply `func` to the array `r[attr]` and assign the output to a new attribute name `*outname`. The returned record array is identical to `r`, with extra arrays for each element in `summaryfuncs`.

`matplotlib.mlab.recs_join(key, name, recs, jointype='outer', missing=0.0, postfixes=None)`

Deprecated since version 2.2: The `recs_join` function was deprecated in version 2.2.

Join a sequence of record arrays on single column key.

This function only joins a single column of the multiple record arrays

`key` is the column name that acts as a key

`name` is the name of the column that we want to join

`recs` is a list of record arrays to join

`jointype` is a string 'inner' or 'outer'

`missing` is what any missing field is replaced by

`postfixes` if not None, a len recs sequence of postfixes

returns a record array with columns [rowkey, name0, name1, ... namen-1]. or if postfixes [PF0, PF1, ..., PFN-1] are supplied, [rowkey, namePF0, namePF1, ... namePFN-1].

Example:

```
r = recs_join("date", "close", recs=[r0, r1], missing=0.)
```

`matplotlib.mlab.rk4(derivs, y0, t)`

Deprecated since version 2.2: `scipy.integrate.ode`

Integrate 1D or ND system of ODEs using 4-th order Runge-Kutta. This is a toy implementation which may be useful if you find yourself stranded on a system w/o `scipy`. Otherwise use `scipy.integrate()`.

Parameters

`y0` initial state vector

`t` sample times

`derivs` returns the derivative of the system and has the signature `dy = derivs(yi, ti)`

Examples

A 2D system:

```
def derivs6(x,t):
    d1 = x[0] + 2*x[1]
    d2 = -3*x[0] + 4*x[1]
    return (d1, d2)
dt = 0.0005
t = arange(0.0, 2.0, dt)
y0 = (1,2)
yout = rk4(derivs6, y0, t)
```

A 1D system:

```
alpha = 2
def derivs(x,t):
    return -alpha*x + exp(-t)

y0 = 1
yout = rk4(derivs, y0, t)
```

If you have access to `scipy`, you should probably be using the `scipy.integrate` tools rather than this function.

`matplotlib.mlab.rms_flat(a)`

Deprecated since version 2.2: The `rms_flat` function was deprecated in version 2.2.

Return the root mean square of all the elements of `a`, flattened out.

`matplotlib.mlab.safe_isinf(x)`

Deprecated since version 2.2: `numpy.isinf`

`numpy.isinf()` for arbitrary types

`matplotlib.mlab.safe_isnan(x)`

Deprecated since version 2.2: `numpy.isnan`

`numpy.isnan()` for arbitrary types

`matplotlib.mlab.segments_intersect(s1, s2)`

Deprecated since version 2.2: The `segments_intersect` function was deprecated in version 2.2.

Return True if `s1` and `s2` intersect. `s1` and `s2` are defined as:

```
s1: (x1, y1), (x2, y2)
s2: (x3, y3), (x4, y4)
```

`matplotlib.mlab.slopes(x, y)`

Deprecated since version 2.2: The `slopes` function was deprecated in version 2.2.

`slopes()` calculates the slope $y'(x)$

The slope is estimated using the slope obtained from that of a parabola through any three consecutive points.

This method should be superior to that described in the appendix of A CONSISTENTLY WELL BEHAVED METHOD OF INTERPOLATION by Russel W. Stineman (Creative Computing July 1980) in at least one aspect:

Circles for interpolation demand a known aspect ratio between `x`- and `y`-values. For many functions, however, the abscissa are given in different dimensions, so an aspect ratio is completely arbitrary.

The parabola method gives very similar results to the circle method for most regular cases but behaves much better in special cases.

Norbert Nemec, Institute of Theoretical Physics, University of Regensburg, April 2006 Norbert.Nemec at physik.uni-regensburg.de

(inspired by a original implementation by Halldor Bjornsson, Icelandic Meteorological Office, March 2006 halldor at vedur.is)

```
matplotlib.mlab.spectrogram(x, NFFT=None, Fs=None, detrend=None, window=None, nover-
                             lap=None, pad_to=None, sides=None, scale_by_freq=None,
                             mode=None)
```

Compute a spectrogram.

Compute and plot a spectrogram of data in x. Data are split into NFFT length segments and the spectrum of each section is computed. The windowing function window is applied to each segment, and the amount of overlap of each segment is specified with noverlap.

Parameters

x [array_like] 1-D array or sequence.

Fs [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, freqs, in cycles per time unit. The default value is 2.

window [callable or ndarray] A function or a vector of length NFFT. To create window vectors see window_hanning(), window_none(), [numpy.blackman\(\)](#), [numpy.hamming\(\)](#), [numpy.bartlett\(\)](#), [scipy.signal\(\)](#), [scipy.signal.get_window\(\)](#), etc. The default is window_hanning(). If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.

sides [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.

pad_to [integer] The number of points to which the data segment is padded when performing the FFT. This can be different from NFFT, which specifies the number of data points used. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the n parameter in the call to `fft()`. The default is None, which sets pad_to equal to NFFT

NFFT [integer] The number of data points used in each block for the FFT. A power 2 is most efficient. The default value is 256. This should NOT be used to get zero padding, or the scaling of the result will be incorrect. Use pad_to for this instead.

detrend [{ 'default', 'constant', 'mean', 'linear', 'none' } or callable] The function applied to each segment before `fft`-ing, designed to remove the mean or linear trend. Unlike in MATLAB, where the detrend parameter is a vector, in matplotlib it is a function. The pylab module defines `detrend_none()`, `detrend_mean()`, and `detrend_linear()`, but you can use a custom function

as well. You can also use a string to choose one of the functions. 'default', 'constant', and 'mean' call `detrend_mean()`. 'linear' calls `detrend_linear()`. 'none' calls `detrend_none()`.

`scale_by_freq` [boolean, optional] Specifies whether the resulting density values should be scaled by the scaling frequency, which gives density in units of Hz^{-1} . This allows for integration over the returned frequency values. The default is True for MATLAB compatibility.

`noverlap` [int, optional] The number of points of overlap between blocks. The default value is 128.

`mode` [str, optional]

What sort of spectrum to use, default is 'psd'.

'psd' Returns the power spectral density.

'complex' Returns the complex-valued frequency spectrum.

'magnitude' Returns the magnitude spectrum.

'angle' Returns the phase spectrum without unwrapping.

'phase' Returns the phase spectrum with unwrapping.

Returns

`spectrum` [array_like] 2-D array, columns are the periodograms of successive segments.

`freqs` [array_like] 1-D array, frequencies corresponding to the rows in spectrum.

`t` [array_like] 1-D array, the times corresponding to midpoints of segments (i.e the columns in spectrum).

See also:

`psd` differs in the overlap and in the return values.

`complex_spectrum` similar, but with complex valued frequencies.

`magnitude_spectrum` similar single segment when mode is 'magnitude'.

`angle_spectrum` similar to single segment when mode is 'angle'.

`phase_spectrum` similar to single segment when mode is 'phase'.

Notes

`detrend` and `scale_by_freq` only apply when mode is set to 'psd'.

`matplotlib.mlab.stineman_interp(xi, x, y, yp=None)`

Deprecated since version 2.2: The `stineman_interp` function was deprecated in version 2.2.

Given data vectors `x` and `y`, the slope vector `yp` and a new abscissa vector `xi`, the function `stineman_interp()` uses Stineman interpolation to calculate a vector `yi` corresponding to `xi`.

Here's an example that generates a coarse sine curve, then interpolates over a finer abscissa:

```
x = linspace(0,2*pi,20); y = sin(x); yp = cos(x)
xi = linspace(0,2*pi,40);
yi = stineman_interp(xi,x,y,yp);
plot(x,y,'o',xi,yi)
```

The interpolation method is described in the article **A CONSISTENTLY WELL BEHAVED METHOD OF INTERPOLATION** by Russell W. Stineman. The article appeared in the July 1980 issue of *Creative Computing* with a note from the editor stating that while they were:

not an academic journal but once in a while something serious and original comes in adding that this was "apparently a real solution" to a well known problem.

For `yp = None`, the routine automatically determines the slopes using the `slopes()` routine.

`x` is assumed to be sorted in increasing order.

For values `xi[j] < x[0]` or `xi[j] > x[-1]`, the routine tries an extrapolation. The relevance of the data obtained from this, of course, is questionable...

Original implementation by Halldor Bjornsson, Icelandic Meteorological Office, March 2006
halldor at vedur.is

Completely reworked and optimized for Python by Norbert Nemec, Institute of Theoretical Physics, University of Regensburg, April 2006 Norbert.Nemec at physik.uni-regensburg.de

`matplotlib.mlab.stride_repeat(x, n, axis=0)`

Repeat the values in an array in a memory-efficient manner. Array `x` is stacked vertically `n` times.

Warning: It is not safe to write to the output array. Multiple elements may point to the same piece of memory, so modifying one value may change others.

Parameters

- `x` [1D array or sequence] Array or sequence containing the data.
- `n` [integer] The number of time to repeat the array.
- `axis` [integer] The axis along which the data will run.

References

[stackoverflow: Repeat NumPy array without replicating data?](#)

`matplotlib.mlab.stride_windows(x, n, noverlap=None, axis=0)`

Get all windows of `x` with length `n` as a single array, using strides to avoid data duplication.

Warning: It is not safe to write to the output array. Multiple elements may point to the same piece of memory, so modifying one value may change others.

Parameters

- `x` [1D array or sequence] Array or sequence containing the data.
- `n` [integer] The number of data points in each window.
- `noverlap` [integer] The overlap between adjacent windows. Default is 0 (no overlap)
- `axis` [integer] The axis along which the windows will run.

References

[stackoverflow: Rolling window for 1D arrays in Numpy?](#) [stackoverflow: Using strides for an efficient moving average filter](#)

`matplotlib.mlab.vector_lengths(X, P=2.0, axis=None)`

Deprecated since version 2.2: The `vector_lengths` function was deprecated in version 2.2.

Finds the length of a set of vectors in `n` dimensions. This is like the `numpy.norm()` function for vectors, but has the ability to work over a particular axis of the supplied array or matrix.

Computes $(\sum(x_i^P))^{1/P}$ for each $\{x_i\}$ being the elements of `X` along the given axis. If `axis` is `None`, compute over all elements of `X`.

`matplotlib.mlab.window_hanning(x)`

Return `x` times the hanning window of `len(x)`.

See also:

`window_none()` `window_none()` is another window algorithm.

`matplotlib.mlab.window_none(x)`

No window function; simply return `x`.

See also:

`window_hanning()` `window_hanning()` is another window algorithm.

OFFSETBOX

56.1 matplotlib.offsetbox

The `OffsetBox` is a simple container artist. The child artist are meant to be drawn at a relative position to its parent. The `[VH]Packer`, `DrawingArea` and `TextArea` are derived from the `OffsetBox`.

The `[VH]Packer` automatically adjust the relative positions of their children, which should be instances of the `OffsetBox`. This is used to align similar artists together, e.g., in legend.

The `DrawingArea` can contain any `Artist` as a child. The `DrawingArea` has a fixed width and height. The position of children relative to the parent is fixed. The `TextArea` contains a single `Text` instance. The width and height of the `TextArea` instance is the width and height of the its child text.

```
class matplotlib.offsetbox.AnchoredOffsetbox(loc,          pad=0.4,          borderpad=0.5,
                                              child=None,      prop=None,
                                              frameon=True,    bbox_to_anchor=None,
                                              bbox_transform=None, **kwargs)
```

Bases: `matplotlib.offsetbox.OffsetBox`

An offset box placed according to the legend location `loc`. `AnchoredOffsetbox` has a single child. When multiple children is needed, use other `OffsetBox` class to enclose them. By default, the offset box is anchored against its parent axes. You may explicitly specify the `bbox_to_anchor`.

`loc` is a string or an integer specifying the legend location. The valid location codes are:

```
'upper right' : 1,
'upper left'  : 2,
'lower left'  : 3,
'lower right' : 4,
'right'       : 5, (same as 'center right', for back-compatibility)
'center left' : 6,
'center right': 7,
'lower center': 8,
'upper center': 9,
'center'      : 10,
```

`pad` [pad around the child for drawing a frame. given in] fraction of fontsize.

`borderpad` : pad between offsetbox frame and the `bbox_to_anchor`,

`child` : `OffsetBox` instance that will be anchored.

`prop` : font property. This is only used as a reference for paddings.

`frameon` : draw a frame box if True.

`bbox_to_anchor` : bbox to anchor. Use `self.axes.bbox` if None.

`bbox_transform` : with which the `bbox_to_anchor` will be transformed.

`codes` = {'center': 10, 'center left': 6, 'center right': 7, 'lower center': 8, 'lower left': 3, 'lower right': 4, 'right': 5, 'upper center': 9, 'upper left': 1, 'upper right': 2}

`draw(renderer)`

draw the artist

`get_bbox_to_anchor()`

return the bbox that the legend will be anchored

`get_child()`

return the child

`get_children()`

return the list of children

`get_extent(renderer)`

return the extent of the artist. The extent of the child added with the pad is returned

`get_window_extent(renderer)`

get the bounding box in display space.

`set_bbox_to_anchor(bbox, transform=None)`

set the bbox that the child will be anchored.

`bbox` can be a `Bbox` instance, a list of [left, bottom, width, height], or a list of [left, bottom] where the width and height will be assumed to be zero. The `bbox` will be transformed to display coordinate by the given transform.

`set_child(child)`

set the child to be anchored

`update_frame(bbox, fontsize=None)`

`zorder` = 5

`class matplotlib.offsetbox.AnchoredText(s, loc, pad=0.4, borderpad=0.5, prop=None, **kwargs)`

Bases: `matplotlib.offsetbox.AnchoredOffsetbox`

`AnchoredOffsetbox` with `Text`.

Parameters

s [string] Text.

loc [str] Location code.

pad [float, optional] Pad between the text and the frame as fraction of the font size.

borderpad [float, optional] Pad between the frame and the axes (or bbox_to_anchor).

prop [matplotlib.font_manager.FontProperties] Font properties.

Notes

Other keyword parameters of AnchoredOffsetbox are also allowed.

```
class matplotlib.offsetbox.AnnotationBbox(offsetbox, xy, xybox=None, xycoords='data',
                                          boxcoords=None, frameon=True, pad=0.4,
                                          annotation_clip=None, box_alignment=(0.5,
                                          0.5), bboxprops=None, arrowprops=None,
                                          fontsize=None, **kwargs)
```

Bases: matplotlib.artist.Artist, matplotlib.text._AnnotationBase

Annotation-like class, but with offsetbox instead of Text.

offsetbox : OffsetBox instance

xycoords [same as Annotation but can be a tuple of two] strings which are interpreted as x and y coordinates.

boxcoords [similar to textcoords as Annotation but can be a] tuple of two strings which are interpreted as x and y coordinates.

box_alignment [a tuple of two floats for a vertical and] horizontal alignment of the offset box w.r.t. the boxcoords. The lower-left corner is (0.0) and upper-right corner is (1.1).

other parameters are identical to that of Annotation.

anncoords

contains(event)

Test whether the artist contains the mouse event.

Returns the truth value and a dictionary of artist specific details of selection, such as which points are contained in the pick radius. See individual artists for details.

draw(renderer)

Draw the Annotation object to the given renderer.

get_children()

Return a list of the child Artist's this :class:`Artist` contains.

get_fontsize(s=None)

return fontsize in points

`set_figure(fig)`

Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`set_fontsize(s=None)`

set fontsize in points

`update_positions(renderer)`

Update the pixel positions of the annotated point and the text.

`xyann`

`zorder = 3`

`class matplotlib.offsetbox.AuxTransformBox(aux__transform)`

Bases: `matplotlib.offsetbox.OffsetBox`

Offset Box with the `aux__transform` . Its children will be transformed with the `aux__transform` first then will be offseted. The absolute coordinate of the `aux__transform` is meaning as it will be automatically adjust so that the left-lower corner of the bounding box of children will be set to (0,0) before the offset transform.

It is similar to drawing area, except that the extent of the box is not predetermined but calculated from the window extent of its children. Furthermore, the extent of the children will be calculated in the transformed coordinate.

`add_artist(a)`

Add any Artist to the container box

`draw(renderer)`

Draw the children

`get_extent(renderer)`

Return with, height, xdescent, ydescent of box

`get_offset()`

return offset of the container.

`get_transform()`

Return the Transform applied to the children

`get_window_extent(renderer)`

get the bounding box in display space.

`set_offset(xy)`

set offset of the container.

Accept : tuple of x,y coordinate in display units.

`set_transform(t)`

set_transform is ignored.

```
class matplotlib.offsetbox.DraggableAnnotation(annotation, use_blit=False)
```

```
    Bases: matplotlib.offsetbox.DraggableBase
```

```
    save__offset()
```

```
    update__offset(dx, dy)
```

```
class matplotlib.offsetbox.DraggableBase(ref_artist, use_blit=False)
```

```
    Bases: object
```

helper code for a draggable artist (legend, offsetbox) The derived class must override following two method.

```
    def save__offset(self): pass
```

```
    def update__offset(self, dx, dy): pass
```

save__offset is called when the object is picked for dragging and it is meant to save reference position of the artist.

update__offset is called during the dragging. dx and dy is the pixel offset from the point where the mouse drag started.

Optionally you may override following two methods.

```
    def artist_picker(self, artist, evt): return self.ref_artist.contains(evt)
```

```
    def finalize__offset(self): pass
```

artist_picker is a picker method that will be used. finalize__offset is called when the mouse is released. In current implementation of DraggableLegend and DraggableAnnotation, update__offset places the artists simply in display coordinates. And finalize__offset recalculate their position in the normalized axes coordinate and set a relevant attribute.

```
artist_picker(artist, evt)
```

```
disconnect()
```

```
    disconnect the callbacks
```

```
finalize__offset()
```

```
on__motion(evt)
```

```
on__motion__blit(evt)
```

```
on__pick(evt)
```

```
on__release(event)
```


save_offset()

update_offset(dx, dy)

class matplotlib.offsetbox.DraggableOffsetBox(ref_artist, offsetbox, use_blit=False)

Bases: matplotlib.offsetbox.DraggableBase

get_loc_in_canvas()

save_offset()

update_offset(dx, dy)

class matplotlib.offsetbox.DrawingArea(width, height, xdescent=0.0, ydescent=0.0,
clip=False)

Bases: matplotlib.offsetbox.OffsetBox

The DrawingArea can contain any Artist as a child. The DrawingArea has a fixed width and height. The position of children relative to the parent is fixed. The children can be clipped at the boundaries of the parent.

width, height : width and height of the container box. xdescent, ydescent : descent of the box in x- and y-direction. clip : Whether to clip the children

add_artist(a)

Add any Artist to the container box

clip_children

If the children of this DrawingArea should be clipped by DrawingArea bounding box.

draw(renderer)

Draw the children

get_extent(renderer)

Return with, height, xdescent, ydescent of box

get_offset()

return offset of the container.

get_transform()

Return the Transform applied to the children

get_window_extent(renderer)

get the bounding box in display space.

set_offset(xy)

set offset of the container.

Accept : tuple of x,y coordinate in display units.

set_transform(t)

set_transform is ignored.

```
class matplotlib.offsetbox.HPacker(pad=None, sep=None, width=None, height=None,
                                   align='baseline', mode='fixed', children=None)
    Bases: matplotlib.offsetbox.PackerBase
```

The HPacker has its children packed horizontally. It automatically adjusts the relative positions of children at draw time.

Parameters

`pad` [float, optional] Boundary pad.

`sep` [float, optional] Spacing between items.

`width` [float, optional]

`height` [float, optional] Width and height of the container box, calculated if `None`.

`align` [str] Alignment of boxes.

`mode` [str] Packing mode.

Notes

`pad` and `sep` need to be given in points and will be scaled with the renderer dpi, while `width` and `height` need to be in pixels.

```
get_extents(renderer)
    update offset of children and return the extents of the box
```

```
class matplotlib.offsetbox.OffsetBox(*args, **kwargs)
    Bases: matplotlib.artist.Artist
```

The OffsetBox is a simple container artist. The child artists are meant to be drawn at a relative position to its parent.

`axes`

The Axes instance the artist resides in, or `None`.

```
contains(mouseevent)
    Test whether the artist contains the mouse event.
```

Returns the truth value and a dictionary of artist specific details of selection, such as which points are contained in the pick radius. See individual artists for details.

```
draw(renderer)
    Update the location of children if necessary and draw them to the given renderer.
```

```
get_children()
    Return a list of artists it contains.
```

```
get_extents(renderer)
    Return width, height, xdescent, ydescent of box
```

```
get_extents_offsets(renderer)
```

`get__offset(width, height, xdescent, ydescent, renderer)`
Get the offset

accepts extent of the box

`get__visible__children()`
Return a list of visible artists it contains.

`get__window__extent(renderer)`
get the bounding box in display space.

`set__figure(fig)`
Set the figure
accepts a class:Figure instance

`set__height(height)`
Set the height
accepts float

`set__offset(xy)`
Set the offset
accepts x, y, tuple, or a callable object.

`set__width(width)`
Set the width
accepts float

`class matplotlib.offsetbox.OffsetImage(arr, zoom=1, cmap=None, norm=None, interpolation=None, origin=None, filtnorm=1, filterrad=4.0, resample=False, dpi__cor=True, **kwargs)`

Bases: `matplotlib.offsetbox.OffsetBox`

`draw(renderer)`
Draw the children

`get__children()`
Return a list of artists it contains.

`get__data()`

`get__extent(renderer)`
Return with, height, xdescent, ydescent of box

`get__offset()`
return offset of the container.

`get__window__extent(renderer)`
get the bounding box in display space.

`get__zoom()`

`set_data(arr)`

`set_zoom(zoom)`

```
class matplotlib.offsetbox.PackerBase(pad=None, sep=None, width=None, height=None,
                                      align=None, mode=None, children=None)
```

Bases: `matplotlib.offsetbox.OffsetBox`

Parameters

`pad` [float, optional] Boundary pad.

`sep` [float, optional] Spacing between items.

`width` [float, optional]

`height` [float, optional] Width and height of the container box, calculated if `None`.

`align` [str, optional] Alignment of boxes. Can be one of top, bottom, left, right, center and baseline

`mode` [str, optional] Packing mode.

Notes

`pad` and `sep` need to be given in points and will be scaled with the renderer dpi, while `width` and `height` need to be in pixels.

```
class matplotlib.offsetbox.PaddedBox(child, pad=None, draw_frame=False,
                                     patch_attrs=None)
```

Bases: `matplotlib.offsetbox.OffsetBox`

`pad` : boundary pad

Note: `pad` need to be given in points and will be scaled with the renderer dpi, while `width` and `height` need to be in pixels.

`draw(renderer)`

Update the location of children if necessary and draw them to the given renderer.

`draw_frame(renderer)`

`get_extents(renderer)`

Update offset of childrens and return the extents of the box

`update_frame(bbox, fontsize=None)`

```
class matplotlib.offsetbox.TextArea(s, textprops=None, multilinebaseline=None, minimumdescent=True)
```

Bases: `matplotlib.offsetbox.OffsetBox`

The `TextArea` contains a single `Text` instance. The text is placed at (0,0) with baseline+left alignment. The width and height of the `TextArea` instance is the width and height of the its child text.

Parameters

`s` [`str`] a string to be displayed.

`textprops` [`FontProperties`, optional]

`multilinebaseline` [`bool`, optional] If `True`, baseline for multiline text is adjusted so that it is (approximately) center-aligned with singleline text.

`minimumdescent` [`bool`, optional] If `True`, the box has a minimum descent of "p".

`draw(renderer)`

Draw the children

`get_extent(renderer)`

Return with, height, xdescent, ydescent of box

`get_minimumdescent()`

get minimumdescent.

`get_multilinebaseline()`

get multilinebaseline .

`get_offset()`

return offset of the container.

`get_text()`

Returns the string representation of this area's text

`get_window_extent(renderer)`

get the bounding box in display space.

`set_minimumdescent(t)`

Set minimumdescent .

If `True`, extent of the single line text is adjusted so that it has minimum descent of "p"

`set_multilinebaseline(t)`

Set multilinebaseline .

If `True`, baseline for multiline text is adjusted so that it is (approximately) center-aligned with singleline text.

`set_offset(xy)`

set offset of the container.

Accept : tuple of x,y coordinates in display units.

`set_text(s)`

Set the text of this area as a string.

`set_transform(t)`

`set_transform` is ignored.

`class matplotlib.offsetbox.VPacker(pad=None, sep=None, width=None, height=None, align='baseline', mode='fixed', children=None)`

Bases: `matplotlib.offsetbox.PackerBase`

The `VPacker` has its children packed vertically. It automatically adjust the relative positions of children in the drawing time.

Parameters

`pad` [float, optional] Boundary pad.

`sep` [float, optional] Spacing between items.

`width` [float, optional]

`height` [float, optional] width and height of the container box, calculated if `None`.

`align` [str, optional] Alignment of boxes.

`mode` [str, optional] Packing mode.

Notes

`pad` and `sep` need to given in points and will be scale with the renderer dpi, while `width` and `height` need to be in pixels.

`get_extent__offsets(renderer)`

update offset of childrens and return the extents of the box

`matplotlib.offsetbox.bbox__artist(*args, **kwargs)`

PATCHES

57.1 matplotlib.patches

57.1.1 Classes

<code>Arc(xy, width, height[, angle, theta1, theta2])</code>	An elliptical arc.
<code>Arrow(x, y, dx, dy[, width])</code>	An arrow patch.
<code>ArrowStyle</code>	<code>ArrowStyle</code> is a container class which defines several arrowstyle classes, which is used to create an arrow path along a given path.
<code>BoxStyle</code>	<code>BoxStyle</code> is a container class which defines several boxstyle classes, which are used for <code>FancyBboxPatch</code> .
<code>Circle(xy[, radius])</code>	A circle patch.
<code>CirclePolygon(xy[, radius, resolution])</code>	A polygon-approximation of a circle patch.
<code>ConnectionPatch(xyA, xyB, coordsA[, ...])</code>	A <code>ConnectionPatch</code> class is to make connecting lines between two points (possibly in different axes).
<code>ConnectionStyle</code>	<code>ConnectionStyle</code> is a container class which defines several connectionstyle classes, which is used to create a path between two points.
<code>Ellipse(xy, width, height[, angle])</code>	A scale-free ellipse.
<code>FancyArrow(x, y, dx, dy[, width, ...])</code>	Like <code>Arrow</code> , but lets you set head width and head height independently.
<code>FancyArrowPatch([posA, posB, path, ...])</code>	A fancy arrow patch.
<code>FancyBboxPatch(xy, width, height[, ...])</code>	Draw a fancy box around a rectangle with lower left at <code>xy*=(*x, y)</code> with specified width and height.
<code>Patch([edgecolor, facecolor, color, ...])</code>	A patch is a 2D artist with a face color and an edge color.
<code>PathPatch(path, **kwargs)</code>	A general polycurve path patch.
<code>Polygon(xy[, closed])</code>	A general polygon patch.
<code>Rectangle(xy, width, height[, angle])</code>	Draw a rectangle with lower left at <code>xy = (x, y)</code> with specified width, height and rotation angle.

Continued on next page

Table 1 – continued from previous page

<code>RegularPolygon(xy, numVertices[, radius, ...])</code>	A regular polygon patch.
<code>Shadow(patch, ox, oy[, props])</code>	Create a shadow of the given patch offset by <code>ox</code> , <code>oy</code> .
<code>Wedge(center, r, theta1, theta2[, width])</code>	Wedge shaped patch.
<code>YAArrow(figure, xytip, xybase[, width, ...])</code>	Yet another arrow class.

`matplotlib.patches.Arc`

```
class matplotlib.patches.Arc(xy, width, height, angle=0.0, theta1=0.0, theta2=360.0,
                             **kwargs)
```

An elliptical arc. Because it performs various optimizations, it can not be filled.

The arc must be used in an Axes instance—it can not be added directly to a Figure—because it is optimized to only render the segments that are inside the axes bounding box with high resolution.

The following args are supported:

`xy` center of ellipse

`width` length of horizontal axis

`height` length of vertical axis

`angle` rotation in degrees (anti-clockwise)

`theta1` starting angle of the arc in degrees

`theta2` ending angle of the arc in degrees

If `theta1` and `theta2` are not provided, the arc will form a complete ellipse.

Valid kwargs are:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\ ' ' ' '- ' '+ ' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

draw(renderer)

Ellipses are normally drawn using an approximation that uses eight cubic bezier splines. The error of this approximation is 1.89818e-6, according to this unverified source:

Lancaster, Don. Approximating a Circle or an Ellipse Using Four Bezier Cubic Splines.

<http://www.tinaja.com/glib/ellipse4.pdf>

There is a use case where very large ellipses must be drawn with very high accuracy, and

it is too expensive to render the entire ellipse with enough segments (either splines or line segments). Therefore, in the case where either radius of the ellipse is large enough that the error of the spline approximation will be visible (greater than one pixel offset from the ideal), a different technique is used.

In that case, only the visible parts of the ellipse are drawn, with each visible arc using a fixed number of spline segments (8). The algorithm proceeds as follows:

1. The points where the ellipse intersects the axes bounding box are located. (This is done by performing an inverse transformation on the axes bbox such that it is relative to the unit circle – this makes the intersection calculation much easier than doing rotated ellipse intersection directly).

This uses the "line intersecting a circle" algorithm from:

Vince, John. Geometry for Computer Graphics: Formulae, Examples & Proofs. London: Springer-Verlag, 2005.

2. The angles of each of the intersection points are calculated.
3. Proceeding counterclockwise starting in the positive x-direction, each of the visible arc-segments between the pairs of vertices are drawn using the bezier arc approximation technique implemented in `matplotlib.path.Path.arc()`.

Examples using `matplotlib.patches.Arc`

- `sphx_glr_gallery_units_ellipse_with_units.py`

`matplotlib.patches.Arrow`

`class matplotlib.patches.Arrow(x, y, dx, dy, width=1.0, **kwargs)`

An arrow patch.

Draws an arrow from (x, y) to (x + dx, y + dy). The width of the arrow is scaled by width.

Parameters

x [scalar] x coordinate of the arrow tail

y [scalar] y coordinate of the arrow tail

dx [scalar] Arrow length in the x direction

dy [scalar] Arrow length in the y direction

width [scalar, optional (default: 1)] Scale factor for the width of the arrow.

With a default value of 1, the tail width is 0.2 and head width is 0.6.

****kwargs :** Keyword arguments control the Patch properties:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

See also:

FancyArrow Patch that allows independent control of the head and tail properties

get_patch_transform()

Return the Transform instance which takes patch coordinates to data coordinates.

For example, one may define a patch of a circle which represents a radius of 5 by providing

coordinates for a unit circle, and a transform which scales the coordinates (the patch coordinate) by 5.

`get_path()`
Return the path of this patch

Examples using `matplotlib.patches.Arrow`

- `sphx_glr_gallery_shapes_and_collections_artist_reference.py`

`matplotlib.patches.ArrowStyle`

`class matplotlib.patches.ArrowStyle`

`ArrowStyle` is a container class which defines several arrowstyle classes, which is used to create an arrow path along a given path. These are mainly used with `FancyArrowPatch`.

A arrowstyle object can be either created as:

```
ArrowStyle.Fancy(head_length=.4, head_width=.4, tail_width=.4)
```

or:

```
ArrowStyle("Fancy", head_length=.4, head_width=.4, tail_width=.4)
```

or:

```
ArrowStyle("Fancy", head_length=.4, head_width=.4, tail_width=.4)
```

The following classes are defined

Class	Name	Attrs
Curve	-	None
CurveB	->	head_length=0.4,head_width=0.2
BracketB	-[widthB=1.0,lengthB=0.2,angleB=None
Curve-FilledB	- >	head_length=0.4,head_width=0.2
CurveA	<-	head_length=0.4,head_width=0.2
CurveAB	<->	head_length=0.4,head_width=0.2
Curve-FilledA	< -	head_length=0.4,head_width=0.2
Curve-FilledAB	< ->	head_length=0.4,head_width=0.2
BracketA]-	widthA=1.0,lengthA=0.2,angleA=None
BracketAB]-[widthA=1.0,lengthA=0.2,angleA=None,widthB=1.0,lengthB=0.2,angleB=None
Fancy	fancy	head_length=0.4,head_width=0.4,tail_width=0.4
Simple	simple	head_length=0.5,head_width=0.5,tail_width=0.2
Wedge	wedge	tail_width=0.3,shrink_factor=0.5
BarAB	-	widthA=1.0,angleA=None,widthB=1.0,angleB=None

An instance of any arrow style class is a callable object, whose call signature is:

```
__call__(self, path, mutation_size, linewidth, aspect_ratio=1.)
```

and it returns a tuple of a Path instance and a boolean value. path is a Path instance along which the arrow will be drawn. mutation_size and aspect_ratio have the same meaning as in BoxStyle. linewidth is a line width to be stroked. This is meant to be used to correct the location of the head so that it does not overshoot the destination point, but not all classes support it.

return the instance of the subclass with the given style name.

```
class BarAB(widthA=1.0, angleA=None, widthB=1.0, angleB=None)
```

An arrow with a bar(|) at both ends.

Parameters

widthA [float, optional, default] Width of the bracket

angleA [float, optional, default] Angle between the bracket and the line

widthB [float, optional, default] Width of the bracket

angleB [float, optional, default] Angle between the bracket and the line

```
class BracketA(widthA=1.0, lengthA=0.2, angleA=None)
```

An arrow with a bracket(|) at its end.

Parameters

widthA [float, optional, default] Width of the bracket

lengthA [float, optional, default] Length of the bracket

angleA [float, optional, default] Angle between the bracket and the line

```
class BracketAB(widthA=1.0, lengthA=0.2, angleA=None, widthB=1.0, lengthB=0.2,
                angleB=None)
```

An arrow with a bracket() at both ends.

Parameters

widthA [float, optional, default] Width of the bracket

lengthA [float, optional, default] Length of the bracket

angleA [float, optional, default] Angle between the bracket and the line

widthB [float, optional, default] Width of the bracket

lengthB [float, optional, default] Length of the bracket

angleB [float, optional, default] Angle between the bracket and the line

```
class BracketB(widthB=1.0, lengthB=0.2, angleB=None)
```

An arrow with a bracket() at its end.

Parameters

widthB [float, optional, default] Width of the bracket

lengthB [float, optional, default] Length of the bracket

angleB [float, optional, default] Angle between the bracket and the line

```
class Curve
```

A simple curve without any arrow head.

```
class CurveA(head_length=0.4, head_width=0.2)
```

An arrow with a head at its begin point.

Parameters

head_length [float, optional, default] Length of the arrow head

head_width [float, optional, default] Width of the arrow head

```
class CurveAB(head_length=0.4, head_width=0.2)
```

An arrow with heads both at the begin and the end point.

Parameters

head_length [float, optional, default] Length of the arrow head

head_width [float, optional, default] Width of the arrow head

```
class CurveB(head_length=0.4, head_width=0.2)
```

An arrow with a head at its end point.

Parameters

head_length [float, optional, default] Length of the arrow head

head_width [float, optional, default] Width of the arrow head

class CurveFilledA(head_length=0.4, head_width=0.2)

An arrow with filled triangle head at the begin.

Parameters

head_length [float, optional, default] Length of the arrow head

head_width [float, optional, default] Width of the arrow head

class CurveFilledAB(head_length=0.4, head_width=0.2)

An arrow with filled triangle heads at both ends.

Parameters

head_length [float, optional, default] Length of the arrow head

head_width [float, optional, default] Width of the arrow head

class CurveFilledB(head_length=0.4, head_width=0.2)

An arrow with filled triangle head at the end.

Parameters

head_length [float, optional, default] Length of the arrow head

head_width [float, optional, default] Width of the arrow head

class Fancy(head_length=0.4, head_width=0.4, tail_width=0.4)

A fancy arrow. Only works with a quadratic bezier curve.

Parameters

head_length [float, optional, default] Length of the arrow head

head_width [float, optional, default] Width of the arrow head

tail_width [float, optional, default] Width of the arrow tail

transmute(path, mutation_size, linewidth)

The transmute method is the very core of the ArrowStyle class and must be overridden in the subclasses. It receives the path object along which the arrow will be drawn, and the mutation_size, with which the arrow head etc. will be scaled. The linewidth may be used to adjust the path so that it does not pass beyond the given points. It returns a tuple of a Path instance and a boolean. The boolean value indicate whether the path can be filled or not. The return value can also be a list of paths and list of booleans of a same length.

class Simple(head_length=0.5, head_width=0.5, tail_width=0.2)

A simple arrow. Only works with a quadratic bezier curve.

Parameters

head_length [float, optional, default] Length of the arrow head

head_width [float, optional, default] Width of the arrow head

tail_width [float, optional, default] Width of the arrow tail


```
transmute(path, mutation_size, linewidth)
```

The `transmute` method is the very core of the `ArrowStyle` class and must be overridden in the subclasses. It receives the path object along which the arrow will be drawn, and the `mutation_size`, with which the arrow head etc. will be scaled. The `linewidth` may be used to adjust the path so that it does not pass beyond the given points. It returns a tuple of a `Path` instance and a boolean. The boolean value indicate whether the path can be filled or not. The return value can also be a list of paths and list of booleans of a same length.

```
class Wedge(tail_width=0.3, shrink_factor=0.5)
```

`Wedge(?)` shape. Only works with a quadratic bezier curve. The begin point has a width of the `tail_width` and the end point has a width of 0. At the middle, the width is `shrink_factor*tail_width`.

Parameters

`tail_width` [float, optional, default] Width of the tail

`shrink_factor` [float, optional, default] Fraction of the arrow width at the middle point

```
transmute(path, mutation_size, linewidth)
```

The `transmute` method is the very core of the `ArrowStyle` class and must be overridden in the subclasses. It receives the path object along which the arrow will be drawn, and the `mutation_size`, with which the arrow head etc. will be scaled. The `linewidth` may be used to adjust the path so that it does not pass beyond the given points. It returns a tuple of a `Path` instance and a boolean. The boolean value indicate whether the path can be filled or not. The return value can also be a list of paths and list of booleans of a same length.

matplotlib.patches.BoxStyle

```
class matplotlib.patches.BoxStyle
```

`BoxStyle` is a container class which defines several boxstyle classes, which are used for `FancyBboxPatch`.

A style object can be created as:

```
BoxStyle.Round(pad=0.2)
```

or:

```
BoxStyle("Round", pad=0.2)
```

or:

```
BoxStyle("Round, pad=0.2")
```

Following boxstyle classes are defined.

Class	Name	Attrs
Circle	circle	pad=0.3
DArrow	darrow	pad=0.3
LArrow	larrow	pad=0.3
RArrow	rarrow	pad=0.3
Round	round	pad=0.3,rounding_size=None
Round4	round4	pad=0.3,rounding_size=None
Roundtooth	roundtooth	pad=0.3,tooth_size=None
Sawtooth	sawtooth	pad=0.3,tooth_size=None
Square	square	pad=0.3

An instance of any boxstyle class is an callable object, whose call signature is:

```
__call__(self, x0, y0, width, height, mutation_size, aspect_ratio=1.)
```

and returns a Path instance. `x0`, `y0`, `width` and `height` specify the location and size of the box to be drawn. `mutation_scale` determines the overall size of the mutation (by which I mean the transformation of the rectangle to the fancy box). `mutation_aspect` determines the aspect-ratio of the mutation.

return the instance of the subclass with the given style name.

```
class Circle(pad=0.3)
```

A simple circle box.

Parameters

`pad` [float] The amount of padding around the original box.

```
transmute(x0, y0, width, height, mutation_size)
```

The `transmute` method is a very core of the `BboxTransmuter` class and must be overridden in the subclasses. It receives the location and size of the rectangle, and the `mutation_size`, with which the amount of padding and etc. will be scaled. It returns a Path instance.

```
class DArrow(pad=0.3)
```

(Double) Arrow Box

```
transmute(x0, y0, width, height, mutation_size)
```

The `transmute` method is a very core of the `BboxTransmuter` class and must be overridden in the subclasses. It receives the location and size of the rectangle, and the `mutation_size`, with which the amount of padding and etc. will be scaled. It returns a Path instance.

```
class LArrow(pad=0.3)
```

(left) Arrow Box

```
transmute(x0, y0, width, height, mutation_size)
```

The `transmute` method is a very core of the `BboxTransmuter` class and must be overridden in the subclasses. It receives the location and size of the rectangle, and

the `mutation_size`, with which the amount of padding and etc. will be scaled. It returns a `Path` instance.

```
class RArrow(pad=0.3)
```

(right) Arrow Box

```
transmute(x0, y0, width, height, mutation_size)
```

The `transmute` method is a very core of the `BboxTransmuter` class and must be overridden in the subclasses. It receives the location and size of the rectangle, and the `mutation_size`, with which the amount of padding and etc. will be scaled. It returns a `Path` instance.

```
class Round(pad=0.3, rounding_size=None)
```

A box with round corners.

`pad` amount of padding

`rounding_size` rounding radius of corners. `pad` if `None`

```
transmute(x0, y0, width, height, mutation_size)
```

The `transmute` method is a very core of the `BboxTransmuter` class and must be overridden in the subclasses. It receives the location and size of the rectangle, and the `mutation_size`, with which the amount of padding and etc. will be scaled. It returns a `Path` instance.

```
class Round4(pad=0.3, rounding_size=None)
```

Another box with round edges.

`pad` amount of padding

`rounding_size` rounding size of edges. `pad` if `None`

```
transmute(x0, y0, width, height, mutation_size)
```

The `transmute` method is a very core of the `BboxTransmuter` class and must be overridden in the subclasses. It receives the location and size of the rectangle, and the `mutation_size`, with which the amount of padding and etc. will be scaled. It returns a `Path` instance.

```
class Roundtooth(pad=0.3, tooth_size=None)
```

A rounded tooth box.

`pad` amount of padding

`tooth_size` size of the sawtooth. `pad*` if `None`

```
transmute(x0, y0, width, height, mutation_size)
```

The `transmute` method is a very core of the `BboxTransmuter` class and must be overridden in the subclasses. It receives the location and size of the rectangle, and the `mutation_size`, with which the amount of padding and etc. will be scaled. It returns a `Path` instance.

```
class Sawtooth(pad=0.3, tooth_size=None)
```

A sawtooth box.

`pad` amount of padding

tooth_size size of the sawtooth. pad* if None

transmute(x0, y0, width, height, mutation_size)

The transmute method is a very core of the BboxTransmuter class and must be overridden in the subclasses. It receives the location and size of the rectangle, and the mutation_size, with which the amount of padding and etc. will be scaled. It returns a Path instance.

class Square(pad=0.3)

A simple square box.

pad amount of padding

transmute(x0, y0, width, height, mutation_size)

The transmute method is a very core of the BboxTransmuter class and must be overridden in the subclasses. It receives the location and size of the rectangle, and the mutation_size, with which the amount of padding and etc. will be scaled. It returns a Path instance.

Examples using matplotlib.patches.BoxStyle

- sphx_glr_gallery_shapes_and_collections_artist_reference.py
- sphx_glr_gallery_shapes_and_collections_fancybox_demo.py

matplotlib.patches.Circle

class matplotlib.patches.Circle(xy, radius=5, **kwargs)

A circle patch.

Create true circle at center xy = (x, y) with given radius. Unlike CirclePolygon which is a polygonal approximation, this uses Bézier splines and is much closer to a scale-free circle.

Valid kwargs are:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

`get_radius()`
return the radius of the circle

`radius`
return the radius of the circle

`set_radius(radius)`
Set the radius of the circle

ACCEPTS: float

Examples using `matplotlib.patches.Circle`

- [sphx_glr_gallery_api_patch_collection.py](#)
- [sphx_glr_gallery_pyplots_whats_new_98_4_fancy.py](#)
- [sphx_glr_gallery_images_contours_and_fields_image_clip_path.py](#)
- [sphx_glr_gallery_shapes_and_collections_donut.py](#)
- [sphx_glr_gallery_shapes_and_collections_artist_reference.py](#)
- [sphx_glr_gallery_shapes_and_collections_dolphin.py](#)
- [sphx_glr_gallery_text_labels_and_annotations_fancyarrow_demo.py](#)
- [sphx_glr_gallery_text_labels_and_annotations_demo_annotation_box.py](#)
- [sphx_glr_gallery_showcase_anatomy.py](#)
- [sphx_glr_gallery_axes_grid1_simple_anchored_artists.py](#)
- [sphx_glr_gallery_event_handling_looking_glass.py](#)
- [sphx_glr_gallery_misc_anchored_artists.py](#)
- [sphx_glr_gallery_misc_custom_projection.py](#)
- [sphx_glr_gallery_mplot3d_pathpatch3d.py](#)
- [sphx_glr_gallery_userdemo_anchored_box02.py](#)
- [Legend guide](#)
- [Transformations Tutorial](#)

`matplotlib.patches.CirclePolygon`

`class matplotlib.patches.CirclePolygon(xy, radius=5, resolution=20, **kwargs)`

A polygon-approximation of a circle patch.

Create a circle at $xy = (x, y)$ with given radius. This circle is approximated by a regular polygon with resolution sides. For a smoother circle drawn with splines, see `Circle`.

Valid kwargs are:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

`matplotlib.patches.ConnectionPatch`

```
class matplotlib.patches.ConnectionPatch(xyA, xyB, coordsA, coordsB=None, axesA=None, axesB=None, arrowstyle='-', arrow_transmuter=None, connectionstyle='arc3', connector=None, patchA=None, patchB=None, shrinkA=0.0, shrinkB=0.0, mutation_scale=10.0, mutation_aspect=None, clip_on=False, dpi_cor=1.0, **kwargs)
```

A `ConnectionPatch` class is to make connecting lines between two points (possibly in different axes).

Connect point `xyA` in `coordsA` with point `xyB` in `coordsB`

Valid keys are

Key	Description
<code>arrowstyle</code>	the arrow style
<code>connectionstyle</code>	the connection style
<code>relpos</code>	default is (0.5, 0.5)
<code>patchA</code>	default is bounding box of the text
<code>patchB</code>	default is None
<code>shrinkA</code>	default is 2 points
<code>shrinkB</code>	default is 2 points
<code>mutation_scale</code>	default is text size (in points)
<code>mutation_aspect</code>	default is 1.
<code>?</code>	any key for <code>matplotlib.patches.PathPatch</code>

`coordsA` and `coordsB` are strings that indicate the coordinates of `xyA` and `xyB`.

Property	Description
'figure points'	points from the lower left corner of the figure
'figure pixels'	pixels from the lower left corner of the figure
'figure fraction'	0,0 is lower left of figure and 1,1 is upper, right
'axes points'	points from lower left corner of axes
'axes pixels'	pixels from lower left corner of axes
'axes fraction'	0,1 is lower left of axes and 1,1 is upper right
'data'	use the coordinate system of the object being annotated (default)
'offset points'	Specify an offset (in points) from the xy value
'polar'	you can specify theta, r for the annotation, even in cartesian plots. Note that if you are using a polar axes, you do not need to specify polar for the coordinate system since that is the native "data" coordinate system.

`draw(renderer)`

Draw.

`get_annotation_clip()`

Return `annotation_clip` attribute. See `set_annotation_clip()` for the meaning of return values.

`get_path_in_displaycoord()`

Return the mutated path of the arrow in the display coord

`set_annotation_clip(b)`

set `annotation_clip` attribute.

- True: the annotation will only be drawn when `self.xy` is inside the axes.
- False: the annotation will always be drawn regardless of its position.
- None: the `self.xy` will be checked only if `xycoords` is "data"

Examples using `matplotlib.patches.ConnectionPatch`

- `sphx_glr_gallery_userdemo_connect_simple01.py`

`matplotlib.patches.ConnectionStyle`

class `matplotlib.patches.ConnectionStyle`

`ConnectionStyle` is a container class which defines several connectionstyle classes, which is used to create a path between two points. These are mainly used with `FancyArrowPatch`.

A connectionstyle object can be either created as:

```
ConnectionStyle.Arc3(rad=0.2)
```

or:

```
ConnectionStyle("Arc3", rad=0.2)
```

or:

```
ConnectionStyle("Arc3, rad=0.2")
```

The following classes are defined

Class	Name	Attrs
Angle	angle	angleA=90,angleB=0,rad=0.0
Angle3	angle3	angleA=90,angleB=0
Arc	arc	angleA=0,angleB=0,armA=None,armB=None,rad=0.0
Arc3	arc3	rad=0.0
Bar	bar	armA=0.0,armB=0.0,fraction=0.3,angle=None

An instance of any connection style class is an callable object, whose call signature is:

```
__call__(self, posA, posB,
         patchA=None, patchB=None,
         shrinkA=2., shrinkB=2.)
```

and it returns a `Path` instance. `posA` and `posB` are tuples of x,y coordinates of the two points to be connected. `patchA` (or `patchB`) is given, the returned path is clipped so that it start (or end) from the boundary of the patch. The path is further shrunk by `shrinkA` (or `shrinkB`) which is given in points.

return the instance of the subclass with the given style name.

```
class Angle(angleA=90, angleB=0, rad=0.0)
```

Creates a piecewise continuous quadratic bezier path between two points. The path has a one passing-through point placed at the intersecting point of two lines which crosses the start (or end) point and has a angle of `angleA` (or `angleB`). The connecting edges are rounded with `rad`.

angleA starting angle of the path
angleB ending angle of the path
rad rounding radius of the edge
connect(posA, posB)

class Angle3(angleA=90, angleB=0)

Creates a simple quadratic bezier curve between two points. The middle control points is placed at the intersecting point of two lines which crosses the start (or end) point and has a angle of angleA (or angleB).

angleA starting angle of the path
angleB ending angle of the path
connect(posA, posB)

class Arc(angleA=0, angleB=0, armA=None, armB=None, rad=0.0)

Creates a picewise continuous quadratic bezier path between two points. The path can have two passing-through points, a point placed at the distance of armA and angle of angleA from point A, another point with respect to point B. The edges are rounded with rad.

angleA : starting angle of the path
angleB : ending angle of the path
armA : length of the starting arm
armB : length of the ending arm
rad : rounding radius of the edges
connect(posA, posB)

class Arc3(rad=0.0)

Creates a simple quadratic bezier curve between two points. The curve is created so that the middle control point (C1) is located at the same distance from the start (C0) and end points(C2) and the distance of the C1 to the line connecting C0-C2 is rad times the distance of C0-C2.

rad curvature of the curve.
connect(posA, posB)

class Bar(armA=0.0, armB=0.0, fraction=0.3, angle=None)

A line with angle between A and B with armA and armB. One of the arms is extended so that they are connected in a right angle. The length of armA is determined by (armA + fraction x AB distance). Same for armB.

Parameters

armA [float] minimum length of armA
armB [float] minimum length of armB
fraction [float] a fraction of the distance between two points that will be added to armA and armB.
angle [float or None] angle of the connecting line (if None, parallel to A and B)
connect(posA, posB)

`matplotlib.patches.Ellipse`

`class matplotlib.patches.Ellipse(xy, width, height, angle=0.0, **kwargs)`

A scale-free ellipse.

xy center of ellipse

width total length (diameter) of horizontal axis

height total length (diameter) of vertical axis

angle rotation in degrees (anti-clockwise)

Valid kwargs are:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\ ' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

`get_patch_transform()`

Return the Transform instance which takes patch coordinates to data coordinates.

For example, one may define a patch of a circle which represents a radius of 5 by providing coordinates for a unit circle, and a transform which scales the coordinates (the patch coordinate) by 5.

`get_path()`

Return the vertices of the rectangle

Examples using `matplotlib.patches.Ellipse`

- `sphinx_glr_gallery_shapes_and_collections_ellipse_demo.py`
- `sphinx_glr_gallery_shapes_and_collections_hatch_demo.py`
- `sphinx_glr_gallery_shapes_and_collections_artist_reference.py`
- `sphinx_glr_gallery_text_labels_and_annotations_annotation_demo.py`
- `sphinx_glr_gallery_misc_anchored_artists.py`
- `sphinx_glr_gallery_units_ellipse_with_units.py`
- `sphinx_glr_gallery_userdemo_anchored_box03.py`
- `sphinx_glr_gallery_userdemo_anchored_box04.py`
- `sphinx_glr_gallery_userdemo_annotate_explain.py`
- `sphinx_glr_gallery_userdemo_simple_annotate01.py`
- [Legend guide](#)

`matplotlib.patches.FancyArrow`

```
class matplotlib.patches.FancyArrow(x, y, dx, dy, width=0.001,
                                     length_includes_head=False, head_width=None,
                                     head_length=None, shape='full', overhang=0,
                                     head_starts_at_zero=False, **kwargs)
```

Like `Arrow`, but lets you set head width and head height independently.

Constructor arguments

`width`: float (default: 0.001) width of full arrow tail

`length_includes_head`: bool (default: False) True if head is to be counted in calculating the length.

`head_width`: float or None (default: 3*width) total width of the full arrow head

`head_length`: float or None (default: 1.5 * head_width) length of arrow head

`shape`: ['full', 'left', 'right'] (default: 'full') draw the left-half, right-half, or full arrow

`overhang`: float (default: 0) fraction that the arrow is swept back (0 overhang means triangular shape). Can be negative or greater than one.

`head_starts_at_zero`: bool (default: False) if True, the head starts being drawn at coordinate 0 instead of ending at coordinate 0.

Other valid kwargs (inherited from `Patch`) are:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

`matplotlib.patches.FancyArrowPatch`

```
class matplotlib.patches.FancyArrowPatch(posA=None, posB=None, path=None, arrowstyle='simple', arrow_transmuter=None, connectionstyle='arc3', connector=None, patchA=None, patchB=None, shrinkA=2, shrinkB=2, mutation_scale=1, mutation_aspect=None, dpi_cor=1, **kwargs)
```

A fancy arrow patch. It draws an arrow using the `ArrowStyle`.

The head and tail positions are fixed at the specified start and end points of the arrow, but the size and shape (in display coordinates) of the arrow does not change when the axis is moved or zoomed.

If `posA` and `posB` are given, a path connecting two points is created according to `connectionstyle`. The path will be clipped with `patchA` and `patchB` and further shrunk by `shrinkA` and `shrinkB`. An arrow is drawn along this resulting path using the `arrowstyle` parameter.

Alternatively if `path` is provided, an arrow is drawn along this path and `patchA`, `patchB`, `shrinkA`, and `shrinkB` are ignored.

Parameters

`posA, posB` [None, tuple, optional (default: None)] (x,y) coordinates of arrow tail and arrow head respectively.

`path` [None, Path (default: None)] `matplotlib.path.Path` instance. If provided, an arrow is drawn along this path and `patchA`, `patchB`, `shrinkA`, and `shrinkB` are ignored.

`arrowstyle` [str or `ArrowStyle`, optional (default: 'simple')]

Describes how the fancy arrow will be drawn. It can be string of the available `arrowstyle` names, with optional comma-separated attributes, or an `ArrowStyle` instance. The optional attributes are meant to be scaled with the `mutation_scale`. The following arrow styles are available:

Class	Name	Attrs
Curve	-	None
CurveB	->	head_length=0.4,head_width=0.2
BracketB	-[widthB=1.0,lengthB=0.2,angleB=None
Curve-FilledB	- >	head_length=0.4,head_width=0.2
CurveA	<-	head_length=0.4,head_width=0.2
CurveAB	<->	head_length=0.4,head_width=0.2
Curve-FilledA	< -	head_length=0.4,head_width=0.2
Curve-FilledAB	< ->	head_length=0.4,head_width=0.2
BracketA] -	widthA=1.0,lengthA=0.2,angleA=None
BracketAB] -[widthA=1.0,lengthA=0.2,angleA=None,widthB=1.0,lengthB=0.2,angleB=None
Fancy	fancy	head_length=0.4,head_width=0.4,tail_width=0.4
Simple	simple	head_length=0.5,head_width=0.5,tail_width=0.2
Wedge	wedge	tail_width=0.3,shrink_factor=0.5
BarAB	-	widthA=1.0,angleA=None,widthB=1.0,angleB=None

arrow_transmuter : Ignored

connectionstyle [str, ConnectionStyle, or None, optional]

(default: 'arc3')

Describes how posA and posB are connected. It can be an instance of the ConnectionStyle class or a string of the connectionstyle name, with optional comma-separated attributes. The following connection styles are available:

Class	Name	Attrs
Angle	angle	angleA=90,angleB=0,rad=0.0
Angle3	angle3	angleA=90,angleB=0
Arc	arc	angleA=0,angleB=0,armA=None,armB=None,rad=0.0
Arc3	arc3	rad=0.0
Bar	bar	armA=0.0,armB=0.0,fraction=0.3,angle=None

connector : Ignored

patchA, patchB [None, Patch, optional (default: None)] Head and tail patch respectively. matplotlib.patch.Patch instance.

shrinkA, shrinkB [scalar, optional (default: 2)] Shrinking factor of the tail and head of the arrow respectively

`mutation_scale` [scalar, optional (default: 1)] Value with which attributes of `arrowstyle` (e.g., `head_length`) will be scaled.

`mutation_aspect` [None, scalar, optional (default: None)] The height of the rectangle will be squeezed by this value before the mutation and the mutated box will be stretched by the inverse of it.

`dpi_cor` [scalar, optional (default: 1)] `dpi_cor` is currently used for linewidth-related things and shrink factor. Mutation scale is affected by this.

Notes

Valid kwargs are:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\' ' ' '-.' '+' 'x' 'o' 'O' '.' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

`draw(renderer)`
Draw the Patch to the given renderer.

`get_arrowstyle()`
Return the arrowstyle object.

`get_connectionstyle()`
Return the ConnectionStyle instance.

`get_dpi_cor()`

`dpi_cor` is currently used for linewidth-related things and shrink factor. Mutation scale is affected by this.

Returns

`dpi_cor` [scalar]

`get_mutation_aspect()`

Return the aspect ratio of the bbox mutation.

`get_mutation_scale()`

Return the mutation scale.

Returns

`scale` [scalar]

`get_path()`

Return the path of the arrow in the data coordinates. Use `get_path_in_displaycoord()` method to retrieve the arrow path in display coordinates.

`get_path_in_displaycoord()`

Return the mutated path of the arrow in display coordinates.

`set_arrowstyle(arrowstyle=None, **kw)`

Set the arrow style. Old attributes are forgotten. Without arguments (or with `arrowstyle=None`) returns available box styles as a list of strings.

Parameters

`arrowstyle` [None, ArrowStyle, str, optional (default: None)] Can be a string with arrowstyle name with optional comma-separated attributes, e.g.:

```
set_arrowstyle("Fancy,head_length=0.2")
```

Alternatively attributes can be provided as keywords, e.g.:

```
set_arrowstyle("fancy", head_length=0.2)
```

`set_connectionstyle(connectionstyle, **kw)`

Set the connection style. Old attributes are forgotten.

Parameters

`connectionstyle` [None, ConnectionStyle instance, or string] Can be a string with connectionstyle name with optional comma-separated attributes, e.g.:

```
set_connectionstyle("arc,angleA=0,armA=30,rad=10")
```

Alternatively, the attributes can be provided as keywords, e.g.:

`set_connectionstyle("arc", angleA=0, armA=30, rad=10)`

Without any arguments (or with `connectionstyle=None`), return available styles as a list of strings.

`set_dpi_cor(dpi_cor)`

`dpi_cor` is currently used for linewidth-related things and shrink factor. Mutation scale is affected by this.

Parameters

`dpi_cor` [scalar]

`set_mutation_aspect(aspect)`

Set the aspect ratio of the bbox mutation.

Parameters

`aspect` [scalar]

`set_mutation_scale(scale)`

Set the mutation scale.

Parameters

`scale` [scalar]

`set_patchA(patchA)`

Set the tail patch.

Parameters

`patchA` [Patch] matplotlib.patch.Patch instance.

`set_patchB(patchB)`

Set the head patch.

Parameters

`patchB` [Patch] matplotlib.patch.Patch instance.

`set_positions(posA, posB)`

Set the begin and end positions of the connecting path.

Parameters

`posA, posB` [None, tuple] (x,y) coordinates of arrow tail and arrow head respectively. If `None` use current value.

`matplotlib.patches.FancyBboxPatch`

```
class matplotlib.patches.FancyBboxPatch(xy, width, height, boxstyle='round',
                                         bbox_transmuter=None, mutation_scale=1.0,
                                         mutation_aspect=None, **kwargs)
```

Draw a fancy box around a rectangle with lower left at `xy*=(*x, y)` with specified width and height.

FancyBboxPatch class is similar to Rectangle class, but it draws a fancy box around the rectangle. The transformation of the rectangle box to the fancy box is delegated to the BoxTransmuterBase and its derived classes.

xy = lower left corner

width, height

boxstyle determines what kind of fancy box will be drawn. It can be a string of the style name with a comma separated attribute, or an instance of BoxStyle. Following box styles are available.

Class	Name	Attrs
Circle	circle	pad=0.3
DArrow	darrow	pad=0.3
LArrow	larrow	pad=0.3
RArrow	rarrow	pad=0.3
Round	round	pad=0.3,rounding_size=None
Round4	round4	pad=0.3,rounding_size=None
Roundtooth	roundtooth	pad=0.3,tooth_size=None
Sawtooth	sawtooth	pad=0.3,tooth_size=None
Square	square	pad=0.3

mutation_scale : a value with which attributes of boxstyle (e.g., pad) will be scaled. default=1.

mutation_aspect : The height of the rectangle will be squeezed by this value before the mutation and the mutated box will be stretched by the inverse of it. default=None.

Valid kwargs are:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

`get_bbox()`

`get_boxstyle()`

Return the boxstyle object

`get_height()`

Return the height of the rectangle

`get_mutation_aspect()`

Return the aspect ratio of the bbox mutation.

`get_mutation_scale()`

Return the mutation scale.

`get_path()`

Return the mutated path of the rectangle

`get_width()`

Return the width of the rectangle

`get_x()`

Return the left coord of the rectangle

`get_y()`

Return the bottom coord of the rectangle

`set_bounds(*args)`

Set the bounds of the rectangle: l,b,w,h

ACCEPTS: (left, bottom, width, height)

`set_boxstyle(boxstyle=None, **kw)`

Set the box style.

boxstyle can be a string with boxstyle name with optional comma-separated attributes. Alternatively, the attrs can be provided as keywords:

```
set_boxstyle("round,pad=0.2")
set_boxstyle("round", pad=0.2)
```

Old attrs simply are forgotten.

Without argument (or with `boxstyle = None`), it returns available box styles.

The following boxstyles are available:

Class	Name	Attrs
Circle	circle	pad=0.3
DArrow	darrow	pad=0.3
LArrow	larrow	pad=0.3
RArrow	rarrow	pad=0.3
Round	round	pad=0.3,rounding_size=None
Round4	round4	pad=0.3,rounding_size=None
Roundtooth	roundtooth	pad=0.3,tooth_size=None
Sawtooth	sawtooth	pad=0.3,tooth_size=None
Square	square	pad=0.3

ACCEPTS: ['circle' | 'darrow' | 'larrow' | 'rarrow' | 'round' | 'round4' | 'roundtooth' | 'sawtooth' | 'square']

`set_height(h)`

Set the width rectangle

ACCEPTS: float

`set_mutation_aspect(aspect)`
Set the aspect ratio of the bbox mutation.

ACCEPTS: float

`set_mutation_scale(scale)`
Set the mutation scale.

ACCEPTS: float

`set_width(w)`
Set the width rectangle

ACCEPTS: float

`set_x(x)`
Set the left coord of the rectangle

ACCEPTS: float

`set_y(y)`
Set the bottom coord of the rectangle

ACCEPTS: float

Examples using `matplotlib.patches.FancyBboxPatch`

- `sphx_glr_gallery_shapes_and_collections_artist_reference.py`
- `sphx_glr_gallery_shapes_and_collections_fancybox_demo.py`

`matplotlib.patches.Patch`

```
class matplotlib.patches.Patch(edgecolor=None,      facecolor=None,      color=None,
                               linewidth=None,      linestyle=None,      antialiased=None,
                               hatch=None, fill=True, capstyle=None, joinstyle=None,
                               **kwargs)
```

A patch is a 2D artist with a face color and an edge color.

If any of `edgecolor`, `facecolor`, `linewidth`, or `antialiased` are `None`, they default to their rc params setting.

The following kwarg properties are supported

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\ ' ' ' '-' '+' 'x' 'o' 'O' '.' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

`contains(mouseevent, radius=None)`

Test whether the mouse event occurred in the patch.

Returns T/F, {}

`contains_point(point, radius=None)`

Returns True if the given point is inside the path (transformed with its transform attribute).

radius allows the path to be made slightly larger or smaller.

`contains_points(points, radius=None)`
Returns a bool array which is True if the (closed) path contains the corresponding point. (transformed with its transform attribute).

points must be Nx2 array. radius allows the path to be made slightly larger or smaller.

`draw(renderer)`
Draw the Patch to the given renderer.

`fill`
return whether fill is set

`get_aa()`
Returns True if the Patch is to be drawn with antialiasing.

`get_antialiased()`
Returns True if the Patch is to be drawn with antialiasing.

`get_capstyle()`
Return the current capstyle

`get_data_transform()`
Return the Transform instance which maps data coordinates to physical coordinates.

`get_ec()`
Return the edge color of the Patch.

`get_edgecolor()`
Return the edge color of the Patch.

`get_extents()`
Return a Bbox object defining the axis-aligned extents of the Patch.

`get_facecolor()`
Return the face color of the Patch.

`get_fc()`
Return the face color of the Patch.

`get_fill()`
return whether fill is set

`get_hatch()`
Return the current hatching pattern

`get_joinstyle()`
Return the current joinstyle

`get_linestyle()`
Return the linestyle. Will be one of ['solid' | 'dashed' | 'dashdot' | 'dotted']

`get_linewidth()`
Return the line width in points.

`get_ls()`
Return the linestyle. Will be one of ['solid' | 'dashed' | 'dashdot' | 'dotted']

`get_lw()`

Return the line width in points.

`get_patch_transform()`

Return the Transform instance which takes patch coordinates to data coordinates.

For example, one may define a patch of a circle which represents a radius of 5 by providing coordinates for a unit circle, and a transform which scales the coordinates (the patch coordinate) by 5.

`get_path()`

Return the path of this patch

`get_transform()`

Return the Transform applied to the Patch.

`get_verts()`

Return a copy of the vertices used in this patch

If the patch contains Bezier curves, the curves will be interpolated by line segments. To access the curves as curves, use `get_path()`.

`get_window_extent(renderer=None)`

Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

`set_aa(aa)`

alias for `set_antialiased`

`set_alpha(alpha)`

Set the alpha transparency of the patch.

ACCEPTS: float or None

`set_antialiased(aa)`

Set whether to use antialiased rendering.

Parameters

b [bool or None]

`set_capstyle(s)`

Set the patch capstyle

ACCEPTS: ['butt' | 'round' | 'projecting']

`set_color(c)`

Set both the edgecolor and the facecolor.

ACCEPTS: matplotlib color spec

See also:

`set_facecolor()`, `set_edgecolor()` For setting the edge or face color individually.

`set_ec(color)`
alias for `set_edgecolor`

`set_edgecolor(color)`
Set the patch edge color

ACCEPTS: mpl color spec, None, 'none', or 'auto'

`set_facecolor(color)`
Set the patch face color

ACCEPTS: mpl color spec, or None for default, or 'none' for no color

`set_fc(color)`
alias for `set_facecolor`

`set_fill(b)`
Set whether to fill the patch.

Parameters

b [bool]

`set_hatch(hatch)`
Set the hatching pattern

hatch can be one of:

/	- diagonal hatching
\	- back diagonal
	- vertical
-	- horizontal
+	- crossed
x	- crossed diagonal
o	- small circle
O	- large circle
.	- dots
*	- stars

Letters can be combined, in which case all the specified hatchings are done. If same letter repeats, it increases the density of hatching of that pattern.

Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

ACCEPTS: `['/' | '\' | '|' | '-' | '+' | 'x' | 'o' | 'O' | '.' | '*']`

`set_joinstyle(s)`
Set the patch joinstyle

ACCEPTS: `['miter' | 'round' | 'bevel']`

`set_linestyle(ls)`
Set the patch linestyle

linestyle	description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line

Alternatively a dash tuple of the following form can be provided:

`(offset, onoffseq),`

where onoffseq is an even length tuple of on and off ink in points.

ACCEPTS: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '--' |
 '-.' | ':' | 'None' | ' ' | '']

Parameters

`ls` [{ '-', '--', '-.', ':'}] and more see description] The line style.

`set_linewidth(w)`
Set the patch linewidth in points

ACCEPTS: float or None for default

`set_ls(ls)`
alias for `set_linestyle`

`set_lw(lw)`
alias for `set_linewidth`

`update_from(other)`
Updates this Patch from the properties of other.

`validCap` = ('butt', 'round', 'projecting')

`validJoin` = ('miter', 'round', 'bevel')

`zorder` = 1

Examples using `matplotlib.patches.Patch`

- [sphx_glr_gallery_text_labels_and_annotations_custom_legends.py](#)
- [Legend guide](#)

`matplotlib.patches.PathPatch`

```
class matplotlib.patches.PathPatch(path, **kwargs)
```

A general polycurve path patch.

`path` is a `matplotlib.path.Path` object.

Valid kwargs are:

Property	Description
<code>agg_filter</code>	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
<code>alpha</code>	float or None
<code>animated</code>	bool
<code>antialiased</code> or <code>aa</code>	bool or None
<code>capstyle</code>	['butt' 'round' 'projecting']
<code>clip_box</code>	a Bbox instance
<code>clip_on</code>	bool
<code>clip_path</code>	[(Path, Transform) Patch None]
<code>color</code>	matplotlib color spec
<code>contains</code>	a callable function
<code>edgecolor</code> or <code>ec</code>	mpl color spec, None, 'none', or 'auto'
<code>facecolor</code> or <code>fc</code>	mpl color spec, or None for default, or 'none' for no color
<code>figure</code>	a Figure instance
<code>fill</code>	bool
<code>gid</code>	an id string
<code>hatch</code>	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '.' '*']
<code>joinstyle</code>	['miter' 'round' 'bevel']
<code>label</code>	object
<code>linestyle</code> or <code>ls</code>	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
<code>linewidth</code> or <code>lw</code>	float or None for default
<code>path_effects</code>	AbstractPathEffect
<code>picker</code>	[None bool float callable]
<code>rasterized</code>	bool or None
<code>sketch_params</code>	(scale: float, length: float, randomness: float)
<code>snap</code>	bool or None
<code>transform</code>	Transform
<code>url</code>	a url string
<code>visible</code>	bool
<code>zorder</code>	float

See also:

`Patch` For additional kwargs

`get_path()`
Return the path of this patch

Examples using `matplotlib.patches.PathPatch`

- `sphx_glr_gallery_api_quad_bezier.py`
- `sphx_glr_gallery_api_histogram_path.py`
- `sphx_glr_gallery_pyplots_compound_path_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_image_demo.py`
- `sphx_glr_gallery_shapes_and_collections_compound_path.py`
- `sphx_glr_gallery_shapes_and_collections_path_patch.py`
- `sphx_glr_gallery_shapes_and_collections_donut.py`
- `sphx_glr_gallery_shapes_and_collections_artist_reference.py`
- `sphx_glr_gallery_shapes_and_collections_dolphin.py`
- `sphx_glr_gallery_text_labels_and_annotations_demo_text_path.py`
- `sphx_glr_gallery_showcase_firefox.py`
- `Animated histogram`
- `sphx_glr_gallery_event_handling_path_editor.py`
- `sphx_glr_gallery_mplot3d_pathpatch3d.py`
- [Path Tutorial](#)

`matplotlib.patches.Polygon`

`class matplotlib.patches.Polygon(xy, closed=True, **kwargs)`
A general polygon patch.

`xy` is a numpy array with shape `Nx2`.

If `closed` is `True`, the polygon will be closed so the starting and ending points are the same.

Valid kwargs are:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\ ' ' ' '-.' '+' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

See also:

`Patch` For additional kwargs

`get_closed()`

Returns if the polygon is closed

Returns

`closed` [bool] If the path is closed

`get_path()`
Get the path of the polygon

Returns

`path` [Path] The Path object for the polygon

`get_xy()`
Get the vertices of the path

Returns

`vertices` [numpy array] The coordinates of the vertices as a Nx2 ndarray.

`set_closed(closed)`
Set if the polygon is closed

Parameters

`closed` [bool] True if the polygon is closed

`set_xy(xy)`
Set the vertices of the polygon

Parameters

`xy` [numpy array or iterable of pairs] The coordinates of the vertices as a Nx2 ndarray or iterable of pairs.

`xy`
Set/get the vertices of the polygon. This property is provided for backward compatibility with matplotlib 0.91.x only. New code should use `get_xy()` and `set_xy()` instead.

Examples using `matplotlib.patches.Polygon`

- `sphinx_glr_gallery_api_patch_collection.py`
- `sphinx_glr_gallery_statistics_boxplot_demo.py`
- `sphinx_glr_gallery_shapes_and_collections_hatch_demo.py`
- `sphinx_glr_gallery_showcase_integral.py`
- `sphinx_glr_gallery_event_handling_trifinder_event_demo.py`
- `sphinx_glr_gallery_event_handling_poly_editor.py`

`matplotlib.patches.Rectangle`

`class matplotlib.patches.Rectangle(xy, width, height, angle=0.0, **kwargs)`
Draw a rectangle with lower left at `xy = (x, y)` with specified width, height and rotation angle.

Parameters

xy: length-2 tuple The bottom and left rectangle coordinates
width: Rectangle width
height: Rectangle height
angle: float, optional rotation in degrees anti-clockwise about xy (default is 0.0)
fill: bool, optional Whether to fill the rectangle (default is True)

Notes

Valid kwargs are:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\ ' ' ' '- ' '+ ' 'x' 'o' 'O' '.' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

get_bbox()

get_height()

Return the height of the rectangle

get_patch_transform()

Return the Transform instance which takes patch coordinates to data coordinates.

For example, one may define a patch of a circle which represents a radius of 5 by providing coordinates for a unit circle, and a transform which scales the coordinates (the patch coordinate) by 5.

`get_path()`
Return the vertices of the rectangle

`get_width()`
Return the width of the rectangle

`get_x()`
Return the left coord of the rectangle

`get_xy()`
Return the left and bottom coords of the rectangle

`get_y()`
Return the bottom coord of the rectangle

`set_bounds(*args)`
Set the bounds of the rectangle: l,b,w,h
ACCEPTS: (left, bottom, width, height)

`set_height(h)`
Set the height of the rectangle

`set_width(w)`
Set the width of the rectangle

`set_x(x)`
Set the left coord of the rectangle

`set_xy(xy)`
Set the left and bottom coords of the rectangle
ACCEPTS: 2-item sequence

`set_y(y)`
Set the bottom coord of the rectangle

`xy`
Return the left and bottom coords of the rectangle

Examples using `matplotlib.patches.Rectangle`

- `sphx_glr_gallery_pyplots_text_layout.py`
- `sphx_glr_gallery_statistics_errorbars_and_boxes.py`
- `sphx_glr_gallery_shapes_and_collections_artist_reference.py`
- `sphx_glr_gallery_event_handling_viewlims.py`
- `sphx_glr_gallery_event_handling_pick_event_demo.py`

- [sphx_glr_gallery_misc_anchored_artists.py](#)
- [sphx_glr_gallery_units_artist_tests.py](#)
- [sphx_glr_gallery_widgets_menu.py](#)
- [Legend guide](#)
- [Transformations Tutorial](#)
- [Specifying Colors](#)
- [Text properties and layout](#)

`matplotlib.patches.RegularPolygon`

```
class matplotlib.patches.RegularPolygon(xy, numVertices, radius=5, orientation=0,  
                                         **kwargs)
```

A regular polygon patch.

Constructor arguments:

`xy` A length 2 tuple (x, y) of the center.

`numVertices` the number of vertices.

`radius` The distance from the center to each of the vertices.

`orientation` rotates the polygon (in radians).

Valid kwargs are:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\ ' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

`get_patch_transform()`

Return the Transform instance which takes patch coordinates to data coordinates.

For example, one may define a patch of a circle which represents a radius of 5 by providing coordinates for a unit circle, and a transform which scales the coordinates (the patch coordinate) by 5.

`get_path()`

Return the path of this patch

numvertices

orientation

radius

xy

Examples using `matplotlib.patches.RegularPolygon`

- `sphx_glr_gallery_shapes_and_collections_artist_reference.py`

`matplotlib.patches.Shadow`

`class matplotlib.patches.Shadow(patch, ox, oy, props=None, **kwargs)`

Create a shadow of the given patch offset by `ox`, `oy`. `props`, if not `None`, is a patch property update dictionary. If `None`, the shadow will have the same color as the face, but darkened.

`kwargs` are

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\ ' ' ' '- ' '+ ' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

`draw(renderer)`

Draw the Patch to the given renderer.

`get_patch_transform()`

Return the Transform instance which takes patch coordinates to data coordinates.

For example, one may define a patch of a circle which represents a radius of 5 by providing coordinates for a unit circle, and a transform which scales the coordinates (the patch coordinate) by 5.

`get_path()`
Return the path of this patch

Examples using `matplotlib.patches.Shadow`

- `sphinx_glr_gallery_text_labels_and_annotations_demo_text_path.py`
- `sphinx_glr_gallery_misc_svg_filter_pie.py`

`matplotlib.patches.Wedge`

`class matplotlib.patches.Wedge(center, r, theta1, theta2, width=None, **kwargs)`
Wedge shaped patch.

Draw a wedge centered at x, y center with radius r that sweeps theta1 to theta2 (in degrees).
If width is given, then a partial wedge is drawn from inner radius r - width to outer radius r.

Valid kwargs are:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

get_path()
Return the path of this patch

set_center(center)

set_radius(radius)

set_theta1(theta1)

```
set_theta2(theta2)
```

```
set_width(width)
```

Examples using `matplotlib.patches.Wedge`

- `sphx_glr_gallery_api_patch_collection.py`
- `sphx_glr_gallery_shapes_and_collections_artist_reference.py`

`matplotlib.patches.YAArrow`

```
class matplotlib.patches.YAArrow(figure, xytip, xybase, width=4, frac=0.1, headwidth=12,  
                                **kwargs)
```

Yet another arrow class.

This is an arrow that is defined in display space and has a tip at `x1, y1` and a base at `x2, y2`.

Constructor arguments:

`xytip` (x, y) location of arrow tip

`xybase` (x, y) location the arrow base mid point

`figure` The Figure instance (`fig.dpi`)

`width` The width of the arrow in points

`frac` The fraction of the arrow length occupied by the head

`headwidth` The width of the base of the arrow head in points

Valid `kwargs` are:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\ ' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed' 'dashdot' 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

`get_patch_transform()`

Return the Transform instance which takes patch coordinates to data coordinates.

For example, one may define a patch of a circle which represents a radius of 5 by providing coordinates for a unit circle, and a transform which scales the coordinates (the patch coordinate) by 5.

`get_path()`

Return the path of this patch

`getpoints(x1, y1, x2, y2, k)`

For line segment defined by (x1, y1) and (x2, y2) return the points on the line that is perpendicular to the line and intersects (x2, y2) and the distance from (x2, y2) of the returned points is k.

57.1.2 Functions

`bbox__artist(artist, renderer[, props, fill])`

This is a debug function to draw a rectangle around the bounding box returned by `get_window_extent()` of an artist, to test whether the artist is returning the correct bbox.

`draw__bbox(bbox, renderer[, color, trans])`

This is a debug function to draw a rectangle around the bounding box returned by `get_window_extent()` of an artist, to test whether the artist is returning the correct bbox.

`matplotlib.patches.bbox__artist`

`matplotlib.patches.bbox__artist(artist, renderer, props=None, fill=True)`

This is a debug function to draw a rectangle around the bounding box returned by `get_window_extent()` of an artist, to test whether the artist is returning the correct bbox.

`props` is a dict of rectangle props with the additional property 'pad' that sets the padding around the bbox in points.

`matplotlib.patches.draw__bbox`

`matplotlib.patches.draw__bbox(bbox, renderer, color='k', trans=None)`

This is a debug function to draw a rectangle around the bounding box returned by `get_window_extent()` of an artist, to test whether the artist is returning the correct bbox.

58.1 matplotlib.path

A module for dealing with the polylines used throughout Matplotlib.

The primary class for polyline handling in Matplotlib is Path. Almost all vector drawing makes use of Paths somewhere in the drawing pipeline.

Whilst a Path instance itself cannot be drawn, some Artist subclasses, such as PathPatch and PathCollection, can be used for convenient Path visualisation.

```
class matplotlib.path.Path(vertices, codes=None, __interpolation_steps=1, closed=False,
                           readonly=False)
```

Bases: `object`

Path represents a series of possibly disconnected, possibly closed, line and curve segments.

The underlying storage is made up of two parallel numpy arrays:

- vertices: an Nx2 float array of vertices
- codes: an N-length uint8 array of vertex types

These two arrays always have the same length in the first dimension. For example, to represent a cubic curve, you must provide three vertices as well as three codes CURVE3.

The code types are:

- STOP [1 vertex (ignored)] A marker for the end of the entire path (currently not required and ignored)
- MOVETO [1 vertex] Pick up the pen and move to the given vertex.
- LINETO [1 vertex] Draw a line from the current position to the given vertex.
- CURVE3 [1 control point, 1 endpoint] Draw a quadratic Bezier curve from the current position, with the given control point, to the given end point.
- CURVE4 [2 control points, 1 endpoint] Draw a cubic Bezier curve from the current position, with the given control points, to the given end point.
- CLOSEPOLY [1 vertex (ignored)] Draw a line segment to the start point of the current polyline.

Users of Path objects should not access the vertices and codes arrays directly. Instead, they should use `iter_segments()` or `cleaned()` to get the vertex/code pairs. This is important, since many Path objects, as an optimization, do not store a codes at all, but have a default one provided for them by `iter_segments()`.

Some behavior of Path objects can be controlled by rcParams. See the rcParams whose keys contain 'path'.

Note: The vertices and codes arrays should be treated as immutable – there are a number of optimizations and assumptions made up front in the constructor that will not change when the data changes.

Create a new path with the given vertices and codes.

Parameters

`vertices` [array_like] The (n, 2) float array, masked array or sequence of pairs representing the vertices of the path.

If vertices contains masked values, they will be converted to NaNs which are then handled correctly by the Agg PathIterator and other consumers of path data, such as `iter_segments()`.

`codes` [{None, array_like}, optional] n-length array integers representing the codes of the path. If not None, codes must be the same length as vertices. If None, vertices will be treated as a series of line segments.

`_interpolation_steps` [int, optional] Used as a hint to certain projections, such as Polar, that this path should be linearly interpolated immediately before drawing. This attribute is primarily an implementation detail and is not intended for public use.

`closed` [bool, optional] If codes is None and closed is True, vertices will be treated as line segments of a closed polygon.

`readonly` [bool, optional] Makes the path behave in an immutable way and sets the vertices and codes as read-only arrays.

CLOSEPOLY = 79

CURVE3 = 3

CURVE4 = 4

LINETO = 2

MOVETO = 1

NUM_VERTICES_FOR_CODE = {0: 1, 1: 1, 2: 1, 3: 2, 4: 3, 79: 1}

A dictionary mapping Path codes to the number of vertices that the code expects.

STOP = 0

classmethod arc(theta1, theta2, n=None, is_wedge=False)

Return an arc on the unit circle from angle theta1 to angle theta2 (in degrees).

theta2 is unwrapped to produce the shortest arc within 360 degrees. That is, if theta2 > theta1 + 360, the arc will be from theta1 to theta2 - 360 and not a full circle plus some extra overlap.

If n is provided, it is the number of spline segments to make. If n is not provided, the number of spline segments is determined based on the delta between theta1 and theta2.

Masionobe, L. 2003. [Drawing an elliptical arc using polylines, quadratic or cubic Bezier curves.](#)

classmethod circle(center=(0.0, 0.0), radius=1.0, readonly=False)

Return a Path representing a circle of a given radius and center.

Parameters

center [pair of floats] The center of the circle. Default (0, 0).

radius [float] The radius of the circle. Default is 1.

readonly [bool] Whether the created path should have the "readonly" argument set when creating the Path instance.

Notes

The circle is approximated using cubic Bezier curves. This uses 8 splines around the circle using the approach presented here:

Lancaster, Don. [Approximating a Circle or an Ellipse Using Four Bezier Cubic Splines.](#)

cleaned(transform=None, remove_nans=False, clip=None, quantize=False, simplify=False, curves=False, stroke_width=1.0, snap=False, sketch=None)

Cleans up the path according to the parameters returning a new Path instance.

See also:

See iter_segments() for details of the keyword arguments.

Returns

Path instance with cleaned up vertices and codes.

clip_to_bbox(bbox, inside=True)

Clip the path to the given bounding box.

The path must be made up of one or more closed polygons. This algorithm will not behave correctly for unclosed paths.

If `inside` is `True`, clip to the inside of the box, otherwise to the outside of the box.

`code_type`

alias of `numpy.uint8`

`codes`

The list of codes in the Path as a 1-D numpy array. Each code is one of STOP, MOVETO, LINETO, CURVE3, CURVE4 or CLOSEPOLY. For codes that correspond to more than one vertex (CURVE3 and CURVE4), that code will be repeated so that the length of `self.vertices` and `self.codes` is always the same.

`contains_path(path, transform=None)`

Returns whether this (closed) path completely contains the given path.

If `transform` is not `None`, the path will be transformed before performing the test.

`contains_point(point, transform=None, radius=0.0)`

Returns whether the (closed) path contains the given point.

If `transform` is not `None`, the path will be transformed before performing the test.

`radius` allows the path to be made slightly larger or smaller.

`contains_points(points, transform=None, radius=0.0)`

Returns a bool array which is `True` if the (closed) path contains the corresponding point.

If `transform` is not `None`, the path will be transformed before performing the test.

`radius` allows the path to be made slightly larger or smaller.

`copy()`

Returns a shallow copy of the Path, which will share the vertices and codes with the source Path.

`deepcopy(memo=None)`

Returns a deepcopy of the Path. The Path will not be readonly, even if the source Path is.

`get_extents(transform=None)`

Returns the extents (`xmin`, `ymin`, `xmax`, `ymax`) of the path.

Unlike computing the extents on the vertices alone, this algorithm will take into account the curves and deal with control points appropriately.

`has_nonfinite`

`True` if the vertices array has nonfinite values.

`classmethod hatch(hatchpattern, density=6)`

Given a hatch specifier, `hatchpattern`, generates a Path that can be used in a repeated hatching pattern. `density` is the number of lines per unit square.

`interpolated(steps)`

Returns a new path resampled to length `N x steps`. Does not currently handle interpolating curves.

`intersects_bbox(bbox, filled=True)`

Returns True if this path intersects a given Bbox.

`filled`, when True, treats the path as if it was filled. That is, if the path completely encloses the bounding box, `intersects_bbox()` will return True.

The bounding box is always considered filled.

`intersects_path(other, filled=True)`

Returns True if this path intersects another given path.

`filled`, when True, treats the paths as if they were filled. That is, if one path completely encloses the other, `intersects_path()` will return True.

`iter_segments(transform=None, remove_nans=True, clip=None, snap=False, stroke_width=1.0, simplify=None, curves=True, sketch=None)`

Iterates over all of the curve segments in the path. Each iteration returns a 2-tuple (vertices, code), where vertices is a sequence of 1 - 3 coordinate pairs, and code is one of the Path codes.

Additionally, this method can provide a number of standard cleanups and conversions to the path.

Parameters

`transform` [None or Transform instance] If not None, the given affine transformation will be applied to the path.

`remove_nans` [{False, True}, optional] If True, will remove all NaNs from the path and insert MOVETO commands to skip over them.

`clip` [None or sequence, optional] If not None, must be a four-tuple (x1, y1, x2, y2) defining a rectangle in which to clip the path.

`snap` [None or bool, optional] If None, auto-snap to pixels, to reduce fuzziness of rectilinear lines. If True, force snapping, and if False, don't snap.

`stroke_width` [float, optional]

The width of the stroke being drawn. Needed as a hint for the snapping algorithm.

`simplify` [None or bool, optional]

If True, perform simplification, to remove vertices that do not affect the appearance of the path. If False, perform no simplification. If None, use the `should_simplify` member variable. See also the `rcParams.path.simplify` and `path.simplify_threshold`.

`curves` [{True, False}, optional] If True, curve segments will be returned as curve segments. If False, all curves will be converted to line segments.

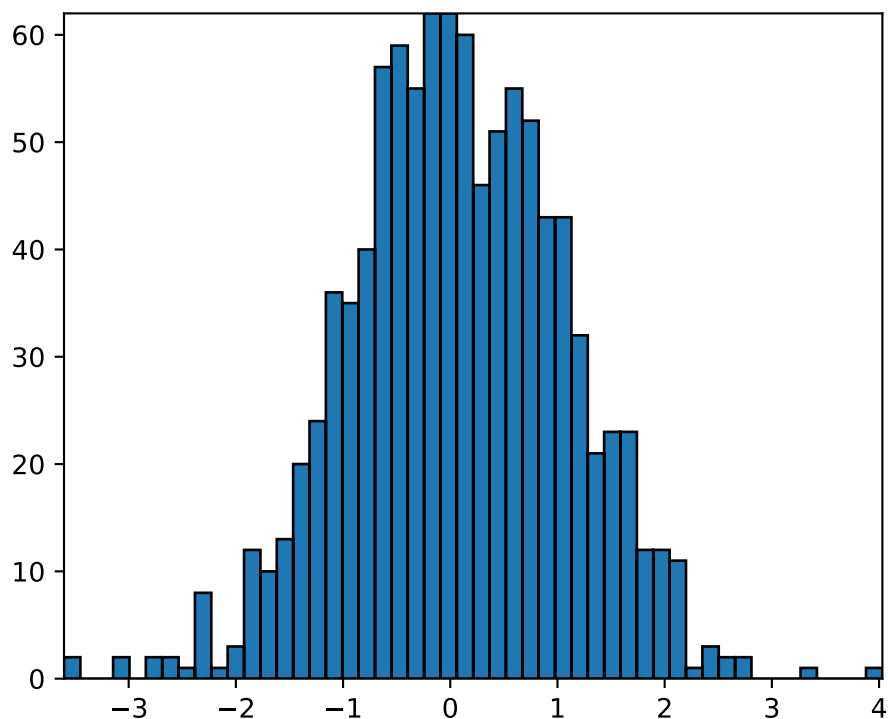
sketch [None or sequence, optional] If not None, must be a 3-tuple of the form (scale, length, randomness), representing the sketch parameters.

classmethod make_compound_path(*args)

Make a compound path from a list of Path objects.

classmethod make_compound_path_from_polys(XY)

Make a compound path object to draw a number of polygons with equal numbers of sides XY is a (numpolys x numsides x 2) numpy array of vertices. Return object is a Path



readonly

True if the Path is read-only.

should_simplify

True if the vertices array should be simplified.

simplify_threshold

The fraction of a pixel difference below which vertices will be simplified out.

to_polygons(transform=None, width=0, height=0, closed_only=True)

Convert this path to a list of polygons or polylines. Each polygon/polyline is an Nx2 array of vertices. In other words, each polygon has no MOVETO instructions or curves. This is useful for displaying in backends that do not support compound paths or Bezier curves, such as GDK.

If width and height are both non-zero then the lines will be simplified so that vertices outside of (0, 0), (width, height) will be clipped.

If `closed_only` is `True` (default), only closed polygons, with the last point being the same as the first point, will be returned. Any unclosed polylines in the path will be explicitly closed. If `closed_only` is `False`, any unclosed polygons in the path will be returned as unclosed polygons, and the closed polygons will be returned explicitly closed by setting the last point to the same as the first point.

`transformed(transform)`

Return a transformed copy of the path.

See also:

`matplotlib.transforms.TransformPath` A specialized path class that will cache the transformed result and automatically update when the transform changes.

`classmethod unit_circle()`

Return the readonly Path of the unit circle.

For most cases, `Path.circle()` will be what you want.

`classmethod unit_circle_righthalf()`

Return a Path of the right half of a unit circle. The circle is approximated using cubic Bezier curves. This uses 4 splines around the circle using the approach presented here:

Lancaster, Don. [Approximating a Circle or an Ellipse Using Four Bezier Cubic Splines](#).

`classmethod unit_rectangle()`

Return a Path instance of the unit rectangle from (0, 0) to (1, 1).

`classmethod unit_regular_asterisk(numVertices)`

Return a Path for a unit regular asterisk with the given `numVertices` and radius of 1.0, centered at (0, 0).

`classmethod unit_regular_polygon(numVertices)`

Return a Path instance for a unit regular polygon with the given `numVertices` and radius of 1.0, centered at (0, 0).

`classmethod unit_regular_star(numVertices, innerCircle=0.5)`

Return a Path for a unit regular star with the given `numVertices` and radius of 1.0, centered at (0, 0).

`vertices`

The list of vertices in the Path as an Nx2 numpy array.

`classmethod wedge(theta1, theta2, n=None)`

Return a wedge of the unit circle from angle `theta1` to angle `theta2` (in degrees).

`theta2` is unwrapped to produce the shortest wedge within 360 degrees. That is, if `theta2 > theta1 + 360`, the wedge will be from `theta1` to `theta2 - 360` and not a full circle plus some extra overlap.

If `n` is provided, it is the number of spline segments to make. If `n` is not provided, the number of spline segments is determined based on the delta between `theta1` and `theta2`.

`matplotlib.path.get_path_collection_extents(master_transform, paths, transforms, offsets, offset_transform)`

Given a sequence of Path objects, Transform objects and offsets, as found in a PathCollection, returns the bounding box that encapsulates all of them.

`master_transform` is a global transformation to apply to all paths

`paths` is a sequence of Path instances.

`transforms` is a sequence of Affine2D instances.

`offsets` is a sequence of (x, y) offsets (or an Nx2 array)

`offset_transform` is a Affine2D to apply to the offsets before applying the offset to the path.

The way that paths, transforms and offsets are combined follows the same method as for collections. Each is iterated over independently, so if you have 3 paths, 2 transforms and 1 offset, their combinations are as follows:

(A, A, A), (B, B, A), (C, A, A)

`matplotlib.path.get_paths_extents(paths, transforms=[])`

Given a sequence of Path objects and optional Transform objects, returns the bounding box that encapsulates all of them.

`paths` is a sequence of Path instances.

`transforms` is an optional sequence of Affine2D instances to apply to each path.

PATHEFFECTS

59.1 matplotlib.patheffects

Defines classes for path effects. The path effects are supported in Text, Line2D and Patch.

```
class matplotlib.patheffects.AbstractPathEffect(offset=(0.0, 0.0))
```

Bases: `object`

A base class for path effects.

Subclasses should override the `draw_path` method to add effect functionality.

Parameters

`offset` [pair of floats] The offset to apply to the path, measured in points.

```
draw_path(renderer, gc, tpath, affine, rgbFace=None)
```

Derived should override this method. The arguments are the same as `matplotlib.backend_bases.RendererBase.draw_path()` except the first argument is a `renderer`.

```
class matplotlib.patheffects.Normal(offset=(0.0, 0.0))
```

Bases: `matplotlib.patheffects.AbstractPathEffect`

The "identity" `PathEffect`.

The `Normal PathEffect`'s sole purpose is to draw the original artist with no special path effect.

Parameters

`offset` [pair of floats] The offset to apply to the path, measured in points.

```
class matplotlib.patheffects.PathEffectRenderer(path_effects, renderer)
```

Bases: `matplotlib.backend_bases.RendererBase`

Implements a `Renderer` which contains another `renderer`.

This proxy then intercepts `draw` calls, calling the appropriate `AbstractPathEffect` `draw` method.

Note: Not all methods have been overridden on this `RendererBase` subclass. It may be necessary to add further methods to extend the `PathEffects` capabilities further.

Parameters

`path_effects` [iterable of `AbstractPathEffect`] The path effects which this renderer represents.

`renderer` [`matplotlib.backend_bases.RendererBase` instance]

`copy_with_path_effect(path_effects)`

`draw_markers(gc, marker_path, marker_trans, path, *args, **kwargs)`

Draws a marker at each of the vertices in `path`. This includes all vertices, including control points on curves. To avoid that behavior, those vertices should be removed before calling this function.

This provides a fallback implementation of `draw_markers` that makes multiple calls to `draw_path()`. Some backends may want to override this method in order to draw the marker only once and reuse it multiple times.

Parameters

`gc` [`GraphicsContextBase`] The graphics context

`marker_trans` [`matplotlib.transforms.Transform`] An affine transform applied to the marker.

`trans` [`matplotlib.transforms.Transform`] An affine transform applied to the path.

`draw_path(gc, tpath, affine, rgbFace=None)`

Draws a `Path` instance using the given affine transform.

`draw_path_collection(gc, master_transform, paths, *args, **kwargs)`

Draws a collection of paths selecting drawing properties from the lists `facecolors`, `edgecolors`, `linewidths`, `linestyles` and `antialiaseds`. `offsets` is a list of offsets to apply to each of the paths. The offsets in `offsets` are first transformed by `offsetTrans` before being applied. `offset_position` may be either "screen" or "data" depending on the space that the offsets are in.

This provides a fallback implementation of `draw_path_collection()` that makes multiple calls to `draw_path()`. Some backends may want to override this in order to render each set of path data only once, and then reference that path multiple times with the different offsets, colors, styles etc. The generator methods `_iter_collection_raw_paths()` and `_iter_collection()` are provided to help with (and standardize) the implementation across backends. It is highly recommended to use those generators, so that changes to the behavior of `draw_path_collection()` can be made globally.

`new_gc()`

Return an instance of a `GraphicsContextBase`

`points_to_pixels(points)`

Convert points to display units

You need to override this function (unless your backend doesn't have a `dpi`, e.g., `postscript` or `svg`). Some imaging systems assume some value for pixels per inch:

$$\text{points to pixels} = \text{points} * \text{pixels_per_inch} / 72.0 * \text{dpi} / 72.0$$

Parameters

points [scalar or array_like] a float or a numpy array of float

Returns

Points converted to pixels

class matplotlib.patheffects.PathPatchEffect(offset=(0, 0), **kwargs)

Bases: matplotlib.patheffects.AbstractPathEffect

Draws a PathPatch instance whose Path comes from the original PathEffect artist.

Parameters

offset [pair of floats] The offset to apply to the path, in points.

**kwargs : All keyword arguments are passed through to the PathPatch constructor. The properties which cannot be overridden are "path", "clip_box" "transform" and "clip_path".

draw_path(renderer, gc, tpath, affine, rgbFace)

Derived should override this method. The arguments are the same as matplotlib.backend_bases.RendererBase.draw_path() except the first argument is a renderer.

class matplotlib.patheffects.SimpleLineShadow(offset=(2, -2), shadow_color='k', alpha=0.3, rho=0.3, **kwargs)

Bases: matplotlib.patheffects.AbstractPathEffect

A simple shadow via a line.

Parameters

offset [pair of floats] The offset to apply to the path, in points.

shadow_color [color] The shadow color. Default is black. A value of None takes the original artist's color with a scale factor of rho.

alpha [float] The alpha transparency of the created shadow patch. Default is 0.3.

rho [float] A scale factor to apply to the rgbFace color if shadow_rgbFace is None. Default is 0.3.

**kwargs Extra keywords are stored and passed through to AbstractPathEffect._update_gc().

draw_path(renderer, gc, tpath, affine, rgbFace)

Overrides the standard draw_path to add the shadow offset and necessary color changes for the shadow.

class matplotlib.patheffects.SimplePatchShadow(offset=(2, -2), shadow_rgbFace=None, alpha=None, rho=0.3, **kwargs)

Bases: matplotlib.patheffects.AbstractPathEffect

A simple shadow via a filled patch.

Parameters

offset [pair of floats] The offset of the shadow in points.

shadow_rgbFace [color] The shadow color.

alpha [float] The alpha transparency of the created shadow patch. Default is 0.3. <http://matplotlib.1069221.n5.nabble.com/path-effects-question-td27630.html>

rho [float] A scale factor to apply to the rgbFace color if shadow_rgbFace is not specified. Default is 0.3.

****kwargs** Extra keywords are stored and passed through to AbstractPathEffect.__update_gc().

draw_path(renderer, gc, tpath, affine, rgbFace)

Overrides the standard draw_path to add the shadow offset and necessary color changes for the shadow.

class matplotlib.patheffects.Stroke(offset=(0, 0), ****kwargs**)

Bases: matplotlib.patheffects.AbstractPathEffect

A line based PathEffect which re-draws a stroke.

The path will be stroked with its gc updated with the given keyword arguments, i.e., the keyword arguments should be valid gc parameter values.

draw_path(renderer, gc, tpath, affine, rgbFace)

draw the path with updated gc.

class matplotlib.patheffects.withSimplePatchShadow(offset=(2, -2), shadow_rgbFace=None, alpha=None, rho=0.3, ****kwargs**)

Bases: matplotlib.patheffects.SimplePatchShadow

Adds a simple SimplePatchShadow and then draws the original Artist to avoid needing to call Normal.

Parameters

offset [pair of floats] The offset of the shadow in points.

shadow_rgbFace [color] The shadow color.

alpha [float] The alpha transparency of the created shadow patch. Default is 0.3. <http://matplotlib.1069221.n5.nabble.com/path-effects-question-td27630.html>

rho [float] A scale factor to apply to the rgbFace color if shadow_rgbFace is not specified. Default is 0.3.

****kwargs** Extra keywords are stored and passed through to AbstractPathEffect.__update_gc().

`draw_path(renderer, gc, tpath, affine, rgbFace)`

Overrides the standard `draw_path` to add the shadow offset and necessary color changes for the shadow.

`class matplotlib.patheffects.withStroke(offset=(0, 0), **kwargs)`

Bases: `matplotlib.patheffects.Stroke`

Adds a simple `Stroke` and then draws the original `Artist` to avoid needing to call `Normal`.

The path will be stroked with its `gc` updated with the given keyword arguments, i.e., the keyword arguments should be valid `gc` parameter values.

`draw_path(renderer, gc, tpath, affine, rgbFace)`

draw the path with updated `gc`.

PROJECTIONS

60.1 matplotlib.projections

class matplotlib.projections.ProjectionRegistry
Bases: `object`

Manages the set of projections available to the system.

`get_projection_class(name)`
Get a projection class from its name.

`get_projection_names()`
Get a list of the names of all projections currently registered.

`register(*projections)`
Register a new set of projection(s).

`matplotlib.projections.get_projection_class(projection=None)`
Get a projection class from its name.

If projection is None, a standard rectilinear projection is returned.

`matplotlib.projections.get_projection_names()`
Get a list of acceptable projection names.

`matplotlib.projections.process_projection_requirements(figure, *args, **kwargs)`
Handle the args/kwargs to for `add_axes/add_subplot/gca`, returning:

<code>(axes_proj_class, proj_class_kwargs, proj_stack_key)</code>

Which can be used for new axes initialization/identification.

Note: `kwargs` is modified in place.

`matplotlib.projections.register_projection(cls)`

60.2 matplotlib.projections.polar

```
class matplotlib.projections.polar.InvertedPolarTransform(axis=None,
                                                         use_rmin=True,          __ap-
                                                         ply_theta_transforms=True)
```

Bases: matplotlib.transforms.Transform

The inverse of the polar transform, mapping Cartesian coordinate space x and y back to theta and r.

input_dims = 2

inverted()

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

x === self.inverted().transform(self.transform(x))

is_separable = False

output_dims = 2

transform_non_affine(xy)

Performs only the non-affine part of the transformation.

transform(values) is always equivalent to transform_affine(transform_non_affine(values)).

In non-affine transformations, this is generally equivalent to transform(values). In affine transformations, this is always a no-op.

Accepts a numpy array of shape (N x input_dims) and returns a numpy array of shape (N x output_dims).

Alternatively, accepts a numpy array of length input_dims and returns a numpy array of length output_dims.

```
class matplotlib.projections.polar.PolarAffine(scale_transform, limits)
```

Bases: matplotlib.transforms.Affine2DBase

The affine part of the polar projection. Scales the output so that maximum radius rests on the edge of the axes circle.

limits is the view limit of the data. The only part of its bounds that is used is the y limits (for the radius limits). The theta range is handled by the non-affine transform.

get_matrix()

Get the Affine transformation array for the affine part of this transform.

```
class matplotlib.projections.polar.PolarAxes(*args, **kwargs)
```

Bases: matplotlib.axes._axes.Axes

A polar graph projection, where the input dimensions are theta, r.

Theta starts pointing east and goes anti-clockwise.

Build an Axes instance in Figure fig with rect=[left, bottom, width, height] in Figure coordinates

Optional keyword arguments:

Keyword	Description
adjustable	['box' 'datalim']
alpha	float: the alpha transparency (can be None)
anchor	['C', 'SW', 'S', 'SE', 'E', 'NE', 'N', 'NW', 'W']
aspect	['auto' 'equal' aspect_ratio]
autoscale_on	bool; whether to autoscale the viewlim
axisbelow	[bool 'line'] draw the grids and ticks below or above most other artists, or below lines but above patches
cursor_props	a (float, color) tuple
figure	a Figure instance
frame_on	bool; whether to draw the axes frame
label	the axes label
navigate	bool
navigate_mode	['PAN' 'ZOOM' None] the navigation toolbar button status
position	[left, bottom, width, height] in class:Figure coords
sharex	an class:Axes instance to share the x-axis with
sharey	an class:Axes instance to share the y-axis with
title	the title string
visible	bool, whether the axes is visible
xlabel	the xlabel
xlim	(xmin, xmax) view limits
xscale	['linear' 'log' 'logit' 'symlog']
xticklabels	sequence of strings
xticks	sequence of floats
ylabel	the ylabel strings
ylim	(ymin, ymax) view limits
yscale	['linear' 'log' 'logit' 'symlog']
yticklabels	sequence of strings
yticks	sequence of floats


```
class InvertedPolarTransform(axis=None, use_rmin=True, _ap-
                             ply_theta_transforms=True)
    Bases: matplotlib.transforms.Transform

    The inverse of the polar transform, mapping Cartesian coordinate space x and y back
    to theta and r.

    input_dims = 2

    inverted()
        Return the corresponding inverse transformation.

        The return value of this method should be treated as temporary. An update to self
        does not cause a corresponding update to its inverted copy.

        x == self.inverted().transform(self.transform(x))

    is_separable = False

    output_dims = 2

    transform_non_affine(xy)
        Performs only the non-affine part of the transformation.

        transform(values) is always equivalent to trans-
        form_affine(transform_non_affine(values)).

        In non-affine transformations, this is generally equivalent to transform(values). In
        affine transformations, this is always a no-op.

        Accepts a numpy array of shape (N x input_dims) and returns a numpy array of
        shape (N x output_dims).

        Alternatively, accepts a numpy array of length input_dims and returns a numpy
        array of length output_dims.

class PolarAffine(scale_transform, limits)
    Bases: matplotlib.transforms.Affine2DBase

    The affine part of the polar projection. Scales the output so that maximum radius rests
    on the edge of the axes circle.

    limits is the view limit of the data. The only part of its bounds that is used is the y
    limits (for the radius limits). The theta range is handled by the non-affine transform.

    get_matrix()
        Get the Affine transformation array for the affine part of this transform.

class PolarTransform(axis=None, use_rmin=True, _apply_theta_transforms=True)
    Bases: matplotlib.transforms.Transform

    The base polar transform. This handles projection theta and r into Cartesian coordinate
    space x and y, but does not perform the ultimate affine transformation into the correct
    position.
```

input_dims = 2

inverted()

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

`x === self.inverted().transform(self.transform(x))`

is_separable = False

output_dims = 2

transform_non_affine(tr)

Performs only the non-affine part of the transformation.

`transform(values)` is always equivalent to `transform_affine(transform_non_affine(values))`.

In non-affine transformations, this is generally equivalent to `transform(values)`. In affine transformations, this is always a no-op.

Accepts a numpy array of shape (N x input_dims) and returns a numpy array of shape (N x output_dims).

Alternatively, accepts a numpy array of length input_dims and returns a numpy array of length output_dims.

transform_path_non_affine(path)

Returns a path, transformed only by the non-affine part of this transform.

path: a Path instance.

`transform_path(path)` is equivalent to `transform_path_affine(transform_path_non_affine(values))`

class RadialLocator(base, axes=None)

Bases: matplotlib.ticker.Locator

Used to locate radius ticks.

Ensures that all ticks are strictly positive. For all other tasks, it delegates to the base Locator (which may be different depending on the scale of the r-axis).

autoscale()

autoscale the view limits

pan(numsteps)

Pan numticks (can be positive or negative)

refresh()

refresh internal information based on current lim

view_limits(vmin, vmax)

select a scale for the range from vmin to vmax

Normally this method is overridden by subclasses to change locator behaviour.

`zoom(direction)`

Zoom in/out on axis; if direction is >0 zoom in, else zoom out

`class ThetaFormatter`

Bases: `matplotlib.ticker.Formatter`

Used to format the theta tick labels. Converts the native unit of radians into degrees and adds a degree symbol.

`class ThetaLocator(base)`

Bases: `matplotlib.ticker.Locator`

Used to locate theta ticks.

This will work the same as the base locator except in the case that the view spans the entire circle. In such cases, the previously used default locations of every 45 degrees are returned.

`autoscale()`

autoscale the view limits

`pan(numsteps)`

Pan numticks (can be positive or negative)

`refresh()`

refresh internal information based on current lim

`set__axis(axis)`

`view_limits(vmin, vmax)`

select a scale for the range from vmin to vmax

Normally this method is overridden by subclasses to change locator behaviour.

`zoom(direction)`

Zoom in/out on axis; if direction is >0 zoom in, else zoom out

`can_pan()`

Return True if this axes supports the pan/zoom button functionality.

For polar axes, this is slightly misleading. Both panning and zooming are performed by the same button. Panning is performed in azimuth while zooming is done along the radial.

`can_zoom()`

Return True if this axes supports the zoom box button functionality.

Polar axes do not support zoom boxes.

`cla()`

Clear the current axes.

`drag_pan(button, key, x, y)`

Called when the mouse moves during a pan operation.

button is the mouse button number:

- 1: LEFT
- 2: MIDDLE
- 3: RIGHT

key is a "shift" key

x, y are the mouse coordinates in display coords.

Note: Intended to be overridden by new projection types.

`draw(*args, **kwargs)`

Draw everything (plot lines, axes, labels)

`end_pan()`

Called when a pan operation completes (when the mouse button is up.)

Note: Intended to be overridden by new projection types.

`format_coord(theta, r)`

Return a format string formatting the coordinate using Unicode characters.

`get_data_ratio()`

Return the aspect ratio of the data itself. For a polar plot, this should always be 1.0

`get_rlabel_position()`

Returns

float The theta position of the radius labels in degrees.

`get_rmax()`

`get_rmin()`

`get_rorigin()`

`get_theta_direction()`

Get the direction in which theta increases.

-1: Theta increases in the clockwise direction

1: Theta increases in the counterclockwise direction

`get_theta_offset()`

Get the offset for the location of 0 in radians.

`get_thetamax()`

`get_thetamin()`

`get_xaxis_text1_transform(pad)`

Get the transformation used for drawing x-axis labels, which will add the given amount of padding (in points) between the axes and the label. The x-direction is in data coordinates and the y-direction is in axis coordinates. Returns a 3-tuple of the form:

(transform, valign, halign)

where valign and halign are requested alignments for the text.

Note: This transformation is primarily used by the Axis class, and is meant to be overridden by new kinds of projections that may need to place axis elements in different locations.

`get_xaxis_text2_transform(pad)`

Get the transformation used for drawing the secondary x-axis labels, which will add the given amount of padding (in points) between the axes and the label. The x-direction is in data coordinates and the y-direction is in axis coordinates. Returns a 3-tuple of the form:

(transform, valign, halign)

where valign and halign are requested alignments for the text.

Note: This transformation is primarily used by the Axis class, and is meant to be overridden by new kinds of projections that may need to place axis elements in different locations.

`get_xaxis_transform(which='grid')`

Get the transformation used for drawing x-axis labels, ticks and gridlines. The x-direction is in data coordinates and the y-direction is in axis coordinates.

Note: This transformation is primarily used by the Axis class, and is meant to be overridden by new kinds of projections that may need to place axis elements in different locations.

`get_yaxis_text1_transform(pad)`

Get the transformation used for drawing y-axis labels, which will add the given amount of padding (in points) between the axes and the label. The x-direction is in axis coordinates and the y-direction is in data coordinates. Returns a 3-tuple of the form:

(transform, valign, halign)

where valign and halign are requested alignments for the text.

Note: This transformation is primarily used by the Axis class, and is meant to be overridden by new kinds of projections that may need to place axis elements in different locations.

`get_yaxis_text2_transform(pad)`

Get the transformation used for drawing the secondary y-axis labels, which will add the given amount of padding (in points) between the axes and the label. The x-direction is in axis coordinates and the y-direction is in data coordinates. Returns a 3-tuple of the form:

(transform, valign, halign)

where valign and halign are requested alignments for the text.

Note: This transformation is primarily used by the Axis class, and is meant to be overridden by new kinds of projections that may need to place axis elements in different locations.

`get_yaxis_transform(which='grid')`

Get the transformation used for drawing y-axis labels, ticks and gridlines. The x-direction is in axis coordinates and the y-direction is in data coordinates.

Note: This transformation is primarily used by the Axis class, and is meant to be overridden by new kinds of projections that may need to place axis elements in different locations.

`name = 'polar'`

`set_rgrids(radii, labels=None, angle=None, fmt=None, **kwargs)`

Set the radial locations and labels of the r grids.

The labels will appear at radial distances radii at the given angle in degrees.

labels, if not None, is a `len(radii)` list of strings of the labels to use at each radius.

If labels is None, the built-in formatter will be used.

Return value is a list of tuples (line, label), where line is Line2D instances and the label is Text instances.

kwargs are optional text properties for the labels:

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value
alpha		float (0.0 transparent through 1.0 opaque)
animated		bool
backgroundcolor		any matplotlib color
bbox		FancyBboxPatch prop dict
clip_box		a matplotlib.transforms.Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
color		any matplotlib color
contains		a callable function
family or fontfamily or fontname or name		[FONTNAME 'serif' 'sans-serif' 'cursive' 'fantasy' 'monospace']
figure		a Figure instance
fontproperties or font_properties		a matplotlib.font_manager.FontProperties instance
gid		an id string
horizontalalignment or ha		['center' 'right' 'left']
label		object
linespacing		float (multiple of font size)
multialignment or ma		['left' 'right' 'center']
path_effects		AbstractPathEffect
picker		[None bool float callable]
position		(x,y)
rasterized		bool or None
rotation		[angle in degrees 'vertical' 'horizontal']
rotation_mode		[None "default" "anchor"]
size or fontsize		[size in points 'xx-small' 'x-small' 'small' 'medium' 'large' 'x-large']
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
stretch or fontstretch		[a numeric value in range 0-1000 'ultra-condensed' 'extra-condensed']
style or fontstyle		['normal' 'italic' 'oblique']
text		string or anything printable with '%s' conversion.
transform		Transform
url		a url string
usetex		bool or None
variant or fontvariant		['normal' 'small-caps']
verticalalignment or va		['center' 'top' 'bottom' 'baseline']
visible		bool
weight or fontweight		[a numeric value in range 0-1000 'ultralight' 'light' 'normal' 'bold']
wrap		bool
x		float
y		float
zorder		float

ACCEPTS: sequence of floats

`set_rlabel_position(value)`

Updates the theta position of the radius labels.

Parameters

`value` [number] The angular position of the radius labels in degrees.

`set_rlim(*args, **kwargs)`

`set_rmax(rmax)`

`set_rmin(rmin)`

`set_rorigin(rorigin)`

`set_rscale(*args, **kwargs)`

`set_rticks(*args, **kwargs)`

`set_theta_direction(direction)`

Set the direction in which theta increases.

clockwise, -1: Theta increases in the clockwise direction

counterclockwise, anticlockwise, 1: Theta increases in the counterclockwise direction

`set_theta_offset(offset)`

Set the offset for the location of 0 in radians.

`set_theta_zero_location(loc, offset=0.0)`

Sets the location of theta's zero. (Calls `set_theta_offset` with the correct value in radians under the hood.)

`loc` [str] May be one of "N", "NW", "W", "SW", "S", "SE", "E", or "NE".

`offset` [float, optional] An offset in degrees to apply from the specified loc. Note: this offset is always applied counter-clockwise regardless of the direction setting.

`set_thetagrids(angles, labels=None, frac=None, fmt=None, **kwargs)`

Set the angles at which to place the theta grids (these gridlines are equal along the theta dimension). `angles` is in degrees.

`labels`, if not None, is a `len(angles)` list of strings of the labels to use at each angle.

If `labels` is None, the labels will be `fmt % angle`

`frac` is the fraction of the polar axes radius at which to place the label (1 is the edge). e.g., 1.05 is outside the axes and 0.95 is inside the axes.

Return value is a list of tuples (line, label), where line is Line2D instances and the label is Text instances.

kwargs are optional text properties for the labels:

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value
alpha		float (0.0 transparent through 1.0 opaque)
animated		bool
backgroundcolor		any matplotlib color
bbox		FancyBboxPatch prop dict
clip_box		a matplotlib.transforms.Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
color		any matplotlib color
contains		a callable function
family or fontfamily or fontname or name		[FONTNAME 'serif' 'sans-serif' 'cursive' 'fantasy' 'monospace']
figure		a Figure instance
fontproperties or font_properties		a matplotlib.font_manager.FontProperties instance
gid		an id string
horizontalalignment or ha		['center' 'right' 'left']
label		object
linespacing		float (multiple of font size)
multialignment or ma		['left' 'right' 'center']
path_effects		AbstractPathEffect
picker		[None bool float callable]
position		(x,y)
rasterized		bool or None
rotation		[angle in degrees 'vertical' 'horizontal']
rotation_mode		[None "default" "anchor"]
size or fontsize		[size in points 'xx-small' 'x-small' 'small' 'medium' 'large' 'x-large']
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
stretch or fontstretch		[a numeric value in range 0-1000 'ultra-condensed' 'extra-condensed']
style or fontstyle		['normal' 'italic' 'oblique']
text		string or anything printable with '%s' conversion.
transform		Transform
url		a url string
usetex		bool or None
variant or fontvariant		['normal' 'small-caps']
verticalalignment or va		['center' 'top' 'bottom' 'baseline']
visible		bool
weight or fontweight		[a numeric value in range 0-1000 'ultralight' 'light' 'normal' 'bold' 'extra-bold']
wrap		bool
x		float
y		float
zorder		float

ACCEPTS: sequence of floats

`set_thetalim(*args, **kwargs)`

`set_thetamax(thetamax)`

`set_thetamin(thetamin)`

`set_xscale(scale, *args, **kwargs)`

Set the x-axis scale.

Parameters

value [{`"linear"`, `"log"`, `"symlog"`, `"logit"`}] scaling strategy to apply

See also:

`matplotlib.scale.LinearScale` linear transform

`matplotlib.scale.LogTransform` log transform

`matplotlib.scale.SymmetricalLogTransform` symlog transform

`matplotlib.scale.LogisticTransform` logit transform

Notes

Different kwargs are accepted, depending on the scale. See the scale module for more information.

`set_yscale(*args, **kwargs)`

Set the y-axis scale.

Parameters

value [{`"linear"`, `"log"`, `"symlog"`, `"logit"`}] scaling strategy to apply

See also:

`matplotlib.scale.LinearScale` linear transform

`matplotlib.scale.LogTransform` log transform

`matplotlib.scale.SymmetricalLogTransform` symlog transform

`matplotlib.scale.LogisticTransform` logit transform

Notes

Different kwargs are accepted, depending on the scale. See the scale module for more information.

`start_pan(x, y, button)`

Called when a pan operation has started.

`x`, `y` are the mouse coordinates in display coords. `button` is the mouse button number:

- 1: LEFT
- 2: MIDDLE
- 3: RIGHT

Note: Intended to be overridden by new projection types.

`class matplotlib.projections.polar.PolarTransform(axis=None, use_rmin=True, __apply_theta_transforms=True)`

Bases: `matplotlib.transforms.Transform`

The base polar transform. This handles projection `theta` and `r` into Cartesian coordinate space `x` and `y`, but does not perform the ultimate affine transformation into the correct position.

`input_dims = 2`

`inverted()`

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

`x == self.inverted().transform(self.transform(x))`

`is_separable = False`

`output_dims = 2`

`transform_non_affine(tr)`

Performs only the non-affine part of the transformation.

`transform(values)` is always equivalent to `transform_affine(transform_non_affine(values))`.

In non-affine transformations, this is generally equivalent to `transform(values)`. In affine transformations, this is always a no-op.

Accepts a numpy array of shape `(N x input_dims)` and returns a numpy array of shape `(N x output_dims)`.

Alternatively, accepts a numpy array of length `input_dims` and returns a numpy array of length `output_dims`.

`transform_path_non_affine(path)`

Returns a path, transformed only by the non-affine part of this transform.

`path`: a `Path` instance.

`transform_path(path)` is equivalent to `transform_path_affine(transform_path_non_affine(values))`.

`class matplotlib.projections.polar.RadialAxis(*args, **kwargs)`

Bases: `matplotlib.axis.YAxis`

A radial Axis.

This overrides certain properties of a `YAxis` to provide special-casing for a radial axis.

`axis_name = 'radius'`

`cla()`

clear the current axis

`class matplotlib.projections.polar.RadialLocator(base, axes=None)`

Bases: `matplotlib.ticker.Locator`

Used to locate radius ticks.

Ensures that all ticks are strictly positive. For all other tasks, it delegates to the base `Locator` (which may be different depending on the scale of the r-axis).

`autoscale()`

autoscale the view limits

`pan(numsteps)`

Pan numticks (can be positive or negative)

`refresh()`

refresh internal information based on current lim

`view_limits(vmin, vmax)`

select a scale for the range from `vmin` to `vmax`

Normally this method is overridden by subclasses to change locator behaviour.

`zoom(direction)`

Zoom in/out on axis; if `direction` is `>0` zoom in, else zoom out

`class matplotlib.projections.polar.RadialTick(axes, loc, label, size=None, width=None, color=None, tickdir=None, pad=None, labelsize=None, labelcolor=None, zorder=None, gridOn=None, tick1On=True, tick2On=True, label1On=True, label2On=False, major=True, labelrotation=0, grid_color=None, grid_linestyle=None, grid_linewidth=None, grid_alpha=None, **kw)`

Bases: `matplotlib.axis.YTick`

A radial-axis tick.

This subclass of `YTick` provides radial ticks with some small modification to their re-positioning such that ticks are rotated based on axes limits. This results in ticks that are correctly perpendicular to the spine. Labels are also rotated to be perpendicular to the spine, when 'auto' rotation is enabled.

bbox is the Bound2D bounding box in display coords of the Axes loc is the tick location in data coords size is the tick size in points

update_position(loc)

Set the location of tick in data coords with scalar loc

class matplotlib.projections.polar.ThetaAxis(axes, pickradius=15)

Bases: matplotlib.axis.XAxis

A theta Axis.

This overrides certain properties of an XAxis to provide special-casing for an angular axis.

Init the axis with the parent Axes instance

axis_name = 'theta'

cla()

clear the current axis

class matplotlib.projections.polar.ThetaFormatter

Bases: matplotlib.ticker.Formatter

Used to format the theta tick labels. Converts the native unit of radians into degrees and adds a degree symbol.

class matplotlib.projections.polar.ThetaLocator(base)

Bases: matplotlib.ticker.Locator

Used to locate theta ticks.

This will work the same as the base locator except in the case that the view spans the entire circle. In such cases, the previously used default locations of every 45 degrees are returned.

autoscale()

autoscale the view limits

pan(numsteps)

Pan numticks (can be positive or negative)

refresh()

refresh internal information based on current lim

set_axis(axis)

view_limits(vmin, vmax)

select a scale for the range from vmin to vmax

Normally this method is overridden by subclasses to change locator behaviour.

zoom(direction)

Zoom in/out on axis; if direction is >0 zoom in, else zoom out

class matplotlib.projections.polar.ThetaTick(axes, *args, **kwargs)

Bases: matplotlib.axis.XTick

A theta-axis tick.

This subclass of XTick provides angular ticks with some small modification to their re-positioning such that ticks are rotated based on tick location. This results in ticks that are correctly perpendicular to the arc spine.

When 'auto' rotation is enabled, labels are also rotated to be parallel to the spine. The label padding is also applied here since it's not possible to use a generic axes transform to produce tick-specific padding.

`update_position(loc)`

Set the location of tick in data coords with scalar loc

61.1 matplotlib.rcsetup

The rcsetup module contains the default values and the validation code for customization using matplotlib's rc settings.

Each rc setting is assigned a default value and a function used to validate any attempted changes to that setting. The default values and validation functions are defined in the rcsetup module, and are used to construct the rcParams global object which stores the settings and is referenced throughout matplotlib.

These default values should be consistent with the default matplotlibrc file that actually reflects the values given here. Any additions or deletions to the parameter set listed here should also be visited to the matplotlibrc.template in matplotlib's root source directory.

```
class matplotlib.rcsetup.ValidateInStrings(key, valid, ignorecase=False)
```

Bases: `object`

valid is a list of legal strings

```
class matplotlib.rcsetup.ValidateInterval(vmin, vmax, closedmin=True, closedmax=True)
```

Bases: `object`

Value must be in interval

```
matplotlib.rcsetup.cycler(*args, **kwargs)
```

Creates a `Cycler` object much like `cycler.cycler()`, but includes input validation.

Call signatures:

```
cycler(cycler)
cycler(label=values[, label2=values2[, ...]])
cycler(label, values)
```

Form 1 copies a given `Cycler` object.

Form 2 creates a `Cycler` which cycles over one or more properties simultaneously. If multiple properties are given, their value lists must have the same length.

Form 3 creates a `Cycler` for a single property. This form exists for compatibility with the original cycler. Its use is discouraged in favor of the kwarg form, i.e. `cycler(label=values)`.

Parameters

`cycler` [Cycler] Copy constructor for Cycler.

`label` [str] The property key. Must be a valid Artist property. For example, 'color' or 'linestyle'. Aliases are allowed, such as 'c' for 'color' and 'lw' for 'linewidth'.

`values` [iterable] Finite-length iterable of the property values. These values are validated and will raise a `ValueError` if invalid.

Returns

`cycler` [Cycler] A new [Cycler](#) for the given properties.

Examples

Creating a cycler for a single property:

```
>>> c = cycler(color=['red', 'green', 'blue'])
```

Creating a cycler for simultaneously cycling over multiple properties (e.g. red circle, green plus, blue cross):

```
>>> c = cycler(color=['red', 'green', 'blue'],  
...            marker=['o', '+', 'x'])
```

`matplotlib.rcsetup.deprecate__axes__hold(value)`

`matplotlib.rcsetup.update__savefig__format(value)`

`matplotlib.rcsetup.validate__animation__writer__path(p)`

`matplotlib.rcsetup.validate__any(s)`

`matplotlib.rcsetup.validate__anylist(s)`

`matplotlib.rcsetup.validate__aspect(s)`

`matplotlib.rcsetup.validate__axisbelow(s)`

`matplotlib.rcsetup.validate__backend(s)`

`matplotlib.rcsetup.validate__bbox(s)`

`matplotlib.rcsetup.validate__bool(b)`
Convert `b` to a boolean or raise

`matplotlib.rcsetup.validate__bool_maybe_none(b)`
Convert `b` to a boolean or raise

`matplotlib.rcsetup.validate__capstylelist(s)`

`matplotlib.rcsetup.validate__color(s)`
return a valid color arg

`matplotlib.rcsetup.validate__color_for_prop_cycle(s)`

`matplotlib.rcsetup.validate__color_or__auto(s)`

`matplotlib.rcsetup.validate__color_or__inherit(s)`
return a valid color arg

`matplotlib.rcsetup.validate__colorlist(s)`
return a list of colorspecs

`matplotlib.rcsetup.validate__cycler(s)`
return a Cycler object from a string repr or the object itself

`matplotlib.rcsetup.validate__dashlist(s)`

`matplotlib.rcsetup.validate__dpi(s)`
confirm `s` is string 'figure' or convert `s` to float or raise

`matplotlib.rcsetup.validate__fillstylelist(s)`

`matplotlib.rcsetup.validate__float(s)`
convert `s` to float or raise

`matplotlib.rcsetup.validate__float_or_None(s)`
convert `s` to float, None or raise

`matplotlib.rcsetup.validate__floatlist(s)`
convert `s` to float or raise

`matplotlib.rcsetup.validate__font_properties(s)`

`matplotlib.rcsetup.validate__fontsize(s)`

`matplotlib.rcsetup.validate__fontsizelist(s)`

`matplotlib.rcsetup.validate__fonttype(s)`
confirm that this is a Postscript or PDF font type that we know how to convert to

`matplotlib.rcsetup.validate_hatch(s)`

Validate a hatch pattern. A hatch pattern string can have any sequence of the following characters: `\ / | - + * . x o O`.

`matplotlib.rcsetup.validate_hatchlist(s)`

Validate a hatch pattern. A hatch pattern string can have any sequence of the following characters: `\ / | - + * . x o O`.

`matplotlib.rcsetup.validate_hinting(s)`

`matplotlib.rcsetup.validate_hist_bins(s)`

`matplotlib.rcsetup.validate_int(s)`

convert `s` to int or raise

`matplotlib.rcsetup.validate_int_or_None(s)`

if not None, tries to validate as an int

`matplotlib.rcsetup.validate_joinstylelist(s)`

`matplotlib.rcsetup.validate_negative_linestyle(s)`

Deprecated since version 2.1: The `validate_negative_linestyle` function was deprecated in version 2.1. See '`validate_negative_linestyle_legacy`' deprecation warning for more information.

`matplotlib.rcsetup.validate_negative_linestyle_legacy(s)`

Deprecated since version 2.1: The `validate_negative_linestyle_legacy` function was deprecated in version 2.1. The '`contour.negative_linestyle`' rcParam now follows the same validation as the other rcParams that are related to line style.

`class matplotlib.rcsetup.validate_nseq_float(n=None, allow_none=False)`

Bases: `object`

`class matplotlib.rcsetup.validate_nseq_int(n=None)`

Bases: `object`

`matplotlib.rcsetup.validate_path_exists(s)`

If `s` is a path, return `s`, else `False`

`matplotlib.rcsetup.validate_ps_distiller(s)`

`matplotlib.rcsetup.validate_qt4(s)`

`matplotlib.rcsetup.validate_qt5(s)`

`matplotlib.rcsetup.validate_sketch(s)`

`matplotlib.rcsetup.validate_string(s)`

`matplotlib.rcsetup.validate__string__or__None(s)`
convert s to string or raise

`matplotlib.rcsetup.validate__stringlist(s)`
return a list

`matplotlib.rcsetup.validate__svg__fonttype(s)`

`matplotlib.rcsetup.validate__toolbar(s)`

`matplotlib.rcsetup.validate__webagg__address(s)`

`matplotlib.rcsetup.validate__whiskers(s)`

62.1 matplotlib.sankey

Module for creating Sankey diagrams using matplotlib

```
class matplotlib.sankey.Sankey(ax=None, scale=1.0, unit="", format='%G', gap=0.25, radius=0.1, shoulder=0.03, offset=0.15, head_angle=100, margin=0.4, tolerance=1e-06, **kwargs)
```

Bases: `object`

Sankey diagram in matplotlib

Sankey diagrams are a specific type of flow diagram, in which the width of the arrows is shown proportionally to the flow quantity. They are typically used to visualize energy or material or cost transfers between processes. [Wikipedia \(6/1/2011\)](#)

Create a new Sankey instance.

Optional keyword arguments:

Field	Description
ax	axes onto which the data should be plotted If ax isn't provided, new axes will be created.
scale	scaling factor for the flows scale sizes the width of the paths in order to maintain proper layout. The same scale is applied to all subdiagrams. The value should be chosen such that the product of the scale and the sum of the inputs is approximately 1.0 (and the product of the scale and the sum of the outputs is approximately -1.0).
unit	string representing the physical unit associated with the flow quantities If unit is None, then none of the quantities are labeled.
format	a Python number formatting string to be used in labeling the flow as a quantity (i.e., a number times a unit, where the unit is given)
gap	space between paths that break in/break away to/from the top or bottom
radius	inner radius of the vertical paths
shoulder	size of the shoulders of output arrows
offset	text offset (from the dip or tip of the arrow)
headangle	angle of the arrow heads (and negative of the angle of the tails) [deg]
margin	minimum space between Sankey outlines and the edge of the plot area
tolerance	acceptable maximum of the magnitude of the sum of flows The magnitude of the sum of connected flows cannot be greater than tolerance.

The optional arguments listed above are applied to all subdiagrams so that there is consistent alignment and formatting.

If Sankey is instantiated with any keyword arguments other than those explicitly listed above (**kwargs), they will be passed to `add()`, which will create the first subdiagram.

In order to draw a complex Sankey diagram, create an instance of Sankey by calling it without any kwargs:

```
sankey = Sankey()
```

Then add simple Sankey sub-diagrams:

```
sankey.add() # 1
sankey.add() # 2
#...
sankey.add() # n
```

Finally, create the full diagram:

```
sankey.finish()
```

Or, instead, simply daisy-chain those calls:

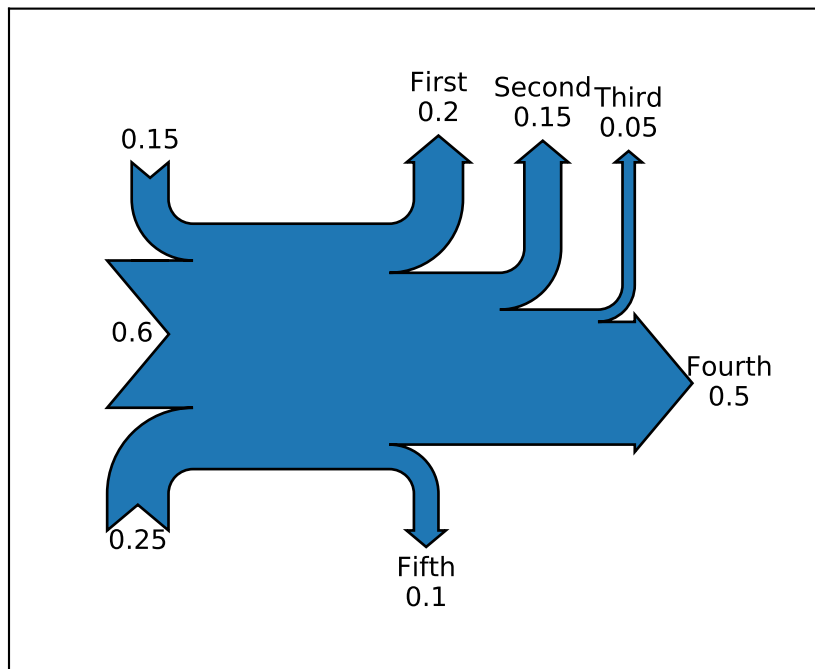
```
Sankey().add().add... .add().finish()
```

See also:

`add()` `finish()`

Examples:

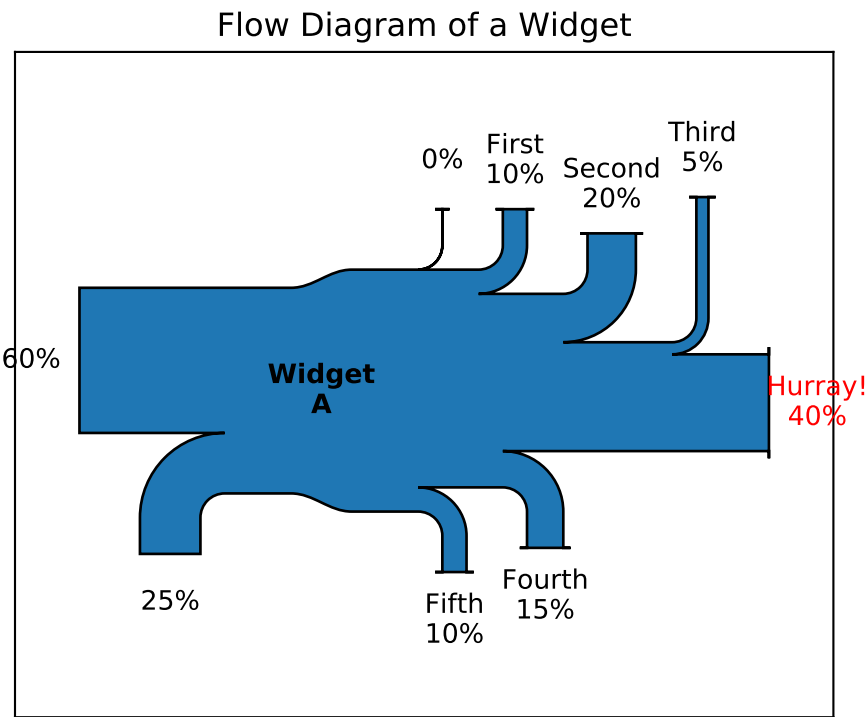
The default settings produce a diagram like this.



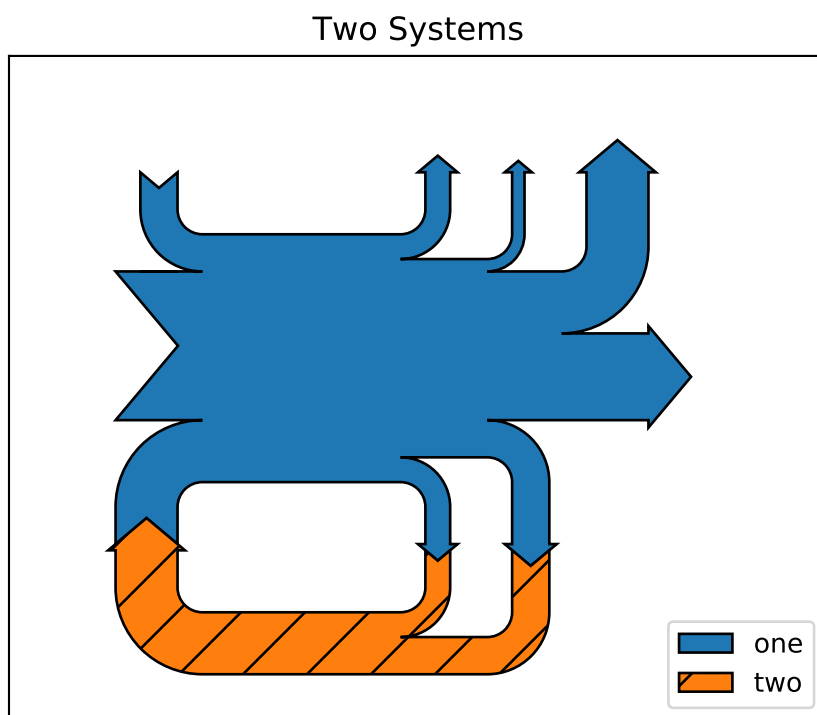
`add(patchlabel="", flows=None, orientations=None, labels="", trunklength=1.0, pathlengths=0.25, prior=None, connect=(0, 0), rotation=0, **kwargs)`
 Add a simple Sankey diagram with flows at the same hierarchical level.

Return value is the instance of `Sankey`.

Optional keyword arguments:



	Key-word	Description
	patch-label	label to be placed at the center of the diagram Note: label (not patch-label) will be passed to the patch through **kwargs and can be used to create an entry in the legend.
	flows	array of flow values By convention, inputs are positive and outputs are negative.
	orientations	list of orientations of the paths Valid values are 1 (from/to the top), 0 (from/to the left or right), or -1 (from/to the bottom). If orientations == 0, inputs will break in from the left and outputs will break away to the right.
	labels	list of specifications of the labels for the flows Each value may be None (no labels), " (just label the quantities), or a labeling string. If a single value is provided, it will be applied to all flows. If an entry is a non-empty string, then the quantity for the corresponding flow will be shown below the string. However, if the unit of the main diagram is None, then quantities are never shown, regardless of the value of this argument.
	trunk-length	length between the bases of the input and output groups
1808	pathlengths	list of lengths of the arrows before break-in or after break-away If a single value is given, then it will be applied to the first (inside) paths on the top and bottom, and the length of all other arrows will be justified accordingly. The pathlengths are not applied to the horizontal inputs and outputs.
	prior	index of the prior diagram to which this diagram should be connected



Valid kwargs are `matplotlib.patches.PathPatch()` arguments:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
an-tialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\ ' ' ' '- ' '+ ' 'x' 'o' 'O' '.' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

As examples, fill=False and label='A legend entry'. By default, facecolor='#bfd1d4' (light blue) and linewidth=0.5.

The indexing parameters (prior and connect) are zero-based.

The flows are placed along the top of the diagram from the inside out in order of their index within the flows list or array. They are placed along the sides of the diagram from the top down and along the bottom from the outside in.

If the sum of the inputs and outputs is nonzero, the discrepancy will appear as a cubic Bezier curve along the top and bottom edges of the trunk.

See also:

`finish()`

`finish()`

Adjust the axes and return a list of information about the Sankey subdiagram(s).

Return value is a list of subdiagrams represented with the following fields:

Field	Description
<code>patch</code>	Sankey outline (an instance of <code>PathPatch</code>)
<code>flows</code>	values of the flows (positive for input, negative for output)
<code>angles</code>	list of angles of the arrows [deg/90] For example, if the diagram has not been rotated, an input to the top side will have an angle of 3 (DOWN), and an output from the top side will have an angle of 1 (UP). If a flow has been skipped (because its magnitude is less than tolerance), then its angle will be <code>None</code> .
<code>tips</code>	array in which each row is an [x, y] pair indicating the positions of the tips (or "dips") of the flow paths If the magnitude of a flow is less the tolerance for the instance of <code>Sankey</code> , the flow is skipped and its tip will be at the center of the diagram.
<code>text</code>	Text instance for the label of the diagram
<code>texts</code>	list of Text instances for the labels of flows

See also:

`add()`

63.1 matplotlib.scale

class matplotlib.scale.InvertedLog10Transform(shorthand__name=None)

Bases: matplotlib.scale.InvertedLogTransformBase

Creates a new TransformNode.

Parameters

shorthand__name [str] A string representing the "name" of the transform.
The name carries no significance other than to improve the readability
of str(transform) when DEBUG=True.

base = 10.0

inverted()

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does
not cause a corresponding update to its inverted copy.

x === self.inverted().transform(self.transform(x))

class matplotlib.scale.InvertedLog2Transform(shorthand__name=None)

Bases: matplotlib.scale.InvertedLogTransformBase

Creates a new TransformNode.

Parameters

shorthand__name [str] A string representing the "name" of the transform.
The name carries no significance other than to improve the readability
of str(transform) when DEBUG=True.

base = 2.0

inverted()

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

```
x === self.inverted().transform(self.transform(x))
```

```
class matplotlib.scale.InvertedLogTransform(base)
```

```
    Bases: matplotlib.scale.InvertedLogTransformBase
```

```
    inverted()
```

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

```
x === self.inverted().transform(self.transform(x))
```

```
class matplotlib.scale.InvertedLogTransformBase(shorthand_name=None)
```

```
    Bases: matplotlib.transforms.Transform
```

Creates a new TransformNode.

Parameters

shorthand_name [str] A string representing the "name" of the transform.
The name carries no significance other than to improve the readability
of str(transform) when DEBUG=True.

```
has_inverse = True
```

```
input_dims = 1
```

```
is_separable = True
```

```
output_dims = 1
```

```
transform_non_affine(a)
```

Performs only the non-affine part of the transformation.

transform(values) is always equivalent to transform_affine(transform_non_affine(values)).

In non-affine transformations, this is generally equivalent to transform(values). In affine transformations, this is always a no-op.

Accepts a numpy array of shape (N x input_dims) and returns a numpy array of shape (N x output_dims).

Alternatively, accepts a numpy array of length input_dims and returns a numpy array of length output_dims.

```
class matplotlib.scale.InvertedNaturalLogTransform(shorthand_name=None)
```

```
    Bases: matplotlib.scale.InvertedLogTransformBase
```

Creates a new TransformNode.

Parameters

`shorthand_name` [str] A string representing the "name" of the transform. The name carries no significance other than to improve the readability of `str(transform)` when `DEBUG=True`.

`base` = 2.718281828459045

`inverted()`

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

`x` == `self.inverted().transform(self.transform(x))`

`class matplotlib.scale.InvertedSymmetricalLogTransform(base, linthresh, linscale)`

Bases: `matplotlib.transforms.Transform`

`has_inverse` = True

`input_dims` = 1

`inverted()`

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

`x` == `self.inverted().transform(self.transform(x))`

`is_separable` = True

`output_dims` = 1

`transform_non_affine(a)`

Performs only the non-affine part of the transformation.

`transform(values)` is always equivalent to `transform_affine(transform_non_affine(values))`.

In non-affine transformations, this is generally equivalent to `transform(values)`. In affine transformations, this is always a no-op.

Accepts a numpy array of shape (N x `input_dims`) and returns a numpy array of shape (N x `output_dims`).

Alternatively, accepts a numpy array of length `input_dims` and returns a numpy array of length `output_dims`.

`class matplotlib.scale.LinearScale(axis, **kwargs)`

Bases: `matplotlib.scale.ScaleBase`

The default linear scale.

`get_transform()`

The transform for linear scaling is just the IdentityTransform.

`name = 'linear'`

`set_default_locators_and_formatters(axis)`

Set the locators and formatters to reasonable defaults for linear scaling.

`class matplotlib.scale.Log10Transform(nonpos='clip')`

Bases: `matplotlib.scale.LogTransformBase`

`base = 10.0`

`inverted()`

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

`x === self.inverted().transform(self.transform(x))`

`class matplotlib.scale.Log2Transform(nonpos='clip')`

Bases: `matplotlib.scale.LogTransformBase`

`base = 2.0`

`inverted()`

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

`x === self.inverted().transform(self.transform(x))`

`class matplotlib.scale.LogScale(axis, **kwargs)`

Bases: `matplotlib.scale.ScaleBase`

A standard logarithmic scale. Care is taken so non-positive values are not plotted.

For computational efficiency (to push as much as possible to Numpy C code in the common cases), this scale provides different transforms depending on the base of the logarithm:

- base 10 (`Log10Transform`)
- base 2 (`Log2Transform`)
- base e (`NaturalLogTransform`)
- arbitrary base (`LogTransform`)

`basex/basey`: The base of the logarithm

`nonposx/nonposy`: `['mask' | 'clip']` non-positive values in x or y can be masked as invalid, or clipped to a very small positive number

subsx/subsy: Where to place the subticks between each major tick. Should be a sequence of integers. For example, in a log10 scale: [2, 3, 4, 5, 6, 7, 8, 9]

will place 8 logarithmically spaced minor ticks between each major tick.

class InvertedLog10Transform(shorthand_name=None)

Bases: matplotlib.scale.InvertedLogTransformBase

Creates a new TransformNode.

Parameters

shorthand_name [str] A string representing the "name" of the transform.
The name carries no significance other than to improve the readability of
str(transform) when DEBUG=True.

base = 10.0

inverted()

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self
does not cause a corresponding update to its inverted copy.

x === self.inverted().transform(self.transform(x))

class InvertedLog2Transform(shorthand_name=None)

Bases: matplotlib.scale.InvertedLogTransformBase

Creates a new TransformNode.

Parameters

shorthand_name [str] A string representing the "name" of the transform.
The name carries no significance other than to improve the readability of
str(transform) when DEBUG=True.

base = 2.0

inverted()

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self
does not cause a corresponding update to its inverted copy.

x === self.inverted().transform(self.transform(x))

class InvertedLogTransform(base)

Bases: matplotlib.scale.InvertedLogTransformBase

inverted()

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self
does not cause a corresponding update to its inverted copy.

```
x === self.inverted().transform(self.transform(x))
```

```
class InvertedNaturalLogTransform(shorthand_name=None)
```

```
Bases: matplotlib.scale.InvertedLogTransformBase
```

Creates a new TransformNode.

Parameters

shorthand_name [str] A string representing the "name" of the transform.
The name carries no significance other than to improve the readability of
str(transform) when DEBUG=True.

```
base = 2.718281828459045
```

```
inverted()
```

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self
does not cause a corresponding update to its inverted copy.

```
x === self.inverted().transform(self.transform(x))
```

```
class Log10Transform(nonpos='clip')
```

```
Bases: matplotlib.scale.LogTransformBase
```

```
base = 10.0
```

```
inverted()
```

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self
does not cause a corresponding update to its inverted copy.

```
x === self.inverted().transform(self.transform(x))
```

```
class Log2Transform(nonpos='clip')
```

```
Bases: matplotlib.scale.LogTransformBase
```

```
base = 2.0
```

```
inverted()
```

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self
does not cause a corresponding update to its inverted copy.

```
x === self.inverted().transform(self.transform(x))
```

```
class LogTransform(base, nonpos='clip')
```

```
Bases: matplotlib.scale.LogTransformBase
```

```
inverted()
```

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

```
x === self.inverted().transform(self.transform(x))
```

```
class LogTransformBase(nonpos='clip')
```

```
    Bases: matplotlib.transforms.Transform
```

```
    has_inverse = True
```

```
    input_dims = 1
```

```
    is_separable = True
```

```
    output_dims = 1
```

```
    transform_non_affine(a)
```

Performs only the non-affine part of the transformation.

transform(values) is always equivalent to transform_affine(transform_non_affine(values)).

In non-affine transformations, this is generally equivalent to transform(values). In affine transformations, this is always a no-op.

Accepts a numpy array of shape (N x input_dims) and returns a numpy array of shape (N x output_dims).

Alternatively, accepts a numpy array of length input_dims and returns a numpy array of length output_dims.

```
class NaturalLogTransform(nonpos='clip')
```

```
    Bases: matplotlib.scale.LogTransformBase
```

```
    base = 2.718281828459045
```

```
    inverted()
```

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

```
x === self.inverted().transform(self.transform(x))
```

```
    get_transform()
```

Return a Transform instance appropriate for the given logarithm base.

```
    limit_range_for_scale(vmin, vmax, minpos)
```

Limit the domain to positive values.

```
    name = 'log'
```

`set_default_locators_and_formatters(axis)`

Set the locators and formatters to specialized versions for log scaling.

`class matplotlib.scale.LogTransform(base, nonpos='clip')`

Bases: `matplotlib.scale.LogTransformBase`

`inverted()`

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

`x === self.inverted().transform(self.transform(x))`

`class matplotlib.scale.LogTransformBase(nonpos='clip')`

Bases: `matplotlib.transforms.Transform`

`has_inverse = True`

`input_dims = 1`

`is_separable = True`

`output_dims = 1`

`transform_non_affine(a)`

Performs only the non-affine part of the transformation.

`transform(values)` is always equivalent to `transform_affine(transform_non_affine(values))`.

In non-affine transformations, this is generally equivalent to `transform(values)`. In affine transformations, this is always a no-op.

Accepts a numpy array of shape (N x `input_dims`) and returns a numpy array of shape (N x `output_dims`).

Alternatively, accepts a numpy array of length `input_dims` and returns a numpy array of length `output_dims`.

`class matplotlib.scale.LogisticTransform(nonpos='mask')`

Bases: `matplotlib.transforms.Transform`

`has_inverse = True`

`input_dims = 1`

`inverted()`

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

```

        x == self.inverted().transform(self.transform(x))

is_separable = True

output_dims = 1

transform_non_affine(a)
    logistic transform (base 10)
class matplotlib.scale.LogitScale(axis, nonpos='mask')
    Bases: matplotlib.scale.ScaleBase
    Logit scale for data between zero and one, both excluded.
    This scale is similar to a log scale close to zero and to one, and almost linear around 0.5. It
    maps the interval ]0, 1[ onto ]-infty, +infty[.
    nonpos: ['mask' | 'clip' ] values beyond ]0, 1[ can be masked as invalid, or clipped to a num-
        ber very close to 0 or 1
    get_transform()
        Return a LogitTransform instance.
    limit_range_for_scale(vmin, vmax, minpos)
        Limit the domain to values between 0 and 1 (excluded).
    name = 'logit'

    set_default_locators_and_formatters(axis)
        Set the Locator and Formatter objects on the given axis to match this scale.
class matplotlib.scale.LogitTransform(nonpos='mask')
    Bases: matplotlib.transforms.Transform
    has_inverse = True

    input_dims = 1

    inverted()
        Return the corresponding inverse transformation.
        The return value of this method should be treated as temporary. An update to self does
        not cause a corresponding update to its inverted copy.
        x == self.inverted().transform(self.transform(x))
    is_separable = True

    output_dims = 1

```

```
transform_non_affine(a)
    logit transform (base 10), masked or clipped
```

```
class matplotlib.scale.NaturalLogTransform(nonpos='clip')
    Bases: matplotlib.scale.LogTransformBase

    base = 2.718281828459045
```

```
inverted()
    Return the corresponding inverse transformation.
```

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

```
x === self.inverted().transform(self.transform(x))
```

```
class matplotlib.scale.ScaleBase
    Bases: object
```

The base class for all scales.

Scales are separable transformations, working on a single dimension.

Any subclasses will want to override:

- `name`
- `get_transform()`
- `set_default_locators_and_formatters()`

And optionally:

- `limit_range_for_scale()`

```
get_transform()
    Return the Transform object associated with this scale.
```

```
limit_range_for_scale(vmin, vmax, minpos)
    Returns the range vmin, vmax, possibly limited to the domain supported by this scale.

    minpos should be the minimum positive value in the data. This is used by log scales to
    determine a minimum value.
```

```
set_default_locators_and_formatters(axis)
    Set the Locator and Formatter objects on the given axis to match this scale.
```

```
class matplotlib.scale.SymmetricalLogScale(axis, **kwargs)
    Bases: matplotlib.scale.ScaleBase
```

The symmetrical logarithmic scale is logarithmic in both the positive and negative directions from the origin.

Since the values close to zero tend toward infinity, there is a need to have a range around zero that is linear. The parameter `linthresh` allows the user to specify the size of this range (`-linthresh, linthresh`).

`basex/basey`: The base of the logarithm

`linthreshx/linthreshy`: A single float which defines the range $(-x, x)$, within which the plot is linear. This avoids having the plot go to infinity around zero.

`subsx/subsy`: Where to place the subticks between each major tick. Should be a sequence of integers. For example, in a log10 scale: `[2, 3, 4, 5, 6, 7, 8, 9]`

will place 8 logarithmically spaced minor ticks between each major tick.

`linscalex/linscaley`: This allows the linear range $(-linthresh, linthresh)$ to be stretched relative to the logarithmic range. Its value is the number of decades to use for each half of the linear range. For example, when `linscale == 1.0` (the default), the space used for the positive and negative halves of the linear range will be equal to one decade in the logarithmic range.

class `InvertedSymmetricalLogTransform(base, linthresh, linscale)`

Bases: `matplotlib.transforms.Transform`

`has_inverse = True`

`input_dims = 1`

`inverted()`

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

`x == self.inverted().transform(self.transform(x))`

`is_separable = True`

`output_dims = 1`

`transform_non_affine(a)`

Performs only the non-affine part of the transformation.

`transform(values)` is always equivalent to `transform_affine(transform_non_affine(values))`.

In non-affine transformations, this is generally equivalent to `transform(values)`. In affine transformations, this is always a no-op.

Accepts a numpy array of shape $(N \times \text{input_dims})$ and returns a numpy array of shape $(N \times \text{output_dims})$.

Alternatively, accepts a numpy array of length `input_dims` and returns a numpy array of length `output_dims`.

class `SymmetricalLogTransform(base, linthresh, linscale)`

Bases: `matplotlib.transforms.Transform`

has_inverse = True

input_dims = 1

inverted()

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

x === self.inverted().transform(self.transform(x))

is_separable = True

output_dims = 1

transform_non_affine(a)

Performs only the non-affine part of the transformation.

transform(values) is always equivalent to transform_affine(transform_non_affine(values)).

In non-affine transformations, this is generally equivalent to transform(values). In affine transformations, this is always a no-op.

Accepts a numpy array of shape (N x input_dims) and returns a numpy array of shape (N x output_dims).

Alternatively, accepts a numpy array of length input_dims and returns a numpy array of length output_dims.

get_transform()

Return a SymmetricalLogTransform instance.

name = 'symlog'

set_default_locators_and_formatters(axis)

Set the locators and formatters to specialized versions for symmetrical log scaling.

class matplotlib.scale.SymmetricalLogTransform(base, linthresh, linscale)

Bases: matplotlib.transforms.Transform

has_inverse = True

input_dims = 1

inverted()

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

```
x === self.inverted().transform(self.transform(x))
```

```
is_separable = True
```

```
output_dims = 1
```

```
transform_non_affine(a)
```

Performs only the non-affine part of the transformation.

transform(values) is always equivalent to transform_affine(transform_non_affine(values)).

In non-affine transformations, this is generally equivalent to transform(values). In affine transformations, this is always a no-op.

Accepts a numpy array of shape (N x input_dims) and returns a numpy array of shape (N x output_dims).

Alternatively, accepts a numpy array of length input_dims and returns a numpy array of length output_dims.

```
matplotlib.scale.get_scale_docs()
```

Helper function for generating docstrings related to scales.

```
matplotlib.scale.get_scale_names()
```

```
matplotlib.scale.register_scale(scale_class)
```

Register a new kind of scale.

scale_class must be a subclass of ScaleBase.

```
matplotlib.scale.scale_factory(scale, axis, **kwargs)
```

Return a scale class by name.

ACCEPTS: [linear | log | logit | symlog]

64.1 matplotlib.spines

`class matplotlib.spines.Spine(axes, spine_type, path, **kwargs)`
Bases: `matplotlib.patches.Patch`

an axis spine – the line noting the data area boundaries

Spines are the lines connecting the axis tick marks and noting the boundaries of the data area. They can be placed at arbitrary positions. See `function:set__position` for more information.

The default position is ('outward',0).

Spines are subclasses of `class:Patch`, and inherit much of their behavior.

Spines draw a line, a circle, or an arc depending if `function:set_patch_line`, `function:set_patch_circle`, or `function:set_patch_arc` has been called. Line-like is the default.

- `axes` : the `Axes` instance containing the spine
- `spine_type` : a string specifying the spine type
- `path` : the `path` instance used to draw the spine

Valid kwargs are:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\ ' ' ' '- ' '+ ' 'x' 'o' 'O' '?' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

classmethod `arc_spine`(axes, spine_type, center, radius, theta1, theta2, **kwargs)
(classmethod) Returns an arc Spine.

classmethod `circular_spine`(axes, center, radius, **kwargs)
(staticmethod) Returns a circular Spine.

`cla()`
Clear the current spine

`draw`(renderer)

Draw the Patch to the given renderer.

`get_bounds()`

Get the bounds of the spine.

`get_patch_transform()`

Return the Transform instance which takes patch coordinates to data coordinates.

For example, one may define a patch of a circle which represents a radius of 5 by providing coordinates for a unit circle, and a transform which scales the coordinates (the patch coordinate) by 5.

`get_path()`

Return the path of this patch

`get_position()`

get the spine position

`get_smart_bounds()`

get whether the spine has smart bounds

`get_spine_transform()`

get the spine transform

`is_frame_like()`

return True if directly on axes frame

This is useful for determining if a spine is the edge of an old style MPL plot. If so, this function will return True.

`classmethod linear_spine(axes, spine_type, **kwargs)`

(staticmethod) Returns a linear Spine.

`register_axis(axis)`

register an axis

An axis should be registered with its corresponding spine from the Axes instance. This allows the spine to clear any axis properties when needed.

`set_bounds(low, high)`

Set the bounds of the spine.

`set_color(c)`

Set the edgecolor.

ACCEPTS: matplotlib color arg or sequence of rgba tuples

See also:

`set_facecolor()`, `set_edgecolor()` For setting the edge or face color individually.

`set_patch_arc(center, radius, theta1, theta2)`

set the spine to be arc-like

`set_patch_circle(center, radius)`

set the spine to be circular

`set_patch_line()`
set the spine to be linear

`set_position(position)`
set the position of the spine

Spine position is specified by a 2 tuple of (position type, amount). The position types are:

- 'outward' : place the spine out from the data area by the specified number of points. (Negative values specify placing the spine inward.)
- 'axes' : place the spine at the specified Axes coordinate (from 0.0-1.0).
- 'data' : place the spine at the specified data coordinate.

Additionally, shorthand notations define a special positions:

- 'center' -> ('axes', 0.5)
- 'zero' -> ('data', 0.0)

`set_smart_bounds(value)`
set the spine and associated axis to have smart bounds

65.1 matplotlib.style

`matplotlib.style.context(style, after_reset=False)`

Context manager for using style settings temporarily.

Parameters

`style` [str, dict, or list] A style specification. Valid options are:

str	The name of a style or a path/URL to a style file. For a list of available style names, see <code>style.available</code> .
dict	Dictionary with valid key/value pairs for <code>matplotlib.rcParams</code> .
list	A list of style specifiers (str or dict) applied from first to last in the list.

`after_reset` [bool] If True, apply style after resetting settings to their defaults; otherwise, apply style on top of the current settings.

`matplotlib.style.reload_library()`

Reload style library.

`matplotlib.style.use(style)`

Use matplotlib style settings from a style specification.

The style name of 'default' is reserved for reverting back to the default style settings.

Parameters

`style` [str, dict, or list] A style specification. Valid options are:

str	The name of a style or a path/URL to a style file. For a list of available style names, see <code>style.available</code> .
dict	Dictionary with valid key/value pairs for <code>matplotlib.rcParams</code> .
list	A list of style specifiers (str or dict) applied from first to last in the list.

`matplotlib.style.library`

Dictionary of available styles

matplotlib.style.available
List of available styles

TABLE

66.1 matplotlib.table

Place a table below the x-axis at location loc.

The table consists of a grid of cells.

The grid need not be rectangular and can have holes.

Cells are added by specifying their row and column.

For the purposes of positioning the cell at (0, 0) is assumed to be at the top left and the cell at (max_row, max_col) is assumed to be at bottom right.

You can add additional cells outside this range to have convenient ways of positioning more interesting grids.

Author : John Gill <jng@europe.renre.com> Copyright : 2004 John Gill and John Hunter License : matplotlib license

```
class matplotlib.table.Cell(xy, width, height, edgecolor='k', facecolor='w', fill=True, text="",
                           loc=None, fontproperties=None)
```

Bases: matplotlib.patches.Rectangle

A cell is a Rectangle with some associated text.

PAD = 0.1

```
auto_set_font_size(renderer)
    Shrink font size until text fits.
```

```
draw(renderer)
    Draw the Patch to the given renderer.
```

```
get_fontsize()
    Return the cell fontsize
```

```
get_required_width(renderer)
    Get width required for this cell.
```

```
get_text()
    Return the cell Text intance
```

`get_text_bounds(renderer)`
Get text bounds in axes co-ordinates.

`set_figure(fig)`
Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`set_fontsize(size)`

`set_text_props(**kwargs)`
update the text properties with kwargs

`set_transform(trans)`
Set the artist transform.

Parameters

`t` [Transform]

`class matplotlib.table.CustomCell(*args, **kwargs)`
Bases: `matplotlib.table.Cell`

A subclass of `Cell` where the sides may be visibly toggled.

`get_path()`
Return a path where the edges specified by `__visible_edges` are drawn

`visible_edges`

`class matplotlib.table.Table(ax, loc=None, bbox=None, **kwargs)`
Bases: `matplotlib.artist.Artist`

Create a table of cells.

Table can have (optional) row and column headers.

Each entry in the table can be either text or patches.

Column widths and row heights for the table can be specified.

Return value is a sequence of text, line and patch instances that make up the table

`AXESPAD = 0.02`

`FONTSIZE = 10`

`add_cell(row, col, *args, **kwargs)`
Add a cell to the table.

Parameters

`row` [int] Row index

`col [int]` Column index

Returns

‘CustomCell’: Automatically created cell

`auto_set_column_width(col)`

Given column indexes in either List, Tuple or int. Will be able to automatically set the columns into optimal sizes.

Here is the example of the input, which trigger automatic adjustment on columns to optimal size by given index numbers. -1: the row labling 0: the 1st column 1: the 2nd column

Args: `col(List)`: list of indexes >>>`table.auto_set_column_width([-1,0,1])`

`col(Tuple)`: tuple of indexes >>>`table.auto_set_column_width((-1,0,1))`

`col(int)`: index integer >>>`table.auto_set_column_width(-1)` >>>`table.auto_set_column_width(0)` >>>`table.auto_set_column_width(1)`

`auto_set_font_size(value=True)`

Automatically set font size.

`codes = {'best': 0, 'bottom': 17, 'bottom left': 12, 'bottom right': 13, 'center': 9, 'center left': 5, 'center right': 11}`

`contains(mouseevent)`

Test whether the mouse event occurred in the table.

Returns T/F, {}

`draw(renderer)`

Derived classes drawing method

`edges`

`get_celld()`

return a dict of cells in the table

`get_child_artists()`

Return the Artists contained by the table

`get_children()`

Return the Artists contained by the table

`get_window_extent(renderer)`

Return the bounding box of the table in window coords

`scale(xscale, yscale)`

Scale column widths by xscale and row heights by yscale.

`set_fontsize(size)`

Set the fontsize of the cell text

ACCEPTS: a float in points

```
matplotlib.table.table(cellText=None, cellColours=None, cellLoc='right', colWidths=None,  
                        rowLabels=None, rowColours=None, rowLoc='left', colLabels=None,  
                        colColours=None, colLoc='center', loc='bottom', bbox=None,  
                        edges='closed')
```

Factory function to generate a Table instance.

Thanks to John Gill for providing the class and table.

67.1 matplotlib.text

Classes for including text in a figure.

```
class matplotlib.text.Annotation(s, xy, xytext=None, xycoords='data', textcoords=None, ar-
                                rowprops=None, annotation_clip=None, **kwargs)
```

Bases: matplotlib.text.Text, matplotlib.text._AnnotationBase

Annotate the point `xy` with text `s`.

Additional kwargs are passed to Text.

Parameters

`s` [str] The text of the annotation

`xy` [iterable] Length 2 sequence specifying the (x,y) point to annotate

`xytext` [iterable, optional] Length 2 sequence specifying the (x,y) to place the text at. If None, defaults to `xy`.

`xycoords` [str, Artist, Transform, callable or tuple, optional] The coordinate system that `xy` is given in.

For a `str` the allowed values are:

Property	Description
'figure points'	points from the lower left of the figure
'figure pixels'	pixels from the lower left of the figure
'figure fraction'	fraction of figure from lower left
'axes points'	points from lower left corner of axes
'axes pixels'	pixels from lower left corner of axes
'axes fraction'	fraction of axes from lower left
'data'	use the coordinate system of the object being annotated (default)
'polar'	(theta,r) if not native 'data' coordinates

If a Artist object is passed in the units are fraction if it's bounding box.

If a Transform object is passed in use that to transform xy to screen coordinates

If a callable it must take a RendererBase object as input and return a Transform or Bbox object

If a [tuple](#) must be length 2 tuple of str, Artist, Transform or callable objects. The first transform is used for the x coordinate and the second for y.

See [Advanced Annotation](#) for more details.

Defaults to 'data'

`textcoords` [str, Artist, Transform, callable or tuple, optional] The coordinate system that xytext is given, which may be different than the coordinate system used for xy.

All xycoords values are valid as well as the following strings:

Property	Description
'offset points'	offset (in points) from the xy value
'offset pixels'	offset (in pixels) from the xy value

defaults to the input of xycoords

`arrowprops` [dict, optional] If not None, properties used to draw a FancyArrowPatch arrow between xy and xytext.

If arrowprops does not contain the key 'arrowstyle' the allowed keys are:

Key	Description
width	the width of the arrow in points
headwidth	the width of the base of the arrow head in points
headlength	the length of the arrow head in points
shrink	fraction of total length to 'shrink' from both ends
?	any key to matplotlib.patches.FancyArrowPatch

If the arrowprops contains the key 'arrowstyle' the above keys are forbidden. The allowed values of 'arrowstyle' are:

Name	Attrs
'-'	None
'->'	head_length=0.4,head_width=0.2
'-['	widthB=1.0,lengthB=0.2,angleB=None
' -'	widthA=1.0,widthB=1.0
'-> '	head_length=0.4,head_width=0.2
'<-'	head_length=0.4,head_width=0.2
'<->'	head_length=0.4,head_width=0.2
'< -'	head_length=0.4,head_width=0.2
'< -> '	head_length=0.4,head_width=0.2
'fancy'	head_length=0.4,head_width=0.4,tail_width=0.4
'simple'	head_length=0.5,head_width=0.5,tail_width=0.2
'wedge'	tail_width=0.3,shrink_factor=0.5

Valid keys for FancyArrowPatch are:

Key	Description
arrowstyle	the arrow style
connectionstyle	the connection style
relpos	default is (0.5, 0.5)
patchA	default is bounding box of the text
patchB	default is None
shrinkA	default is 2 points
shrinkB	default is 2 points
mutation_scale	default is text size (in points)
mutation_aspect	default is 1.
?	any key for matplotlib.patches.PathPatch

Defaults to None

`annotation_clip` [bool, optional] Controls the visibility of the annotation when it goes outside the axes area.

If **True**, the annotation will only be drawn when the `xy` is inside the axes.

If **False**, the annotation will always be drawn regardless of its position.

The default is **None**, which behave as **True** only if `xycoords` is "data".

Returns

Annotation

`anncoords`

`contains(event)`

Test whether the mouse event occurred in the patch.

In the case of text, a hit is true anywhere in the axis-aligned bounding-box containing the text.

Returns True or False.

`draw(renderer)`

Draw the Annotation object to the given renderer.

`get_window_extent(renderer=None)`

Return a Bbox object bounding the text and arrow annotation, in display units.

renderer defaults to the `_renderer` attribute of the text object. This is not assigned until the first execution of `draw()`, so you must use this kwarg if you want to call `get_window_extent()` prior to the first `draw()`. For getting web page regions, it is simpler to call the method after saving the figure. The dpi used defaults to `self.figure.dpi`; the renderer dpi is irrelevant.

`set_figure(fig)`

Set the Figure instance the artist belongs to.

Parameters

`fig` [Figure]

`update_positions(renderer)`

”Update the pixel positions of the annotated point and the text.

xyann

`class matplotlib.text.OffsetFrom(artist, ref_coord, unit='points')`

Bases: `object`

Callable helper class for working with Annotation

Parameters

`artist` [Artist, BboxBase, or Transform] The object to compute the offset from.

`ref_coord` [length 2 sequence] If artist is an Artist or BboxBase, this values is the location to of the offset origin in fractions of the artist bounding box.

If artist is a transform, the offset origin is the transform applied to this value.

`unit` [{‘points’, ‘pixels’}] The screen units to use (pixels or points) for the offset input.

`get_unit()`

The unit for input to the transform used by `__call__`

`set_unit(unit)`

The unit for input to the transform used by `__call__`

Parameters

`unit` [{‘points’, ‘pixels’}]

```
class matplotlib.text.Text(x=0, y=0, text="", color=None, verticalalignment='baseline',
                           horizontalalignment='left', multialignment=None, font-
                           properties=None, rotation=None, linespacing=None, rota-
                           tion_mode=None, usetex=None, wrap=False, **kwargs)
```

Bases: matplotlib.artist.Artist

Handle storing and drawing of text in window or data coordinates.

Create a Text instance at x, y with string text.

Valid kwargs are

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value
alpha	float (0.0 transparent through 1.0 opaque)
animated	bool
backgroundcolor	any matplotlib color
bbox	FancyBboxPatch prop dict
clip_box	a matplotlib.transforms.Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	any matplotlib color
contains	a callable function
family or fontfamily or fontname or name	[FONTNAME 'serif' 'sans-serif' 'cursive' 'fantasy' 'monospace']
figure	a Figure instance
fontproperties or font_properties	a matplotlib.font_manager.FontProperties instance
gid	an id string
horizontalalignment or ha	['center' 'right' 'left']
label	object
linespacing	float (multiple of font size)
multialignment or ma	['left' 'right' 'center']
path_effects	AbstractPathEffect
picker	[None bool float callable]
position	(x,y)
rasterized	bool or None
rotation	[angle in degrees 'vertical' 'horizontal']
rotation_mode	[None "default" "anchor"]
size or fontsize	[size in points 'xx-small' 'x-small' 'small' 'medium' 'large' 'x-large']
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
stretch or fontstretch	[a numeric value in range 0-1000 'ultra-condensed' 'extra-condensed']
style or fontstyle	['normal' 'italic' 'oblique']
text	string or anything printable with '%s' conversion.
transform	Transform
url	a url string
usetex	bool or None
variant or fontvariant	['normal' 'small-caps']

Continued on next page

Table 1 – continued from previous page

Property	Description
verticalalignment or va	['center' 'top' 'bottom' 'baseline']
visible	bool
weight or fontweight	[a numeric value in range 0-1000 'ultralight' 'light' 'normal' 'r
wrap	bool
x	float
y	float
zorder	float

`contains(mouseevent)`
Test whether the mouse event occurred in the patch.

In the case of text, a hit is true anywhere in the axis-aligned bounding-box containing the text.

Returns True or False.

`draw(renderer)`
Draws the Text object to the given renderer.

`get_bbox_patch()`
Return the bbox Patch object. Returns None if the FancyBboxPatch is not made.

`get_color()`
Return the color of the text

`get_family()`
Return the list of font families used for font lookup

`get_font_properties()`
alias for `get_fontproperties`

`get_fontfamily()`
alias for `get_family`

`get_fontname()`
alias for `get_name`

`get_fontproperties()`
Return the FontProperties object

`get_fontsize()`
alias for `get_size`

`get_fontstretch()`
alias for `get_stretch`

`get_fontstyle()`
alias for `get_style`

`get_fontvariant()`
alias for `get_variant`

`get_fontweight()`
alias for `get_weight`

`get_ha()`
alias for `get_horizontalalignment`

`get_horizontalalignment()`
Return the horizontal alignment as string. Will be one of 'left', 'center' or 'right'.

`get_name()`
Return the font name as string

`get_position()`
Return the position of the text as a tuple (x, y)

`get_prop_tup(renderer=None)`
Return a hashable tuple of properties.

Not intended to be human readable, but useful for backends who want to cache derived information about text (e.g., layouts) and need to know if the text has changed.

`get_rotation()`
return the text angle as float in degrees

`get_rotation_mode()`
get text rotation mode

`get_size()`
Return the font size as integer

`get_stretch()`
Get the font stretch as a string or number

`get_style()`
Return the font style as string

`get_text()`
Get the text as string

`get_unitless_position()`
Return the unitless position of the text as a tuple (x, y)

`get_usetex()`
Return whether this Text object uses TeX for rendering.

If the user has not manually set this value, it defaults to `rcParams["text.usetex"]`.

`get_va()`
alias for `getverticalalignment()`

`get_variant()`
Return the font variant as a string

`get_verticalalignment()`
Return the vertical alignment as string. Will be one of 'top', 'center', 'bottom' or 'baseline'.

`get_weight()`

Get the font weight as string or number

`get_window_extent(renderer=None, dpi=None)`

Return a Bbox object bounding the text, in display units.

In addition to being used internally, this is useful for specifying clickable regions in a png file on a web page.

`renderer` defaults to the `__renderer` attribute of the text object. This is not assigned until the first execution of `draw()`, so you must use this kwarg if you want to call `get_window_extent()` prior to the first `draw()`. For getting web page regions, it is simpler to call the method after saving the figure.

`dpi` defaults to `self.figure.dpi`; the `renderer dpi` is irrelevant. For the web application, if `figure.dpi` is not the value used when saving the figure, then the value that was used must be specified as the `dpi` argument.

`get_wrap()`

Returns the wrapping state for the text.

`static is_math_text(s, usetex=None)`

Returns a cleaned string and a boolean flag. The flag indicates if the given string `s` contains any mathtext, determined by counting unescaped dollar signs. If no mathtext is present, the cleaned string has its dollar signs unescaped. If `usetex` is on, the flag always has the value "TeX".

`set_backgroundcolor(color)`

Set the background color of the text by updating the bbox.

See also:

`set_bbox()` To change the position of the bounding box.

ACCEPTS: any matplotlib color

`set_bbox(rectprops)`

Draw a bounding box around self. `rectprops` are any settable properties for a `FancyBboxPatch`, e.g., `facecolor='red'`, `alpha=0.5`.

`t.set_bbox(dict(facecolor='red', alpha=0.5))`

The default `boxstyle` is 'square'. The mutation scale of the `FancyBboxPatch` is set to the `fontsize`.

ACCEPTS: `FancyBboxPatch` prop dict

`set_clip_box(clipbox)`

Set the artist's clip Bbox.

ACCEPTS: a `matplotlib.transforms.Bbox` instance

`set_clip_on(b)`

Set whether artist uses clipping.

When False, artists will be visible outside of the axes, which can lead to unexpected results.

Parameters

b [bool]

set_clip_path(path, transform=None)

Set the artist's clip path, which may be:

- a Patch (or subclass) instance
- a Path instance, in which case an optional Transform instance may be provided, which will be applied to the path before using it for clipping.
- None, to remove the clipping path

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to None.

ACCEPTS: [(Path, Transform) | Patch | None]

set_color(color)

Set the foreground color of the text

ACCEPTS: any matplotlib color

set_family(fontname)

Set the font family. May be either a single string, or a list of strings in decreasing priority. Each string may be either a real font name or a generic font class name. If the latter, the specific font names will be looked up in the matplotlibrc file.

ACCEPTS: [FONTNAME | 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace']

set_font_properties(fp)

alias for set_fontproperties

set_fontname(fontname)

alias for set_family

set_fontproperties(fp)

Set the font properties that control the text. fp must be a matplotlib.font_manager.FontProperties object.

ACCEPTS: a matplotlib.font_manager.FontProperties instance

set_fontsize(fontsize)

alias for set_size

set_fontstretch(stretch)

alias for set_stretch

set_fontstyle(fontstyle)

alias for set_style

set_fontvariant(variant)

alias for set_variant

`set_fontweight(weight)`
alias for `set_weight`

`set_ha(alignment)`
alias for `set_horizontalalignment`

`set_horizontalalignment(alignment)`
Set the horizontal alignment to one of
ACCEPTS: ['center' | 'right' | 'left']

`set_linespacing(spacing)`
Set the line spacing as a multiple of the font size. Default is 1.2.
ACCEPTS: float (multiple of font size)

`set_ma(alignment)`
alias for `set_multialignment`

`set_multialignment(alignment)`
Set the alignment for multiple lines layout. The layout of the bounding box of all the lines is determined by the `horizontalalignment` and `verticalalignment` properties, but the multiline text within that box can be
ACCEPTS: ['left' | 'right' | 'center']

`set_name(fontname)`
alias for `set_family`

`set_position(xy)`
Set the (x, y) position of the text
ACCEPTS: (x,y)

`set_rotation(s)`
Set the rotation of the text
ACCEPTS: [angle in degrees | 'vertical' | 'horizontal']

`set_rotation_mode(m)`
Set text rotation mode.

Parameters

`m` [None or "default" or "anchor"] If None or "default", the text will be first rotated, then aligned according to their horizontal and vertical alignments. If "anchor", then alignment occurs before rotation.

`set_size(fontsize)`
Set the font size. May be either a size string, relative to the default font size, or an absolute font size in points.
ACCEPTS: [size in points | 'xx-small' | 'x-small' | 'small' | 'medium' | 'large' | 'x-large' | 'xx-large']

`set_stretch(stretch)`
Set the font stretch (horizontal condensation or expansion).

ACCEPTS: [a numeric value in range 0-1000 | 'ultra-condensed' | 'extra-condensed' | 'condensed' | 'semi-condensed' | 'normal' | 'semi-expanded' | 'expanded' | 'extra-expanded' | 'ultra-expanded']

`set_style(fontstyle)`
Set the font style.

ACCEPTS: ['normal' | 'italic' | 'oblique']

`set_text(s)`
Set the text string `s`

It may contain newlines (`\n`) or math in LaTeX syntax.

ACCEPTS: string or anything printable with '%s' conversion.

`set_usetex(usetex)`

Parameters

`usetex` [bool or None] Whether to render using TeX, None means to use `rcParams["text.usetex"]`.

`set_va(align)`
alias for `set_verticalalignment`

`set_variant(variant)`
Set the font variant, either 'normal' or 'small-caps'.

ACCEPTS: ['normal' | 'small-caps']

`set_verticalalignment(align)`
Set the vertical alignment

ACCEPTS: ['center' | 'top' | 'bottom' | 'baseline']

`set_weight(weight)`
Set the font weight.

ACCEPTS: [a numeric value in range 0-1000 | 'ultralight' | 'light' | 'normal' | 'regular' | 'book' | 'medium' | 'roman' | 'semibold' | 'demibold' | 'demi' | 'bold' | 'heavy' | 'extra bold' | 'black']

`set_wrap(wrap)`
Sets the wrapping state for the text.

Parameters

`wrap` [bool]

`set_x(x)`
Set the x position of the text

ACCEPTS: float

`set_y(y)`
Set the y position of the text

ACCEPTS: float

update(kwargs)

Update properties from a dictionary.

update_bbox_position_size(renderer)

Update the location and the size of the bbox. This method should be used when the position and size of the bbox needs to be updated before actually drawing the bbox.

update_from(other)

Copy properties from other to self

zorder = 3

```
class matplotlib.text.TextWithDash(x=0, y=0, text="", color=None, verticalalign=
    'center', horizontalalignment='center', multi-
    alignment=None, fontproperties=None, rotation=None,
    linespacing=None, dashlength=0.0, dashdirection=0,
    dashrotation=None, dashpad=3, dashpush=0)
```

Bases: matplotlib.text.Text

This is basically a Text with a dash (drawn with a Line2D) before/after it. It is intended to be a drop-in replacement for Text, and should behave identically to it when dashlength = 0.0.

The dash always comes between the point specified by set_position() and the text. When a dash exists, the text alignment arguments (horizontalalignment, verticalalignment) are ignored.

dashlength is the length of the dash in canvas units. (default = 0.0).

dashdirection is one of 0 or 1, where 0 draws the dash after the text and 1 before. (default = 0).

dashrotation specifies the rotation of the dash, and should generally stay None. In this case get_dashrotation() returns get_rotation(). (i.e., the dash takes its rotation from the text's rotation). Because the text center is projected onto the dash, major deviations in the rotation cause what may be considered visually unappealing results. (default = None)

dashpad is a padding length to add (or subtract) space between the text and the dash, in canvas units. (default = 3)

dashpush "pushes" the dash and text away from the point specified by set_position() by the amount in canvas units. (default = 0)

Note: The alignment of the two objects is based on the bounding box of the Text, as obtained by get_window_extent(). This, in turn, appears to depend on the font metrics as given by the rendering backend. Hence the quality of the "centering" of the label text with respect to the dash varies depending on the backend used.

Note: I'm not sure that I got the `get_window_extent()` right, or whether that's sufficient for providing the object bounding box.

`draw(renderer)`

Draw the `TextWithDash` object to the given `renderer`.

`get_dashdirection()`

Get the direction dash. 1 is before the text and 0 is after.

`get_dashlength()`

Get the length of the dash.

`get_dashpad()`

Get the extra spacing between the dash and the text, in canvas units.

`get_dashpush()`

Get the extra spacing between the dash and the specified text position, in canvas units.

`get_dashrotation()`

Get the rotation of the dash in degrees.

`get_figure()`

return the figure instance the artist belongs to

`get_position()`

Return the position of the text as a tuple (x, y)

`get_prop_tup(renderer=None)`

Return a hashable tuple of properties.

Not intended to be human readable, but useful for backends who want to cache derived information about text (e.g., layouts) and need to know if the text has changed.

`get_unitless_position()`

Return the unitless position of the text as a tuple (x, y)

`get_window_extent(renderer=None)`

Return a `Bbox` object bounding the text, in display units.

In addition to being used internally, this is useful for specifying clickable regions in a png file on a web page.

`renderer` defaults to the `__renderer` attribute of the text object. This is not assigned until the first execution of `draw()`, so you must use this kwarg if you want to call `get_window_extent()` prior to the first `draw()`. For getting web page regions, it is simpler to call the method after saving the figure.

`set_dashdirection(dd)`

Set the direction of the dash following the text. 1 is before the text and 0 is after. The default is 0, which is what you'd want for the typical case of ticks below and on the left of the figure.

ACCEPTS: int (1 is before, 0 is after)

`set_dashlength(dl)`

Set the length of the dash.

ACCEPTS: float (canvas units)

`set_dashpad(dp)`

Set the "pad" of the `TextWithDash`, which is the extra spacing between the dash and the text, in canvas units.

ACCEPTS: float (canvas units)

`set_dashpush(dp)`

Set the "push" of the `TextWithDash`, which is the extra spacing between the beginning of the dash and the specified position.

ACCEPTS: float (canvas units)

`set_dashrotation(dr)`

Set the rotation of the dash, in degrees

ACCEPTS: float (degrees)

`set_figure(fig)`

Set the figure instance the artist belong to.

ACCEPTS: a `matplotlib.figure.Figure` instance

`set_position(xy)`

Set the (x, y) position of the `TextWithDash`.

ACCEPTS: (x, y)

`set_transform(t)`

Set the `matplotlib.transforms.Transform` instance used by this artist.

ACCEPTS: a `matplotlib.transforms.Transform` instance

`set_x(x)`

Set the x position of the `TextWithDash`.

ACCEPTS: float

`set_y(y)`

Set the y position of the `TextWithDash`.

ACCEPTS: float

`update_coords(renderer)`

Computes the actual x, y coordinates for text based on the input x, y and the dash-length. Since the rotation is with respect to the actual canvas's coordinates we need to map back and forth.

`matplotlib.text.get_rotation(rotation)`

Return the text angle as float. The returned angle is between 0 and 360 deg.

rotation may be 'horizontal', 'vertical', or a numeric value in degrees.

68.1 matplotlib.ticker

68.1.1 Tick locating and formatting

This module contains classes to support completely configurable tick locating and formatting. Although the locators know nothing about major or minor ticks, they are used by the Axis class to support major and minor tick locating and formatting. Generic tick locators and formatters are provided, as well as domain specific custom ones.

Default Formatter

The default formatter identifies when the x-data being plotted is a small range on top of a large offset. To reduce the chances that the ticklabels overlap, the ticks are labeled as deltas from a fixed offset. For example:

```
ax.plot(np.arange(2000, 2010), range(10))
```

will have tick of 0-9 with an offset of +2e3. If this is not desired turn off the use of the offset on the default formatter:

```
ax.get_xaxis().get_major_formatter().set_useOffset(False)
```

set the rcParam axes.formatter.useoffset=False to turn it off globally, or set a different formatter.

Tick locating

The Locator class is the base class for all tick locators. The locators handle autoscaling of the view limits based on the data limits, and the choosing of tick locations. A useful semi-automatic tick locator is MultipleLocator. It is initialized with a base, e.g., 10, and it picks axis limits and ticks that are multiples of that base.

The Locator subclasses defined here are

AutoLocator MaxNLocator with simple defaults. This is the default tick locator for most plotting.

MaxNLocator Finds up to a max number of intervals with ticks at nice locations.

LinearLocator Space ticks evenly from min to max.

LogLocator Space ticks logarithmically from min to max.

MultipleLocator Ticks and range are a multiple of base; either integer or float.

FixedLocator Tick locations are fixed.

IndexLocator Locator for index plots (e.g., where $x = \text{range}(\text{len}(y))$).

NullLocator No ticks.

SymmetricalLogLocator Locator for use with with the symlog norm; works like LogLocator for the part outside of the threshold and adds 0 if inside the limits.

LogitLocator Locator for logit scaling.

OldAutoLocator Choose a MultipleLocator and dynamically reassign it for intelligent ticking during navigation.

AutoMinorLocator Locator for minor ticks when the axis is linear and the major ticks are uniformly spaced. Subdivides the major tick interval into a specified number of minor intervals, defaulting to 4 or 5 depending on the major interval.

There are a number of locators specialized for date locations - see the dates module.

You can define your own locator by deriving from Locator. You must override the `__call__` method, which returns a sequence of locations, and you will probably want to override the `autoscale` method to set the view limits from the data limits.

If you want to override the default locator, use one of the above or a custom locator and pass it to the x or y axis instance. The relevant methods are:

```
ax.xaxis.set_major_locator(xmajor_locator)
ax.xaxis.set_minor_locator(xminor_locator)
ax.yaxis.set_major_locator(ymajor_locator)
ax.yaxis.set_minor_locator(yminor_locator)
```

The default minor locator is NullLocator, i.e., no minor ticks on by default.

Tick formatting

Tick formatting is controlled by classes derived from Formatter. The formatter operates on a single tick value and returns a string to the axis.

NullFormatter No labels on the ticks.

IndexFormatter Set the strings from a list of labels.

FixedFormatter Set the strings manually for the labels.

FuncFormatter User defined function sets the labels.

StrMethodFormatter Use string `format` method.

FormatStrFormatter Use an old-style `sprintf` format string.

`ScalarFormatter` Default formatter for scalars: autopick the format string.

`LogFormatter` Formatter for log axes.

`LogFormatterExponent` Format values for log axis using `exponent = log_base(value)`.

`LogFormatterMathtext` Format values for log axis using `exponent = log_base(value)` using `Math text`.

`LogFormatterSciNotation` Format values for log axis using scientific notation.

`LogitFormatter` Probability formatter.

`EngFormatter` Format labels in engineering notation

`PercentFormatter` Format labels as a percentage

You can derive your own formatter from the `Formatter` base class by simply overriding the `__call__` method. The formatter class has access to the axis view and data limits.

To control the major and minor tick label formats, use one of the following methods:

```
ax.xaxis.set_major_formatter(xmajor_formatter)
ax.xaxis.set_minor_formatter(xminor_formatter)
ax.yaxis.set_major_formatter(ymajor_formatter)
ax.yaxis.set_minor_formatter(yminor_formatter)
```

See [/gallery/ticks_and_spines/major_minor_demo](#) for an example of setting major and minor ticks. See the `matplotlib.dates` module for more information and examples of using date locators and formatters.

```
class matplotlib.ticker.TickHelper
```

Bases: `object`

`axis = None`

`create_dummy_axis(**kwargs)`

`set_axis(axis)`

`set_bounds(vmin, vmax)`

`set_data_interval(vmin, vmax)`

`set_view_interval(vmin, vmax)`

```
class matplotlib.ticker.Formatter
```

Bases: `matplotlib.ticker.TickHelper`

Create a string based on a tick value and location.

`fix_minus(s)`

Some classes may want to replace a hyphen for minus with the proper unicode symbol (U+2212) for typographical correctness. The default is to not replace it.

Note, if you use this method, e.g., in `format_data()` or call, you probably don't want to use it for `format_data_short()` since the toolbar uses this for interactive coord reporting and I doubt we can expect GUIs across platforms will handle the unicode correctly. So for now the classes that override `fix_minus()` should have an explicit `format_data_short()` method

`format_data(value)`

Returns the full string representation of the value with the position unspecified.

`format_data_short(value)`

Return a short string version of the tick value.

Defaults to the position-independent long value.

`get_offset()`

`locs = []`

`set_locs(locs)`

`class matplotlib.ticker.FixedFormatter(seq)`

Bases: `matplotlib.ticker.Formatter`

Return fixed strings for tick labels based only on position, not value.

Set the sequence of strings that will be used for labels.

`get_offset()`

`set_offset_string(ofs)`

`class matplotlib.ticker.NullFormatter`

Bases: `matplotlib.ticker.Formatter`

Always return the empty string.

`class matplotlib.ticker.FuncFormatter(func)`

Bases: `matplotlib.ticker.Formatter`

Use a user-defined function for formatting.

The function should take in two inputs (a tick value `x` and a position `pos`), and return a string containing the corresponding tick label.

`class matplotlib.ticker.FormatStrFormatter(fmt)`

Bases: `matplotlib.ticker.Formatter`

Use an old-style ('%' operator) format string to format the tick.

The format string should have a single variable format (%) in it. It will be applied to the value (not the position) of the tick.

```
class matplotlib.ticker.StrMethodFormatter(fmt)
```

Bases: matplotlib.ticker.Formatter

Use a new-style format string (as used by `str.format()`) to format the tick.

The field used for the value must be labeled x and the field used for the position must be labeled pos.

```
class matplotlib.ticker.ScalarFormatter(useOffset=None,      useMathText=None,      useLo-
                                       cale=None)
```

Bases: matplotlib.ticker.Formatter

Format tick values as a number.

Tick value is interpreted as a plain old number. If `useOffset==True` and the data range is much smaller than the data average, then an offset will be determined such that the tick labels are meaningful. Scientific notation is used for data $< 10^{-n}$ or data $\geq 10^m$, where `n` and `m` are the power limits set using `set_powerlimits((n,m))`. The defaults for these are controlled by the `axes.formatter.limits` rc parameter.

```
fix_minus(s)
```

Replace hyphens with a unicode minus.

```
format_data(value)
```

Return a formatted string representation of a number.

```
format_data_short(value)
```

Return a short formatted string representation of a number.

```
get_offset()
```

Return scientific notation, plus offset.

```
get_useLocale()
```

```
get_useMathText()
```

```
get_useOffset()
```

```
pprint_val(x)
```

```
set_locs(locs)
```

Set the locations of the ticks.

```
set_powerlimits(lims)
```

Sets size thresholds for scientific notation.

Parameters

`limits` [(min_exp, max_exp)] A tuple containing the powers of 10 that determine the switchover threshold. Numbers below $10^{\text{min_exp}}$ and above $10^{\text{max_exp}}$ will be displayed in scientific notation.

For example, `formatter.set__powerlimits((-3, 4))` sets the pre-2007 default in which scientific notation is used for numbers less than 10^{-3} or greater than 10^4 .

.. seealso:: Method :meth:`set__scientific`

`set__scientific(b)`

Turn scientific notation on or off.

See also:

Method `set__powerlimits()`

`set__useLocale(val)`

`set__useMathText(val)`

`set__useOffset(val)`

`useLocale`

`useMathText`

`useOffset`

`class matplotlib.ticker.LogFormatter(base=10.0, labelOnlyBase=False, minor_thresholds=None, linthresh=None)`

Bases: `matplotlib.ticker.Formatter`

Base class for formatting ticks on a log or symlog scale.

It may be instantiated directly, or subclassed.

Parameters

`base` [float, optional, default: 10.] Base of the logarithm used in all calculations.

`labelOnlyBase` [bool, optional, default: False] If True, label ticks only at integer powers of base. This is normally True for major ticks and False for minor ticks.

`minor_thresholds` [(subset, all), optional, default: (1, 0.4)] If `labelOnlyBase` is False, these two numbers control the labeling of ticks that are not at integer powers of base; normally these are the minor ticks. The controlling parameter is the log of the axis data range. In the typical case where base is 10 it is the number of decades spanned by the axis, so we can call

it 'numdec'. If numdec \leq all, all minor ticks will be labeled. If all $<$ numdec \leq subset, then only a subset of minor ticks will be labeled, so as to avoid crowding. If numdec $>$ subset then no minor ticks will be labeled.

linthresh [None or float, optional, default: None] If a symmetric log scale is in use, its linthresh parameter must be supplied here.

Notes

The `set_locs` method must be called to enable the subsetting logic controlled by the `minor_thresholds` parameter.

In some cases such as the colorbar, there is no distinction between major and minor ticks; the tick locations might be set manually, or by a locator that puts ticks at integer powers of base and at intermediate locations. For this situation, disable the `minor_thresholds` logic by using `minor_thresholds=(np.inf, np.inf)`, so that all ticks will be labeled.

To disable labeling of minor ticks when 'labelOnlyBase' is False, use `minor_thresholds=(0, 0)`. This is the default for the "classic" style.

Examples

To label a subset of minor ticks when the view limits span up to 2 decades, and all of the ticks when zoomed in to 0.5 decades or less, use `minor_thresholds=(2, 0.5)`.

To label all minor ticks when the view limits span up to 1.5 decades, use `minor_thresholds=(1.5, 1.5)`.

`base(base)`

Change the base for labeling.

Warning: Should always match the base used for LogLocator

`format_data(value)`

Returns the full string representation of the value with the position unspecified.

`format_data_short(value)`

Return a short formatted string representation of a number.

`label_minor(labelOnlyBase)`

Switch minor tick labeling on or off.

Parameters

labelOnlyBase [bool] If True, label ticks only at integer powers of base.

`pprint_val(x, d)`

`set_locs(locs=None)`

Use axis view limits to control which ticks are labeled.

The `locs` parameter is ignored in the present algorithm.

`class matplotlib.ticker.LogFormatterExponent(base=10.0, labelOnlyBase=False, minor_thresholds=None, linthresh=None)`

Bases: `matplotlib.ticker.LogFormatter`

Format values for log axis using `exponent = log_base(value)`.

`class matplotlib.ticker.LogFormatterMathtext(base=10.0, labelOnlyBase=False, minor_thresholds=None, linthresh=None)`

Bases: `matplotlib.ticker.LogFormatter`

Format values for log axis using `exponent = log_base(value)`.

`class matplotlib.ticker.IndexFormatter(labels)`

Bases: `matplotlib.ticker.Formatter`

Format the position `x` to the nearest `i`-th label where `i=int(x+0.5)`

`class matplotlib.ticker.LogFormatterSciNotation(base=10.0, labelOnlyBase=False, minor_thresholds=None, linthresh=None)`

Bases: `matplotlib.ticker.LogFormatterMathtext`

Format values following scientific notation in a logarithmic axis.

`class matplotlib.ticker.LogitFormatter`

Bases: `matplotlib.ticker.Formatter`

Probability formatter (using Math text).

`format_data_short(value)`

return a short formatted string representation of a number

`class matplotlib.ticker.EngFormatter(unit="", places=None, sep=' ')`

Bases: `matplotlib.ticker.Formatter`

Formats axis values using engineering prefixes to represent powers of 1000, plus a specified unit, e.g., 10 MHz instead of 1e7.

Parameters

`unit` [str (default: "")] Unit symbol to use, suitable for use with single-letter representations of powers of 1000. For example, 'Hz' or 'm'.

`places` [int (default: None)] Precision with which to display the number, specified in digits after the decimal point (there will be between one and three digits before the decimal point). If it is None, the formatting falls back to the floating point format '%g', which displays up to 6 significant digits, i.e. the equivalent value for `places` varies between 0 and 5 (inclusive).

`sep` [str (default: " ")] Separator used between the value and the prefix/unit. For example, one get '3.14 mV' if `sep` is " " (default) and '3.14mV' if `sep` is "". Besides the default behavior, some other useful options may be:

- `sep=""` to append directly the prefix/unit to the value;

- `sep="\N{THIN SPACE}"` (U+2009);
- `sep="\N{NARROW NO-BREAK SPACE}"` (U+202F);
- `sep="\N{NO-BREAK SPACE}"` (U+00A0).

`ENG_PREFIXES = {-24: 'y', -21: 'z', -18: 'a', -15: 'f', -12: 'p', -9: 'n', -6: ' μ ', -3: 'm', 0: '', 3: 'k', 6: 'M',`

`format_eng(num)`

Formats a number in engineering notation, appending a letter representing the power of 1000 of the original number. Some examples:

```
>>> format_eng(0)      # for self.places = 0
'0'
```

```
>>> format_eng(1000000) # for self.places = 1
'1.0 M'
```

```
>>> format_eng("-1e-6") # for self.places = 2
'u'-1.00  $\mu$ '
```

`num` may be a numeric value or a string that can be converted to a numeric value with `float(num)`.

```
class matplotlib.ticker.PercentFormatter(xmax=100,      decimals=None,      symbol='%',
                                         is_latex=False)
```

Bases: `matplotlib.ticker.Formatter`

Format numbers as a percentage.

Parameters

`xmax` [float] Determines how the number is converted into a percentage. `xmax` is the data value that corresponds to 100%. Percentages are computed as `x / xmax * 100`. So if the data is already scaled to be percentages, `xmax` will be 100. Another common situation is where `xmax` is 1.0.

`decimals` [None or int] The number of decimal places to place after the point. If None (the default), the number will be computed automatically.

`symbol` [string or None] A string that will be appended to the label. It may be None or empty to indicate that no symbol should be used. LaTeX special characters are escaped in symbol whenever latex mode is enabled, unless `is_latex` is True.

`is_latex` [bool] If False, reserved LaTeX characters in symbol will be escaped.

`convert_to_pct(x)`

`format_pct(x, display_range)`

Formats the number as a percentage number with the correct number of decimals and adds the percent symbol, if any.

If `self.decimals` is `None`, the number of digits after the decimal point is set based on the `display_range` of the axis as follows:

<code>display_range</code>	<code>decimals</code>	sample
<code>>50</code>	0	<code>x = 34.5 => 35%</code>
<code>>5</code>	1	<code>x = 34.5 => 34.5%</code>
<code>>0.5</code>	2	<code>x = 34.5 => 34.50%</code>
<code>...</code>	<code>...</code>	<code>...</code>

This method will not be very good for tiny axis ranges or extremely large ones. It assumes that the values on the chart are percentages displayed on a reasonable scale.

`symbol`

The configured percent symbol as a string.

If LaTeX is enabled via `rcParams["text.usetex"]`, the special characters `{'##', '$', '%', '&', '~', '_', '^', '\\', '{', '}'}` are automatically escaped in the string.

class `matplotlib.ticker.Locator`

Bases: `matplotlib.ticker.TickHelper`

Determine the tick locations;

Note, you should not use the same locator between different `Axis` because the locator stores references to the `Axis` data and view limits

`MAXTICKS = 1000`

`autoscale()`

autoscale the view limits

`pan(numsteps)`

Pan numticks (can be positive or negative)

`raise_if_exceeds(locs)`

raise a `RuntimeError` if `Locator` attempts to create more than `MAXTICKS` locs

`refresh()`

refresh internal information based on current lim

`set_params(**kwargs)`

Do nothing, and raise a warning. Any locator class not supporting the `set_params()` function will call this.

`tick_values(vmin, vmax)`

Return the values of the located ticks given `vmin` and `vmax`.

Note: To get tick locations with the `vmin` and `vmax` values defined automatically for the associated axis simply call the `Locator` instance:

```
>>> print((type(loc)))
<type 'Locator'>
>>> print((loc()))
[1, 2, 3, 4]
```

`view_limits(vmin, vmax)`

select a scale for the range from `vmin` to `vmax`

Normally this method is overridden by subclasses to change locator behaviour.

`zoom(direction)`

Zoom in/out on axis; if `direction` is `>0` zoom in, else zoom out

class `matplotlib.ticker.IndexLocator(base, offset)`

Bases: `matplotlib.ticker.Locator`

Place a tick on every multiple of some base number of points plotted, e.g., on every 5th point. It is assumed that you are doing index plotting; i.e., the axis is `0, len(data)`. This is mainly useful for x ticks.

place ticks on the `i`-th data points where `(i-offset)%base==0`

`set_params(base=None, offset=None)`

Set parameters within this locator

`tick_values(vmin, vmax)`

Return the values of the located ticks given `vmin` and `vmax`.

Note: To get tick locations with the `vmin` and `vmax` values defined automatically for the associated axis simply call the `Locator` instance:

```
>>> print((type(loc)))
<type 'Locator'>
>>> print((loc()))
[1, 2, 3, 4]
```

class `matplotlib.ticker.FixedLocator(locs, nbins=None)`

Bases: `matplotlib.ticker.Locator`

Tick locations are fixed. If `nbins` is not `None`, the array of possible positions will be subsampled to keep the number of ticks `<= nbins + 1`. The subsampling will be done so as to include the smallest absolute value; for example, if zero is included in the array of possibilities, then it is guaranteed to be one of the chosen ticks.

`set_params(nbins=None)`

Set parameters within this locator.

`tick_values(vmin, vmax)`

” Return the locations of the ticks.

Note: Because the values are fixed, vmin and vmax are not used in this method.

```
class matplotlib.ticker.NullLocator
    Bases: matplotlib.ticker.Locator

    No ticks

    tick_values(vmin, vmax)
        " Return the locations of the ticks.
```

Note: Because the values are Null, vmin and vmax are not used in this method.

```
class matplotlib.ticker.LinearLocator(numticks=None, presets=None)
    Bases: matplotlib.ticker.Locator

    Determine the tick locations

    The first time this function is called it will try to set the number of ticks to make a nice tick
    partitioning. Thereafter the number of ticks will be fixed so that interactive navigation will
    be nice

    Use presets to set locs based on lom. A dict mapping vmin, vmax->locs

    set_params(numticks=None, presets=None)
        Set parameters within this locator.

    tick_values(vmin, vmax)
        Return the values of the located ticks given vmin and vmax.
```

Note: To get tick locations with the vmin and vmax values defined automatically for the associated axis simply call the Locator instance:

```
>>> print((type(loc)))
<type 'Locator'>
>>> print((loc()))
[1, 2, 3, 4]
```

```
view_limits(vmin, vmax)
    Try to choose the view limits intelligently
```

```
class matplotlib.ticker.LogLocator(base=10.0, subs=(1.0, ), numdecs=4, numticks=None)
    Bases: matplotlib.ticker.Locator

    Determine the tick locations for log axes

    Place ticks on the locations : subs[j] * base**i

    Parameters
```

`subs` [None, string, or sequence of float, optional, default (1.0,)] Gives the multiples of integer powers of the base at which to place ticks. The default places ticks only at integer powers of the base. The permitted string values are 'auto' and 'all', both of which use an algorithm based on the axis view limits to determine whether and how to put ticks between integer powers of the base. With 'auto', ticks are placed only between integer powers; with 'all', the integer powers are included. A value of None is equivalent to 'auto'.

`base(base)`

set the base of the log scaling (major tick every base^i , i integer)

`nonsingular(vmin, vmax)`

`set_params(base=None, subs=None, numdecs=None, numticks=None)`

Set parameters within this locator.

`subs(subs)`

set the minor ticks for the log scaling every $\text{base}^i \cdot \text{subs}[j]$

`tick_values(vmin, vmax)`

Return the values of the located ticks given `vmin` and `vmax`.

Note: To get tick locations with the `vmin` and `vmax` values defined automatically for the associated axis simply call the Locator instance:

```
>>> print((type(loc)))
<type 'Locator'>
>>> print((loc()))
[1, 2, 3, 4]
```

`view_limits(vmin, vmax)`

Try to choose the view limits intelligently

`class matplotlib.ticker.AutoLocator`

Bases: `matplotlib.ticker.MaxNLocator`

Dynamically find major tick positions. This is actually a subclass of `MaxNLocator`, with parameters `nbins = 'auto'` and `steps = [1, 2, 2.5, 5, 10]`.

To know the values of the non-public parameters, please have a look to the defaults of `MaxNLocator`.

`class matplotlib.ticker.MultipleLocator(base=1.0)`

Bases: `matplotlib.ticker.Locator`

Set a tick on every integer that is multiple of `base` in the view interval

`set_params(base)`

Set parameters within this locator.

`tick_values(vmin, vmax)`

Return the values of the located ticks given `vmin` and `vmax`.

Note: To get tick locations with the `vmin` and `vmax` values defined automatically for the associated axis simply call the Locator instance:

```
>>> print((type(loc)))
<type 'Locator'>
>>> print((loc()))
[1, 2, 3, 4]
```

`view_limits(dmin, dmax)`

Set the view limits to the nearest multiples of base that contain the data

`class matplotlib.ticker.MaxNLocator(*args, **kwargs)`

Bases: `matplotlib.ticker.Locator`

Select no more than `N` intervals at nice locations.

Keyword args:

`nbins` Maximum number of intervals; one less than max number of ticks. If the string 'auto', the number of bins will be automatically determined based on the length of the axis.

`steps` Sequence of nice numbers starting with 1 and ending with 10; e.g., [1, 2, 4, 5, 10], where the values are acceptable tick multiples. i.e. for the example, 20, 40, 60 would be an acceptable set of ticks, as would 0.4, 0.6, 0.8, because they are multiples of 2. However, 30, 60, 90 would not be allowed because 3 does not appear in the list of steps.

`integer` If True, ticks will take only integer values, provided at least `min_n_ticks` integers are found within the view limits.

`symmetric` If True, autoscaling will result in a range symmetric about zero.

`prune` ['lower' | 'upper' | 'both' | None] Remove edge ticks – useful for stacked or ganged plots where the upper tick of one axes overlaps with the lower tick of the axes above it, primarily when `rcParams["axes.autolimit_mode"]` is 'round_numbers'. If `prune=='lower'`, the smallest tick will be removed. If `prune=='upper'`, the largest tick will be removed. If `prune=='both'`, the largest and smallest ticks will be removed. If `prune==None`, no ticks will be removed.

`min_n_ticks` Relax `nbins` and `integer` constraints if necessary to obtain this minimum number of ticks.

`default_params` = {'integer': False, 'min_n_ticks': 2, 'nbins': 10, 'prune': None, 'steps': None, 'symmetric': False}

`set_params(**kwargs)`

Set parameters within this locator.

`tick_values(vmin, vmax)`

Return the values of the located ticks given `vmin` and `vmax`.

Note: To get tick locations with the vmin and vmax values defined automatically for the associated axis simply call the Locator instance:

```
>>> print((type(loc)))
<type 'Locator'>
>>> print((loc()))
[1, 2, 3, 4]
```

`view_limits(dmin, dmax)`
 select a scale for the range from vmin to vmax

Normally this method is overridden by subclasses to change locator behaviour.

`class matplotlib.ticker.AutoMinorLocator(n=None)`
 Bases: `matplotlib.ticker.Locator`

Dynamically find minor tick positions based on the positions of major ticks. The scale must be linear with major ticks evenly spaced.

`n` is the number of subdivisions of the interval between major ticks; e.g., `n=2` will place a single minor tick midway between major ticks.

If `n` is omitted or `None`, it will be set to 5 or 4.

`tick_values(vmin, vmax)`
 Return the values of the located ticks given `vmin` and `vmax`.

Note: To get tick locations with the vmin and vmax values defined automatically for the associated axis simply call the Locator instance:

```
>>> print((type(loc)))
<type 'Locator'>
>>> print((loc()))
[1, 2, 3, 4]
```

`class matplotlib.ticker.SymmetricalLogLocator(transform=None, subs=None, linthresh=None, base=None)`

Bases: `matplotlib.ticker.Locator`

Determine the tick locations for symmetric log axes

place ticks on the location= `base**i*subs[j]`

`set_params(subs=None, numticks=None)`
 Set parameters within this locator.

`tick_values(vmin, vmax)`
 Return the values of the located ticks given `vmin` and `vmax`.

Note: To get tick locations with the `vmin` and `vmax` values defined automatically for the associated axis simply call the Locator instance:

```
>>> print((type(loc)))  
<type 'Locator'>  
>>> print((loc()))  
[1, 2, 3, 4]
```

`view_limits(vmin, vmax)`

Try to choose the view limits intelligently

`class matplotlib.ticker.LogitLocator(minor=False)`

Bases: `matplotlib.ticker.Locator`

Determine the tick locations for logit axes

place ticks on the logit locations

`nonsingular(vmin, vmax)`

`set_params(minor=None)`

Set parameters within this locator.

`tick_values(vmin, vmax)`

Return the values of the located ticks given `vmin` and `vmax`.

Note: To get tick locations with the `vmin` and `vmax` values defined automatically for the associated axis simply call the Locator instance:

```
>>> print((type(loc)))  
<type 'Locator'>  
>>> print((loc()))  
[1, 2, 3, 4]
```

TIGHT_LAYOUT

69.1 matplotlib.tight_layout

This module provides routines to adjust subplot params so that subplots are nicely fit in the figure. In doing so, only axis labels, tick labels, axes titles and offsetboxes that are anchored to axes are currently considered.

Internally, it assumes that the margins (`left_margin`, etc.) which are differences between `ax.get_tightbbox` and `ax.bbox` are independent of axes position. This may fail if `Axes.adjustable` is `datalim`. Also, This will fail for some cases (for example, left or right margin is affected by `xlabel`).

```
matplotlib.tight_layout.auto_adjust_subplotpars(fig,          renderer,          nrows_ncols,
                                                num1num2_list,      subplot_list,
                                                ax_bbox_list=None,    pad=1.08,
                                                h_pad=None,          w_pad=None,
                                                rect=None)
```

Return a dict of subplot parameters to adjust spacing between subplots.

Note that this function ignores geometry information of subplot itself, but uses what is given by the `nrows_ncols` and `num1num2_list` parameters. Also, the results could be incorrect if some subplots have `adjustable=datalim`.

Parameters

`nrows_ncols` [Tuple[int, int]] Number of rows and number of columns of the grid.

`num1num2_list` [List[int]] List of numbers specifying the area occupied by the subplot

`subplot_list` [list of subplots] List of subplots that will be used to calculate optimal subplot_params.

`pad` [float] Padding between the figure edge and the edges of subplots, as a fraction of the font size.

`h_pad, w_pad` [float] Padding (height/width) between edges of adjacent subplots, as a fraction of the font size. Defaults to `pad`.

rect [Tuple[float, float, float, float]] [left, bottom, right, top] in normalized (0, 1) figure coordinates.

matplotlib.tight_layout.get_renderer(fig)

matplotlib.tight_layout.get_subplotspec_list(axes_list, grid_spec=None)

Return a list of subplotspec from the given list of axes.

For an instance of axes that does not support subplotspec, None is inserted in the list.

If grid_spec is given, None is inserted for those not from the given grid_spec.

matplotlib.tight_layout.get_tight_layout_figure(fig, axes_list, subplotspec_list, renderer,
pad=1.08, h_pad=None, w_pad=None,
rect=None)

Return subplot parameters for tight-laid-out-figure with specified padding.

Parameters

fig [Figure]

axes_list [list of Axes]

subplotspec_list [list of SubplotSpec] The subplotspecs of each axes.

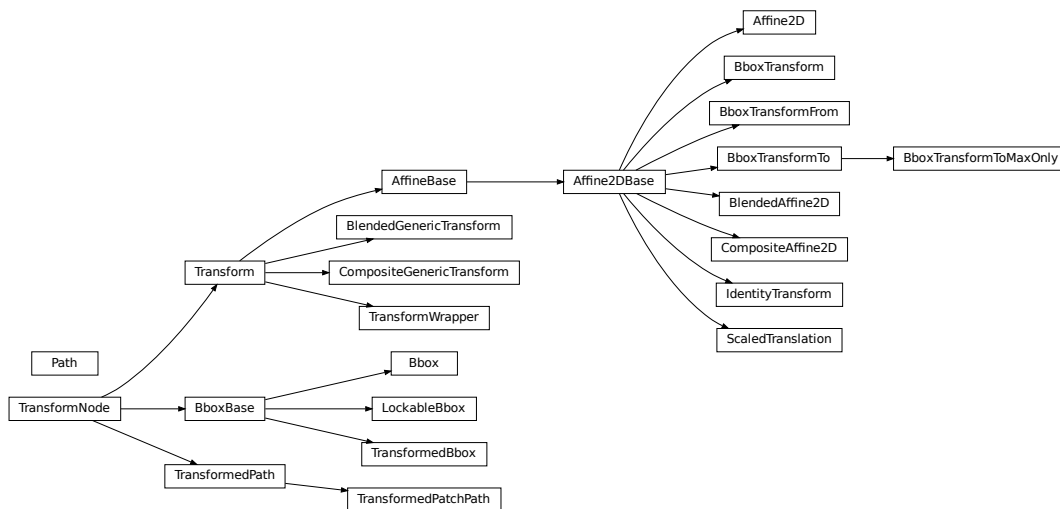
renderer [renderer]

pad [float] Padding between the figure edge and the edges of subplots, as a fraction of the font size.

h_pad, w_pad [float] Padding (height/width) between edges of adjacent subplots. Defaults to pad_inches.

rect [Tuple[float, float, float, float], optional] (left, bottom, right, top) rectangle in normalized figure coordinates that the whole subplots area (including labels) will fit into. Defaults to using the entire figure.

WORKING WITH TRANSFORMATIONS

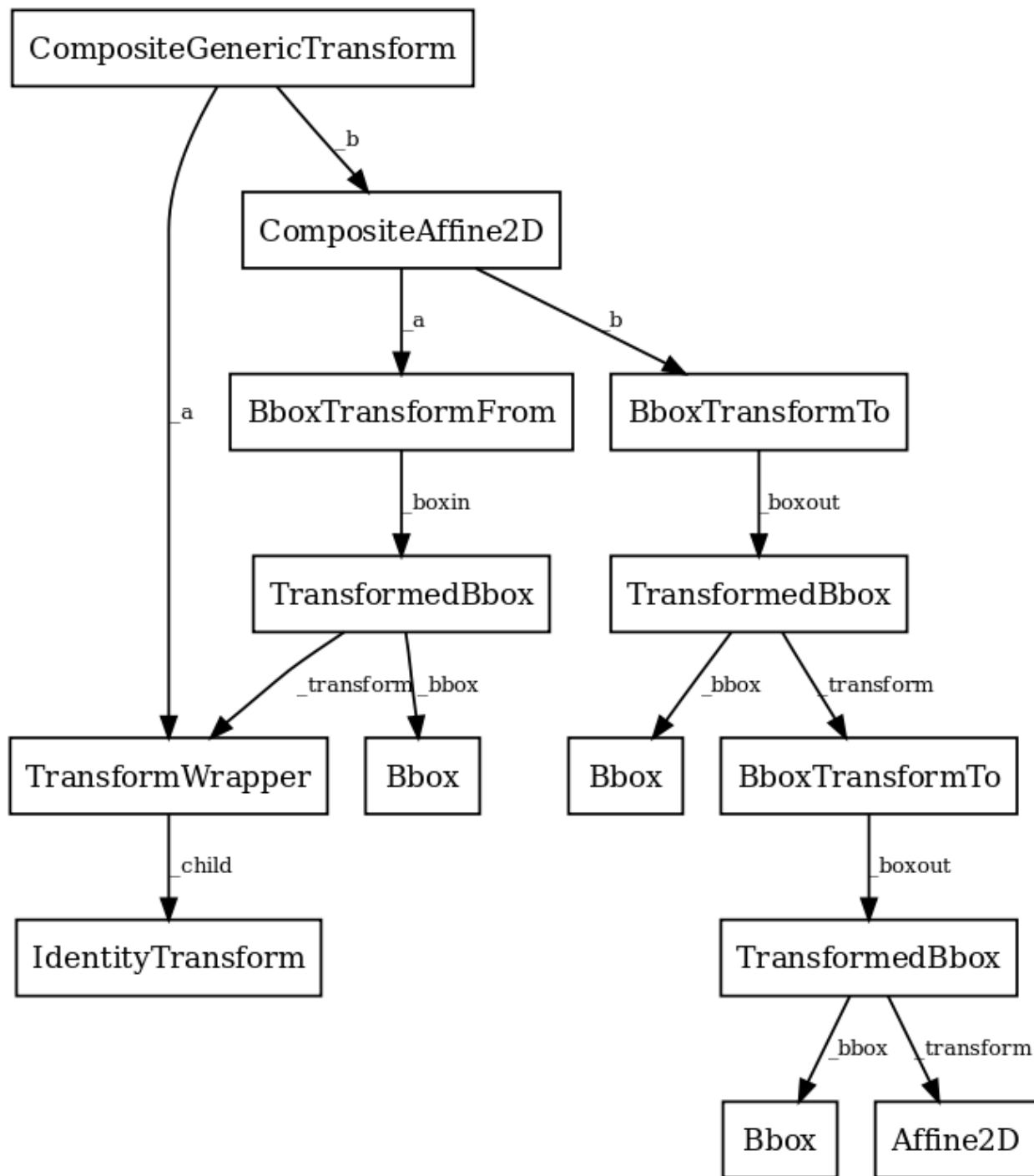


70.1 matplotlib.transforms

matplotlib includes a framework for arbitrary geometric transformations that is used to determine the final position of all elements drawn on the canvas.

Transforms are composed into trees of `TransformNode` objects whose actual value depends on their children. When the contents of children change, their parents are automatically invalidated. The next time an invalidated transform is accessed, it is recomputed to reflect those changes. This invalidation/caching approach prevents unnecessary recomputations of transforms, and contributes to better interactive performance.

For example, here is a graph of the transform tree used to plot data to the graph:



The framework can be used for both affine and non-affine transformations. However, for speed, we want use the backend renderers to perform affine transformations whenever possible. Therefore, it is possible to perform just the affine or non-affine part of a transformation on a set of data. The affine is always assumed to occur after the non-affine. For any transform:

$$\text{full transform} == \text{non-affine part} + \text{affine part}$$

The backends are not expected to handle non-affine transformations themselves.

class matplotlib.transforms.Affine2D(matrix=None, **kwargs)

Bases: matplotlib.transforms.Affine2DBase

A mutable 2D affine transformation.

Initialize an Affine transform from a 3x3 numpy float array:

```
a c e
b d f
0 0 1
```

If matrix is None, initialize with the identity transform.

clear()

Reset the underlying matrix to the identity transform.

static from_values(a, b, c, d, e, f)

(staticmethod) Create a new Affine2D instance from the given values:

```
a c e
b d f
0 0 1
```

.

get_matrix()

Get the underlying transformation matrix as a 3x3 numpy array:

```
a c e
b d f
0 0 1
```

.

static identity()

(staticmethod) Return a new Affine2D object that is the identity transform.

Unless this transform will be mutated later on, consider using the faster IdentityTransform class instead.

is_separable

rotate(theta)

Add a rotation (in radians) to this transform in place.

Returns self, so this method can easily be chained with more calls to rotate(), rotate_deg(), translate() and scale().

rotate_around(x, y, theta)

Add a rotation (in radians) around the point (x, y) in place.

Returns self, so this method can easily be chained with more calls to rotate(), rotate_deg(), translate() and scale().

`rotate_deg(degrees)`

Add a rotation (in degrees) to this transform in place.

Returns self, so this method can easily be chained with more calls to `rotate()`, `rotate_deg()`, `translate()` and `scale()`.

`rotate_deg_around(x, y, degrees)`

Add a rotation (in degrees) around the point (x, y) in place.

Returns self, so this method can easily be chained with more calls to `rotate()`, `rotate_deg()`, `translate()` and `scale()`.

`scale(sx, sy=None)`

Adds a scale in place.

If sy is None, the same scale is applied in both the x- and y-directions.

Returns self, so this method can easily be chained with more calls to `rotate()`, `rotate_deg()`, `translate()` and `scale()`.

`set(other)`

Set this transformation from the frozen copy of another Affine2DBase object.

`set_matrix(mtx)`

Set the underlying transformation matrix from a 3x3 numpy array:

a	c	e
b	d	f
0	0	1

.

`skew(xShear, yShear)`

Adds a skew in place.

xShear and yShear are the shear angles along the x- and y-axes, respectively, in radians.

Returns self, so this method can easily be chained with more calls to `rotate()`, `rotate_deg()`, `translate()` and `scale()`.

`skew_deg(xShear, yShear)`

Adds a skew in place.

xShear and yShear are the shear angles along the x- and y-axes, respectively, in degrees.

Returns self, so this method can easily be chained with more calls to `rotate()`, `rotate_deg()`, `translate()` and `scale()`.

`translate(tx, ty)`

Adds a translation in place.

Returns self, so this method can easily be chained with more calls to `rotate()`, `rotate_deg()`, `translate()` and `scale()`.

`class matplotlib.transforms.Affine2DBase(*args, **kwargs)`

Bases: `matplotlib.transforms.AffineBase`

The base class of all 2D affine transformations.

2D affine transformations are performed using a 3x3 numpy array:

```
a c e
b d f
0 0 1
```

This class provides the read-only interface. For a mutable 2D affine transformation, use `Affine2D`.

Subclasses of this class will generally only need to override a constructor and `get_matrix()` that generates a custom 3x3 matrix.

`frozen()`

Returns a frozen copy of this transform node. The frozen copy will not update when its children change. Useful for storing a previously known state of a transform where `copy.deepcopy()` might normally be used.

`has_inverse = True`

`input_dims = 2`

`inverted()`

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

`x === self.inverted().transform(self.transform(x))`

`is_separable`

`static matrix_from_values(a, b, c, d, e, f)`

(staticmethod) Create a new transformation matrix as a 3x3 numpy array of the form:

```
a c e
b d f
0 0 1
```

`output_dims = 2`

`to_values()`

Return the values of the matrix as a sequence (a,b,c,d,e,f)

`transform_affine(points)`

Performs only the affine part of this transformation on the given array of values.

`transform(values)` is always equivalent to `transform_affine(transform_non_affine(values))`.

In non-affine transformations, this is generally a no-op. In affine transformations, this is equivalent to `transform(values)`.

Accepts a numpy array of shape $(N \times \text{input_dims})$ and returns a numpy array of shape $(N \times \text{output_dims})$.

Alternatively, accepts a numpy array of length `input_dims` and returns a numpy array of length `output_dims`.

`transform_point(point)`

A convenience function that returns the transformed copy of a single point.

The point is given as a sequence of length `input_dims`. The transformed point is returned as a sequence of length `output_dims`.

`class matplotlib.transforms.AffineBase(*args, **kwargs)`

Bases: `matplotlib.transforms.Transform`

The base class of all affine transformations of any number of dimensions.

`get_affine()`

Get the affine part of this transform.

`is_affine = True`

`transform(values)`

Performs the transformation on the given array of values.

Accepts a numpy array of shape $(N \times \text{input_dims})$ and returns a numpy array of shape $(N \times \text{output_dims})$.

Alternatively, accepts a numpy array of length `input_dims` and returns a numpy array of length `output_dims`.

`transform_affine(values)`

Performs only the affine part of this transformation on the given array of values.

`transform(values)` is always equivalent to `transform_affine(transform_non_affine(values))`.

In non-affine transformations, this is generally a no-op. In affine transformations, this is equivalent to `transform(values)`.

Accepts a numpy array of shape $(N \times \text{input_dims})$ and returns a numpy array of shape $(N \times \text{output_dims})$.

Alternatively, accepts a numpy array of length `input_dims` and returns a numpy array of length `output_dims`.

`transform_non_affine(points)`

Performs only the non-affine part of the transformation.

`transform(values)` is always equivalent to `transform_affine(transform_non_affine(values))`.

In non-affine transformations, this is generally equivalent to `transform(values)`. In affine transformations, this is always a no-op.

Accepts a numpy array of shape (N x input_dims) and returns a numpy array of shape (N x output_dims).

Alternatively, accepts a numpy array of length input_dims and returns a numpy array of length output_dims.

`transform_path(path)`

Returns a transformed path.

path: a Path instance.

In some cases, this transform may insert curves into the path that began as line segments.

`transform_path_affine(path)`

Returns a path, transformed only by the affine part of this transform.

path: a Path instance.

`transform_path(path)` is equivalent to `transform_path_affine(transform_path_non_affine(values))`.

`transform_path_non_affine(path)`

Returns a path, transformed only by the non-affine part of this transform.

path: a Path instance.

`transform_path(path)` is equivalent to `transform_path_affine(transform_path_non_affine(values))`.

`class matplotlib.transforms.Bbox(points, **kwargs)`

Bases: `matplotlib.transforms.BboxBase`

A mutable bounding box.

Parameters

points [ndarray] A 2x2 numpy array of the form `[[x0, y0], [x1, y1]]`.

Notes

If you need to create a Bbox object from another form of data, consider the static methods `unit()`, `from_bounds()` and `from_extents()`.

`bounds`

Returns (x0, y0, width, height).

`static from_bounds(x0, y0, width, height)`

(staticmethod) Create a new Bbox from x0, y0, width and height.

width and height may be negative.

`static from_extents(*args)`

(staticmethod) Create a new Bbox from left, bottom, right and top.

The y-axis increases upwards.

`get_points()`

Get the points of the bounding box directly as a numpy array of the form: `[[x0, y0], [x1, y1]]`.

`ignore(value)`

Set whether the existing bounds of the box should be ignored by subsequent calls to `update_from_data_xy()`.

value [bool]

- When True, subsequent calls to `update_from_data_xy()` will ignore the existing bounds of the Bbox.
- When False, subsequent calls to `update_from_data_xy()` will include the existing bounds of the Bbox.

`intervalx`

`intervalx` is the pair of x coordinates that define the bounding box. It is not guaranteed to be sorted from left to right.

`intervaly`

`intervaly` is the pair of y coordinates that define the bounding box. It is not guaranteed to be sorted from bottom to top.

`minpos`

`minposx`

`minposy`

`mutated()`

Return whether the bbox has changed since init.

`mutatedx()`

Return whether the x-limits have changed since init.

`mutatedy()`

Return whether the y-limits have changed since init.

`static null()`

(staticmethod) Create a new null Bbox from (inf, inf) to (-inf, -inf).

`p0`

`p0` is the first pair of (x, y) coordinates that define the bounding box. It is not guaranteed to be the bottom-left corner. For that, use `min`.

`p1`

`p1` is the second pair of (x, y) coordinates that define the bounding box. It is not guaranteed to be the top-right corner. For that, use `max`.

`set(other)`

Set this bounding box from the "frozen" bounds of another Bbox.

`set_points(points)`

Set the points of the bounding box directly from a numpy array of the form: `[[x0, y0], [x1, y1]]`. No error checking is performed, as this method is mainly for internal use.

`static unit()`

(staticmethod) Create a new unit Bbox from (0, 0) to (1, 1).

`update_from_data_xy(xy, ignore=None, updatex=True, updatey=True)`

Update the bounds of the Bbox based on the passed in data. After updating, the bounds will have positive width and height; x0 and y0 will be the minimal values.

Parameters

`xy` [ndarray] A numpy array of 2D points.

`ignore` [bool, optional]

- When True, ignore the existing bounds of the Bbox.
- When False, include the existing bounds of the Bbox.
- When None, use the last value passed to `ignore()`.

`updatex, updatey` [bool, optional] When True, update the x/y values.

`update_from_path(path, ignore=None, updatex=True, updatey=True)`

Update the bounds of the Bbox based on the passed in data. After updating, the bounds will have positive width and height; x0 and y0 will be the minimal values.

Parameters

`path` [Path]

`ignore` [bool, optional]

- when True, ignore the existing bounds of the Bbox.
- when False, include the existing bounds of the Bbox.
- when None, use the last value passed to `ignore()`.

`updatex, updatey` [bool, optional] When True, update the x/y values.

`x0`

x0 is the first of the pair of x coordinates that define the bounding box. x0 is not guaranteed to be less than x1. If you require that, use `xmin`.

`x1`

x1 is the second of the pair of x coordinates that define the bounding box. x1 is not guaranteed to be greater than x0. If you require that, use `xmax`.

`y0`

y0 is the first of the pair of y coordinates that define the bounding box. y0 is not guaranteed to be less than y1. If you require that, use `ymin`.

`y1`

y1 is the second of the pair of y coordinates that define the bounding box. y1 is not guaranteed to be greater than y0. If you require that, use `ymax`.

`class matplotlib.transforms.BboxBase(shorthand_name=None)`

Bases: `matplotlib.transforms.TransformNode`

This is the base class of all bounding boxes, and provides read-only access to its data. A mutable bounding box is provided by the Bbox class.

The canonical representation is as two points, with no restrictions on their ordering. Convenience properties are provided to get the left, bottom, right and top edges and width and height, but these are not stored explicitly.

Creates a new TransformNode.

Parameters

shorthand_name [str] A string representing the "name" of the transform.
The name carries no significance other than to improve the readability of str(transform) when DEBUG=True.

anchored(c, container=None)

Return a copy of the Bbox, shifted to position c within a container.

Parameters

c : May be either:

- A sequence (cx, cy) where cx and cy range from 0 to 1, where 0 is left or bottom and 1 is right or top
- a string: - 'C' for centered - 'S' for bottom-center - 'SE' for bottom-left - 'E' for left - etc.

container [Bbox, optional] The box within which the Bbox is positioned; it defaults to the initial Bbox.

bounds

Returns (x0, y0, width, height).

coefs = {'C': (0.5, 0.5), 'E': (1.0, 0.5), 'N': (0.5, 1.0), 'NE': (1.0, 1.0), 'NW': (0, 1.0), 'S': (0.5, 0), 'SE': (1.0,

contains(x, y)

Returns whether (x, y) is in the bounding box or on its edge.

containsx(x)

Returns whether x is in the closed (x0, x1) interval.

containsy(y)

Returns whether y is in the closed (y0, y1) interval.

corners()

Return an array of points which are the four corners of this rectangle. For example, if this Bbox is defined by the points (a, b) and (c, d), corners() returns (a, b), (a, d), (c, b) and (c, d).

count_contains(vertices)

Count the number of vertices contained in the Bbox. Any vertices with a non-finite x or y value are ignored.

Parameters

vertices [Nx2 Numpy array.]

count_overlaps(bboxes)
Count the number of bounding boxes that overlap this one.

Parameters

bboxes [sequence of BboxBase objects]

expanded(sw, sh)
Return a new Bbox which is this Bbox expanded around its center by the given factors sw and sh.

extents
Returns (x0, y0, x1, y1).

frozen()
TransformNode is the base class for anything that participates in the transform tree and needs to invalidate its parents or be invalidated. This includes classes that are not really transforms, such as bounding boxes, since some transforms depend on bounding boxes to compute their values.

fully_contains(x, y)
Returns whether x, y is in the bounding box, but not on its edge.

fully_containsx(x)
Returns whether x is in the open (x0, x1) interval.

fully_containsy(y)
Returns whether y is in the open (y0, y1) interval.

fully_overlaps(other)
Returns whether this bounding box overlaps with the other bounding box, not including the edges.

Parameters

other [BboxBase]

get_points()

height
The height of the bounding box. It may be negative if $y1 < y0$.

static intersection(bbox1, bbox2)
Return the intersection of the two bboxes or None if they do not intersect.

intervalx
intervalx is the pair of x coordinates that define the bounding box. It is not guaranteed to be sorted from left to right.

intervaly
intervaly is the pair of y coordinates that define the bounding box. It is not guaranteed to be sorted from bottom to top.

`inverse_transformed(transform)`

Return a new Bbox object, statically transformed by the inverse of the given transform.

`is_affine = True`

`is_bbox = True`

`is_unit()`

Returns True if the Bbox is the unit bounding box from (0, 0) to (1, 1).

`max`

max is the top-right corner of the bounding box.

`min`

min is the bottom-left corner of the bounding box.

`overlaps(other)`

Returns whether this bounding box overlaps with the other bounding box.

Parameters

`other` [BboxBase]

`p0`

p0 is the first pair of (x, y) coordinates that define the bounding box. It is not guaranteed to be the bottom-left corner. For that, use min.

`p1`

p1 is the second pair of (x, y) coordinates that define the bounding box. It is not guaranteed to be the top-right corner. For that, use max.

`padded(p)`

Return a new Bbox that is padded on all four sides by the given value.

`rotated(radians)`

Return a new bounding box that bounds a rotated version of this bounding box by the given radians. The new bounding box is still aligned with the axes, of course.

`shrunk(mx, my)`

Return a copy of the Bbox, shrunk by the factor mx in the x direction and the factor my in the y direction. The lower left corner of the box remains unchanged. Normally mx and my will be less than 1, but this is not enforced.

`shrunk_to_aspect(box_aspect, container=None, fig_aspect=1.0)`

Return a copy of the Bbox, shrunk so that it is as large as it can be while having the desired aspect ratio, box_aspect. If the box coordinates are relative—that is, fractions of a larger box such as a figure—then the physical aspect ratio of that figure is specified with fig_aspect, so that box_aspect can also be given as a ratio of the absolute dimensions, not the relative dimensions.

`size`

The width and height of the bounding box. May be negative, in the same way as width and height.

`splitx(*args)`

e.g., `bbox.splitx(f1, f2, ...)`

Returns a list of new Bbox objects formed by splitting the original one with vertical lines at fractional positions `f1`, `f2`, ...

`splity(*args)`

e.g., `bbox.splity(f1, f2, ...)`

Returns a list of new Bbox objects formed by splitting the original one with horizontal lines at fractional positions `f1`, `f2`, ...

`transformed(transform)`

Return a new Bbox object, statically transformed by the given transform.

`translated(tx, ty)`

Return a copy of the Bbox, statically translated by `tx` and `ty`.

`static union(bboxes)`

Return a Bbox that contains all of the given bboxes.

`width`

The width of the bounding box. It may be negative if `x1 < x0`.

`x0`

`x0` is the first of the pair of x coordinates that define the bounding box. `x0` is not guaranteed to be less than `x1`. If you require that, use `xmin`.

`x1`

`x1` is the second of the pair of x coordinates that define the bounding box. `x1` is not guaranteed to be greater than `x0`. If you require that, use `xmax`.

`xmax`

`xmax` is the right edge of the bounding box.

`xmin`

`xmin` is the left edge of the bounding box.

`y0`

`y0` is the first of the pair of y coordinates that define the bounding box. `y0` is not guaranteed to be less than `y1`. If you require that, use `ymin`.

`y1`

`y1` is the second of the pair of y coordinates that define the bounding box. `y1` is not guaranteed to be greater than `y0`. If you require that, use `ymax`.

`ymax`

`ymax` is the top edge of the bounding box.

`ymin`

`ymin` is the bottom edge of the bounding box.

`class matplotlib.transforms.BboxTransform(boxin, boxout, **kwargs)`

Bases: `matplotlib.transforms.Affine2DBase`

`BboxTransform` linearly transforms points from one Bbox to another Bbox.

Create a new BboxTransform that linearly transforms points from boxin to boxout.

`get_matrix()`

Get the Affine transformation array for the affine part of this transform.

`is_separable = True`

`class matplotlib.transforms.BboxTransformFrom(boxin, **kwargs)`

Bases: `matplotlib.transforms.Affine2DBase`

BboxTransformFrom linearly transforms points from a given Bbox to the unit bounding box.

`get_matrix()`

Get the Affine transformation array for the affine part of this transform.

`is_separable = True`

`class matplotlib.transforms.BboxTransformTo(boxout, **kwargs)`

Bases: `matplotlib.transforms.Affine2DBase`

BboxTransformTo is a transformation that linearly transforms points from the unit bounding box to a given Bbox.

Create a new BboxTransformTo that linearly transforms points from the unit bounding box to boxout.

`get_matrix()`

Get the Affine transformation array for the affine part of this transform.

`is_separable = True`

`class matplotlib.transforms.BboxTransformToMaxOnly(boxout, **kwargs)`

Bases: `matplotlib.transforms.BboxTransformTo`

BboxTransformTo is a transformation that linearly transforms points from the unit bounding box to a given Bbox with a fixed upper left of (0, 0).

Create a new BboxTransformTo that linearly transforms points from the unit bounding box to boxout.

`get_matrix()`

Get the Affine transformation array for the affine part of this transform.

`class matplotlib.transforms.BlendedAffine2D(x_transform, y_transform, **kwargs)`

Bases: `matplotlib.transforms.Affine2DBase`

A "blended" transform uses one transform for the x-direction, and another transform for the y-direction.

This version is an optimization for the case where both child transforms are of type `Affine2DBase`.

Create a new "blended" transform using `x_transform` to transform the x-axis and `y_transform` to transform the y-axis.

Both `x_transform` and `y_transform` must be 2D affine transforms.

You will generally not call this constructor directly but use the `blended_transform_factory()` function instead, which can determine automatically which kind of blended transform to create.

`contains_branch_seperately(transform)`

Returns whether the given branch is a sub-tree of this transform on each separate dimension.

A common use for this method is to identify if a transform is a blended transform containing an axes' data transform. e.g.:

```
x_isdata, y_isdata = trans.contains_branch_seperately(ax.transData)
```

`get_matrix()`

Get the Affine transformation array for the affine part of this transform.

`is_separable = True`

```
class matplotlib.transforms.BlendedGenericTransform(x_transform, y_transform,
                                                    **kwargs)
```

Bases: `matplotlib.transforms.Transform`

A "blended" transform uses one transform for the x-direction, and another transform for the y-direction.

This "generic" version can handle any given child transform in the x- and y-directions.

Create a new "blended" transform using `x_transform` to transform the x-axis and `y_transform` to transform the y-axis.

You will generally not call this constructor directly but use the `blended_transform_factory()` function instead, which can determine automatically which kind of blended transform to create.

`contains_branch(other)`

Return whether the given transform is a sub-tree of this transform.

This routine uses transform equality to identify sub-trees, therefore in many situations it is object id which will be used.

For the case where the given transform represents the whole of this transform, returns `True`.

`contains_branch_seperately(transform)`

Returns whether the given branch is a sub-tree of this transform on each separate dimension.

A common use for this method is to identify if a transform is a blended transform containing an axes' data transform. e.g.:

`x_isdata, y_isdata = trans.contains_branch_seperately(ax.transData)`

`depth`

Returns the number of transforms which have been chained together to form this Transform instance.

Note: For the special case of a Composite transform, the maximum depth of the two is returned.

`frozen()`

Returns a frozen copy of this transform node. The frozen copy will not update when its children change. Useful for storing a previously known state of a transform where `copy.deepcopy()` might normally be used.

`get_affine()`

Get the affine part of this transform.

`has_inverse`

`input_dims = 2`

`inverted()`

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

`x === self.inverted().transform(self.transform(x))`

`is_affine`

`is_separable = True`

`output_dims = 2`

`pass_through = True`

`transform_non_affine(points)`

Performs only the non-affine part of the transformation.

`transform(values)` is always equivalent to `transform_affine(transform_non_affine(values))`.

In non-affine transformations, this is generally equivalent to `transform(values)`. In affine transformations, this is always a no-op.

Accepts a numpy array of shape (N x `input_dims`) and returns a numpy array of shape (N x `output_dims`).

Alternatively, accepts a numpy array of length `input_dims` and returns a numpy array of length `output_dims`.

```
class matplotlib.transforms.CompositeAffine2D(a, b, **kwargs)
```

Bases: `matplotlib.transforms.Affine2DBase`

A composite transform formed by applying transform `a` then transform `b`.

This version is an optimization that handles the case where both `a` and `b` are 2D affines.

Create a new composite transform that is the result of applying transform `a` then transform `b`.

Both `a` and `b` must be instances of `Affine2DBase`.

You will generally not call this constructor directly but use the `composite_transform_factory()` function instead, which can automatically choose the best kind of composite transform instance to create.

`depth`

Returns the number of transforms which have been chained together to form this Transform instance.

Note: For the special case of a Composite transform, the maximum depth of the two is returned.

`get_matrix()`

Get the Affine transformation array for the affine part of this transform.

```
class matplotlib.transforms.CompositeGenericTransform(a, b, **kwargs)
```

Bases: `matplotlib.transforms.Transform`

A composite transform formed by applying transform `a` then transform `b`.

This "generic" version can handle any two arbitrary transformations.

Create a new composite transform that is the result of applying transform `a` then transform `b`.

You will generally not call this constructor directly but use the `composite_transform_factory()` function instead, which can automatically choose the best kind of composite transform instance to create.

`depth`

Returns the number of transforms which have been chained together to form this Transform instance.

Note: For the special case of a Composite transform, the maximum depth of the two is returned.

`frozen()`

Returns a frozen copy of this transform node. The frozen copy will not update when

its children change. Useful for storing a previously known state of a transform where `copy.deepcopy()` might normally be used.

`get_affine()`

Get the affine part of this transform.

`has_inverse`

`inverted()`

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

`x === self.inverted().transform(self.transform(x))`

`is_affine`

`is_separable`

`pass_through = True`

`transform_affine(points)`

Performs only the affine part of this transformation on the given array of values.

`transform(values)` is always equivalent to `transform_affine(transform_non_affine(values))`.

In non-affine transformations, this is generally a no-op. In affine transformations, this is equivalent to `transform(values)`.

Accepts a numpy array of shape (N x input_dims) and returns a numpy array of shape (N x output_dims).

Alternatively, accepts a numpy array of length input_dims and returns a numpy array of length output_dims.

`transform_non_affine(points)`

Performs only the non-affine part of the transformation.

`transform(values)` is always equivalent to `transform_affine(transform_non_affine(values))`.

In non-affine transformations, this is generally equivalent to `transform(values)`. In affine transformations, this is always a no-op.

Accepts a numpy array of shape (N x input_dims) and returns a numpy array of shape (N x output_dims).

Alternatively, accepts a numpy array of length input_dims and returns a numpy array of length output_dims.

`transform_path_non_affine(path)`

Returns a path, transformed only by the non-affine part of this transform.

path: a Path instance.

`transform_path(path)` is equivalent to `transform_path_affine(transform_path_non_affine(values))`.

`class matplotlib.transforms.IdentityTransform(*args, **kwargs)`

Bases: `matplotlib.transforms.Affine2DBase`

A special class that does one thing, the identity transform, in a fast way.

`frozen()`

Returns a frozen copy of this transform node. The frozen copy will not update when its children change. Useful for storing a previously known state of a transform where `copy.deepcopy()` might normally be used.

`get_affine()`

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

`x === self.inverted().transform(self.transform(x))`

`get_matrix()`

Get the Affine transformation array for the affine part of this transform.

`inverted()`

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

`x === self.inverted().transform(self.transform(x))`

`transform(points)`

Performs only the non-affine part of the transformation.

`transform(values)` is always equivalent to `transform_affine(transform_non_affine(values))`.

In non-affine transformations, this is generally equivalent to `transform(values)`. In affine transformations, this is always a no-op.

Accepts a numpy array of shape (N x input_dims) and returns a numpy array of shape (N x output_dims).

Alternatively, accepts a numpy array of length input_dims and returns a numpy array of length output_dims.

`transform_affine(points)`

Performs only the non-affine part of the transformation.

`transform(values)` is always equivalent to `transform_affine(transform_non_affine(values))`.

In non-affine transformations, this is generally equivalent to `transform(values)`. In affine transformations, this is always a no-op.

Accepts a numpy array of shape (N x input_dims) and returns a numpy array of shape (N x output_dims).

Alternatively, accepts a numpy array of length input_dims and returns a numpy array of length output_dims.

`transform_non_affine(points)`

Performs only the non-affine part of the transformation.

`transform(values)` is always equivalent to `transform_affine(transform_non_affine(values))`.

In non-affine transformations, this is generally equivalent to `transform(values)`. In affine transformations, this is always a no-op.

Accepts a numpy array of shape (N x input_dims) and returns a numpy array of shape (N x output_dims).

Alternatively, accepts a numpy array of length input_dims and returns a numpy array of length output_dims.

`transform_path(path)`

Returns a path, transformed only by the non-affine part of this transform.

path: a Path instance.

`transform_path(path)` is equivalent to `transform_path_affine(transform_path_non_affine(values))`.

`transform_path_affine(path)`

Returns a path, transformed only by the non-affine part of this transform.

path: a Path instance.

`transform_path(path)` is equivalent to `transform_path_affine(transform_path_non_affine(values))`.

`transform_path_non_affine(path)`

Returns a path, transformed only by the non-affine part of this transform.

path: a Path instance.

`transform_path(path)` is equivalent to `transform_path_affine(transform_path_non_affine(values))`.

`class matplotlib.transforms.LockableBbox(bbox, x0=None, y0=None, x1=None, y1=None, **kwargs)`

Bases: `matplotlib.transforms.BboxBase`

A Bbox where some elements may be locked at certain values.

When the child bounding box changes, the bounds of this bbox will update accordingly with the exception of the locked elements.

Parameters

bbox [Bbox] The child bounding box to wrap.

x0 [float or None] The locked value for x0, or None to leave unlocked.

y0 [float or None] The locked value for y0, or None to leave unlocked.

x1 [float or None] The locked value for x1, or None to leave unlocked.

y1 [float or None] The locked value for y1, or None to leave unlocked.

`get_points()`

Get the points of the bounding box directly as a numpy array of the form: `[[x0, y0], [x1, y1]]`.

`locked_x0`
float or None: The value used for the locked x0.

`locked_x1`
float or None: The value used for the locked x1.

`locked_y0`
float or None: The value used for the locked y0.

`locked_y1`
float or None: The value used for the locked y1.

`class matplotlib.transforms.ScaledTranslation(xt, yt, scale_trans, **kwargs)`

Bases: `matplotlib.transforms.Affine2DBase`

A transformation that translates by `xt` and `yt`, after `xt` and `yt` have been transformed by the given transform `scale_trans`.

`get_matrix()`
Get the Affine transformation array for the affine part of this transform.

`class matplotlib.transforms.Transform(shorthand_name=None)`

Bases: `matplotlib.transforms.TransformNode`

The base class of all `TransformNode` instances that actually perform a transformation.

All non-affine transformations should be subclasses of this class. New affine transformations should be subclasses of `Affine2D`.

Subclasses of this class should override the following members (at minimum):

- `input_dims`
- `output_dims`
- `transform()`
- `is_separable`
- `has_inverse`
- `inverted()` (if `has_inverse` is `True`)

If the transform needs to do something non-standard with `matplotlib.path.Path` objects, such as adding curves where there were once line segments, it should override:

- `transform_path()`

Creates a new `TransformNode`.

Parameters

`shorthand_name` [str] A string representing the "name" of the transform.
The name carries no significance other than to improve the readability of `str(transform)` when `DEBUG=True`.

`contains_branch(other)`

Return whether the given transform is a sub-tree of this transform.

This routine uses transform equality to identify sub-trees, therefore in many situations it is object id which will be used.

For the case where the given transform represents the whole of this transform, returns True.

`contains_branch_seperately(other_transform)`

Returns whether the given branch is a sub-tree of this transform on each separate dimension.

A common use for this method is to identify if a transform is a blended transform containing an axes' data transform. e.g.:

```
x_isdata, y_isdata = trans.contains_branch_seperately(ax.transData)
```

`depth`

Returns the number of transforms which have been chained together to form this Transform instance.

Note: For the special case of a Composite transform, the maximum depth of the two is returned.

`get_affine()`

Get the affine part of this transform.

`get_matrix()`

Get the Affine transformation array for the affine part of this transform.

`has_inverse = False`

True if this transform has a corresponding inverse transform.

`input_dims = None`

The number of input dimensions of this transform. Must be overridden (with integers) in the subclass.

`inverted()`

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to self does not cause a corresponding update to its inverted copy.

`x === self.inverted().transform(self.transform(x))`

`is_separable = False`

True if this transform is separable in the x- and y- dimensions.

`output_dims = None`

The number of output dimensions of this transform. Must be overridden (with integers) in the subclass.

`transform(values)`

Performs the transformation on the given array of values.

Accepts a numpy array of shape (N x input_dims) and returns a numpy array of shape (N x output_dims).

Alternatively, accepts a numpy array of length input_dims and returns a numpy array of length output_dims.

`transform_affine(values)`

Performs only the affine part of this transformation on the given array of values.

`transform(values)` is always equivalent to `transform_affine(transform_non_affine(values))`.

In non-affine transformations, this is generally a no-op. In affine transformations, this is equivalent to `transform(values)`.

Accepts a numpy array of shape (N x input_dims) and returns a numpy array of shape (N x output_dims).

Alternatively, accepts a numpy array of length input_dims and returns a numpy array of length output_dims.

`transform_angles(angles, pts, radians=False, pushoff=1e-05)`

Performs transformation on a set of angles anchored at specific locations.

The angles must be a column vector (i.e., numpy array).

The pts must be a two-column numpy array of x,y positions (angle transforms currently only work in 2D). This array must have the same number of rows as angles.

radians indicates whether or not input angles are given in radians (True) or degrees (False; the default).

pushoff is the distance to move away from pts for determining transformed angles (see discussion of method below).

The transformed angles are returned in an array with the same size as angles.

The generic version of this method uses a very generic algorithm that transforms pts, as well as locations very close to pts, to find the angle in the transformed system.

`transform_bbox(bbox)`

Transform the given bounding box.

Note, for smarter transforms including caching (a common requirement for matplotlib figures), see `TransformedBbox`.

`transform_non_affine(values)`

Performs only the non-affine part of the transformation.

`transform(values)` is always equivalent to `transform_affine(transform_non_affine(values))`.

In non-affine transformations, this is generally equivalent to `transform(values)`. In affine transformations, this is always a no-op.

Accepts a numpy array of shape (N x input_dims) and returns a numpy array of shape (N x output_dims).

Alternatively, accepts a numpy array of length input_dims and returns a numpy array of length output_dims.

`transform_path(path)`

Returns a transformed path.

path: a Path instance.

In some cases, this transform may insert curves into the path that began as line segments.

`transform_path_affine(path)`

Returns a path, transformed only by the affine part of this transform.

path: a Path instance.

`transform_path(path)` is equivalent to `transform_path_affine(transform_path_non_affine(values))`.

`transform_path_non_affine(path)`

Returns a path, transformed only by the non-affine part of this transform.

path: a Path instance.

`transform_path(path)` is equivalent to `transform_path_affine(transform_path_non_affine(values))`.

`transform_point(point)`

A convenience function that returns the transformed copy of a single point.

The point is given as a sequence of length `input_dims`. The transformed point is returned as a sequence of length `output_dims`.

`class matplotlib.transforms.TransformNode(shorthand_name=None)`

Bases: `object`

`TransformNode` is the base class for anything that participates in the transform tree and needs to invalidate its parents or be invalidated. This includes classes that are not really transforms, such as bounding boxes, since some transforms depend on bounding boxes to compute their values.

Creates a new `TransformNode`.

Parameters

`shorthand_name` [str] A string representing the "name" of the transform.
The name carries no significance other than to improve the readability of `str(transform)` when `DEBUG=True`.

`INVALID = 3`

`INVALID_AFFINE = 2`

`INVALID_NON_AFFINE = 1`

`frozen()`

Returns a frozen copy of this transform node. The frozen copy will not update when its children change. Useful for storing a previously known state of a transform where `copy.deepcopy()` might normally be used.

`invalidate()`

Invalidate this `TransformNode` and triggers an invalidation of its ancestors. Should be called any time the transform changes.

`is_affine = False`

`is_bbox = False`

`pass_through = False`

If `pass_through` is `True`, all ancestors will always be invalidated, even if 'self' is already invalid.

`set_children(*children)`

Set the children of the transform, to let the invalidation system know which transforms can invalidate this transform. Should be called from the constructor of any transforms that depend on other transforms.

`class matplotlib.transforms.TransformWrapper(child)`

Bases: `matplotlib.transforms.Transform`

A helper class that holds a single child transform and acts equivalently to it.

This is useful if a node of the transform tree must be replaced at run time with a transform of a different type. This class allows that replacement to correctly trigger invalidation.

Note that `TransformWrapper` instances must have the same input and output dimensions during their entire lifetime, so the child transform may only be replaced with another child transform of the same dimensions.

`child`: A `class:Transform` instance. This child may later be replaced with `set()`.

`frozen()`

Returns a frozen copy of this transform node. The frozen copy will not update when its children change. Useful for storing a previously known state of a transform where `copy.deepcopy()` might normally be used.

`has_inverse`

`is_affine`

`is_separable`

`pass_through = True`

`set(child)`

Replace the current child of this transform with another one.

The new child must have the same number of input and output dimensions as the current child.

```
class matplotlib.transforms.TransformBbox(bbox, transform, **kwargs)
    Bases: matplotlib.transforms.BboxBase
```

A Bbox that is automatically transformed by a given transform. When either the child bounding box or transform changes, the bounds of this bbox will update accordingly.

Parameters

bbox [Bbox]

transform [Transform]

```
get_points()
```

Get the points of the bounding box directly as a numpy array of the form: `[[x0, y0], [x1, y1]]`.

```
class matplotlib.transforms.TransformPatchPath(patch)
    Bases: matplotlib.transforms.TransformPath
```

A TransformPatchPath caches a non-affine transformed copy of the Patch. This cached copy is automatically updated when the non-affine part of the transform or the patch changes.

Create a new TransformPatchPath from the given Patch.

```
class matplotlib.transforms.TransformPath(path, transform)
    Bases: matplotlib.transforms.TransformNode
```

A TransformPath caches a non-affine transformed copy of the Path. This cached copy is automatically updated when the non-affine part of the transform changes.

Note: Paths are considered immutable by this class. Any update to the path's vertices/codes will not trigger a transform recomputation.

Create a new TransformPath from the given Path and Transform.

```
get_affine()
```

```
get_fully_transformed_path()
```

Return a fully-transformed copy of the child path.

```
get_transformed_path_and_affine()
```

Return a copy of the child path, with the non-affine part of the transform already applied, along with the affine part of the path necessary to complete the transformation.

```
get_transformed_points_and_affine()
```

Return a copy of the child path, with the non-affine part of the transform already applied, along with the affine part of the path necessary to complete the transformation. Unlike `get_transformed_path_and_affine()`, no interpolation will be performed.

```
matplotlib.transforms.blended_transform_factory(x_transform, y_transform)
```

Create a new "blended" transform using `x_transform` to transform the x-axis and `y_transform` to transform the y-axis.

A faster version of the blended transform is returned for the case where both child transforms are affine.

`matplotlib.transforms.composite_transform_factory(a, b)`

Create a new composite transform that is the result of applying transform a then transform b.

Shortcut versions of the blended transform are provided for the case where both child transforms are affine, or one or the other is the identity transform.

Composite transforms may also be created using the '+' operator, e.g.:

```
c = a + b
```

`matplotlib.transforms.interval_contains(interval, val)`

Check, inclusively, whether an interval includes a given value.

Parameters

`interval` [sequence of scalar] A 2-length sequence, endpoints that define the interval.

`val` [scalar] Value to check is within interval.

Returns

`bool` Returns true if given val is within the interval.

`matplotlib.transforms.interval_contains_open(interval, val)`

Check, excluding endpoints, whether an interval includes a given value.

Parameters

`interval` [sequence of scalar] A 2-length sequence, endpoints that define the interval.

`val` [scalar] Value to check is within interval.

Returns

`bool` Returns true if given val is within the interval.

`matplotlib.transforms.nonsingular(vmin, vmax, expander=0.001, tiny=1e-15, increasing=True)`

Modify the endpoints of a range as needed to avoid singularities.

Parameters

`vmin, vmax` [float] The initial endpoints.

`expander` [float, optional, default: 0.001] Fractional amount by which vmin and vmax are expanded if the original interval is too small, based on tiny.

`tiny` [float, optional, default: 1e-15] Threshold for the ratio of the interval to the maximum absolute value of its endpoints. If the interval is smaller than this, it will be expanded. This value should be around 1e-15 or larger;

otherwise the interval will be approaching the double precision resolution limit.

increasing [bool, optional, default: True] If True, swap vmin, vmax if vmin > vmax.

Returns

vmin, vmax [float] Endpoints, expanded and/or swapped if necessary. If either input is inf or NaN, or if both inputs are 0 or very close to zero, it returns -expander, expander.

`matplotlib.transforms.offset_copy(trans, fig=None, x=0.0, y=0.0, units='inches')`

Return a new transform with an added offset.

Parameters

trans [Transform instance] Any transform, to which offset will be applied.

fig [Figure, optional, default: None] Current figure. It can be None if units are 'dots'.

x, y [float, optional, default: 0.0] Specifies the offset to apply.

units [{'inches', 'points', 'dots'}, optional] Units of the offset.

Returns

trans [Transform instance] Transform with applied offset.

TRIANGULAR GRIDS

71.1 matplotlib.tri

Unstructured triangular grid functions.

`class matplotlib.tri.Triangulation(x, y, triangles=None, mask=None)`

An unstructured triangular grid consisting of `npoints` points and `ntri` triangles. The triangles can either be specified by the user or automatically generated using a Delaunay triangulation.

Parameters

`x, y` [array_like of shape (`npoints`)] Coordinates of grid points.

`triangles` [integer array_like of shape (`ntri`, 3), optional] For each triangle, the indices of the three points that make up the triangle, ordered in an anticlockwise manner. If not specified, the Delaunay triangulation is calculated.

`mask` [boolean array_like of shape (`ntri`), optional] Which triangles are masked out.

Notes

For a `Triangulation` to be valid it must not have duplicate points, triangles formed from colinear points, or overlapping triangles.

Attributes

`'edges'`

`'neighbors'`

`is_delaunay` [bool] Whether the `Triangulation` is a calculated Delaunay triangulation (where triangles was not specified) or not.

`calculate_plane_coefficients(z)`

Calculate plane equation coefficients for all unmasked triangles from the point (`x,y`) coordinates and specified `z`-array of shape (`npoints`). Returned array has shape (`npoints,3`) and allows `z`-value at (`x,y`) position in triangle `tri` to be calculated using `z = array[tri,0]*x + array[tri,1]*y + array[tri,2]`.

edges

Return integer array of shape (nedges,2) containing all edges of non-masked triangles.

Each edge is the start point index and end point index. Each edge (start,end and end,start) appears only once.

get_cpp_triangulation()

static get_from_args_and_kwargs(*args, **kwargs)

Return a Triangulation object from the args and kwargs, and the remaining args and kwargs with the consumed values removed.

There are two alternatives: either the first argument is a Triangulation object, in which case it is returned, or the args and kwargs are sufficient to create a new Triangulation to return. In the latter case, see Triangulation.__init__ for the possible args and kwargs.

get_masked_triangles()

Return an array of triangles that are not masked.

get_trifinder()

Return the default matplotlib.tri.TriFinder of this triangulation, creating it if necessary. This allows the same TriFinder object to be easily shared.

neighbors

Return integer array of shape (ntri,3) containing neighbor triangles.

For each triangle, the indices of the three triangles that share the same edges, or -1 if there is no such neighboring triangle. neighbors[i,j] is the triangle that is the neighbor to the edge from point index triangles[i,j] to point index triangles[i,(j+1)%3].

set_mask(mask)

Set or clear the mask array. This is either None, or a boolean array of shape (ntri).

class matplotlib.tri.TriFinder(triangulation)

Abstract base class for classes used to find the triangles of a Triangulation in which (x,y) points lie.

Rather than instantiate an object of a class derived from TriFinder, it is usually better to use the function matplotlib.tri.Triangulation.get_trifinder().

Derived classes implement __call__(x,y) where x,y are array_like point coordinates of the same shape.

class matplotlib.tri.TrapezoidMapTriFinder(triangulation)

Bases: matplotlib.tri.trifinder.TriFinder

TriFinder class implemented using the trapezoid map algorithm from the book "Computational Geometry, Algorithms and Applications", second edition, by M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf.

The triangulation must be valid, i.e. it must not have duplicate points, triangles formed from colinear points, or overlapping triangles. The algorithm has some tolerance to triangles formed from colinear points, but this should not be relied upon.

class matplotlib.tri.TriInterpolator(triangulation, z, trifinder=None)

Abstract base class for classes used to perform interpolation on triangular grids.

Derived classes implement the following methods:

- `__call__(x, y)` , where x, y are array_like point coordinates of the same shape, and that returns a masked array of the same shape containing the interpolated z-values.
- `gradient(x, y)` , where x, y are array_like point coordinates of the same shape, and that returns a list of 2 masked arrays of the same shape containing the 2 derivatives of the interpolator (derivatives of interpolated z values with respect to x and y).

class matplotlib.tri.LinearTriInterpolator(triangulation, z, trifinder=None)

Bases: matplotlib.tri.triinterpolate.TriInterpolator

A LinearTriInterpolator performs linear interpolation on a triangular grid.

Each triangle is represented by a plane so that an interpolated value at point (x,y) lies on the plane of the triangle containing (x,y). Interpolated values are therefore continuous across the triangulation, but their first derivatives are discontinuous at edges between triangles.

Parameters

triangulation [Triangulation object] The triangulation to interpolate over.

z [array_like of shape (npoints,)] Array of values, defined at grid points, to interpolate between.

trifinder [TriFinder object, optional] If this is not specified, the Triangulation's default TriFinder will be used by calling matplotlib.tri.Triangulation.get_trifinder().

Methods

'__call__' (x, y)	(Returns interpolated values at x,y points)
'gradient' (x, y)	(Returns interpolated derivatives at x,y points)

gradient(x, y)

Returns a list of 2 masked arrays containing interpolated derivatives at the specified x,y points.

Parameters

x, y [array-like] x and y coordinates of the same shape and any number of dimensions.

Returns

dzdx, dzdy [np.ma.array] 2 masked arrays of the same shape as x and y ; values corresponding to (x,y) points outside of the triangulation are masked out. The first returned array contains the values of $\frac{\partial z}{\partial x}$ and the second those of $\frac{\partial z}{\partial y}$.

```
class matplotlib.tri.CubicTriInterpolator(triangulation, z, kind='min_E', trfinder=None,
                                          dz=None)
```

Bases: matplotlib.tri.triinterpolate.TriInterpolator

A CubicTriInterpolator performs cubic interpolation on triangular grids.

In one-dimension - on a segment - a cubic interpolating function is defined by the values of the function and its derivative at both ends. This is almost the same in 2-d inside a triangle, except that the values of the function and its 2 derivatives have to be defined at each triangle node.

The CubicTriInterpolator takes the value of the function at each node - provided by the user - and internally computes the value of the derivatives, resulting in a smooth interpolation. (As a special feature, the user can also impose the value of the derivatives at each node, but this is not supposed to be the common usage.)

Parameters

- triangulation [Triangulation object] The triangulation to interpolate over.
- z [array_like of shape (npoints,)] Array of values, defined at grid points, to interpolate between.
- kind [{ 'min_E', 'geom', 'user' }, optional] Choice of the smoothing algorithm, in order to compute the interpolant derivatives (defaults to 'min_E'):
- if 'min_E': (default) The derivatives at each node is computed to minimize a bending energy.
 - if 'geom': The derivatives at each node is computed as a weighted average of relevant triangle normals. To be used for speed optimization (large grids).
 - if 'user': The user provides the argument dz, no computation is hence needed.
- trfinder [TriFinder object, optional] If not specified, the Triangulation's default TriFinder will be used by calling matplotlib.tri.Triangulation.get_trifinder().
- dz [tuple of array_likes (dzdx, dzdy), optional] Used only if kind = 'user'. In this case dz must be provided as (dzdx, dzdy) where dzdx, dzdy are arrays of the same shape as z and are the interpolant first derivatives at the triangulation points.

Notes

This note is a bit technical and details the way a CubicTriInterpolator computes a cubic interpolation.

The interpolation is based on a Clough-Tocher subdivision scheme of the triangulation mesh (to make it clearer, each triangle of the grid will be divided in 3 child-triangles, and on each child triangle the interpolated function is a cubic polynomial of the 2 coordinates). This technique originates from FEM (Finite Element Method) analysis; the element used is a

reduced Hsieh-Clough-Tocher (HCT) element. Its shape functions are described in [?]. The assembled function is guaranteed to be C1-smooth, i.e. it is continuous and its first derivatives are also continuous (this is easy to show inside the triangles but is also true when crossing the edges).

In the default case (kind = 'min_E'), the interpolant minimizes a curvature energy on the functional space generated by the HCT element shape functions - with imposed values but arbitrary derivatives at each node. The minimized functional is the integral of the so-called total curvature (implementation based on an algorithm from [?] - PCG sparse solver):

$$E(z) = \frac{1}{2} \int_{\Omega} \left(\left(\frac{\partial^2 z}{\partial x^2} \right)^2 + \left(\frac{\partial^2 z}{\partial y^2} \right)^2 + 2 \left(\frac{\partial^2 z}{\partial y \partial x} \right)^2 \right) dx dy$$

If the case kind = 'geom' is chosen by the user, a simple geometric approximation is used (weighted average of the triangle normal vectors), which could improve speed on very large grids.

References

[?], [?]

Methods

'__call__' (x, y)	(Returns interpolated values at x,y points)
'gradient' (x, y)	(Returns interpolated derivatives at x,y points)

gradient(x, y)

Returns a list of 2 masked arrays containing interpolated derivatives at the specified x,y points.

Parameters

x, y [array-like] x and y coordinates of the same shape and any number of dimensions.

Returns

dzdx, dzdy [np.ma.array] 2 masked arrays of the same shape as x and y ; values corresponding to (x,y) points outside of the triangulation are masked out. The first returned array contains the values of $\frac{\partial z}{\partial x}$ and the second those of $\frac{\partial z}{\partial y}$.

class matplotlib.tri.TriRefiner(triangulation)

Abstract base class for classes implementing mesh refinement.

A TriRefiner encapsulates a Triangulation object and provides tools for mesh refinement and interpolation.

Derived classes must implements:

- `refine_triangulation(return_tri_index=False, **kwargs)` , where the optional keyword arguments `kwargs` are defined in each `TriRefiner` concrete implementation, and which returns :
 - a refined triangulation
 - optionally (depending on `return_tri_index`), for each point of the refined triangulation: the index of the initial triangulation triangle to which it belongs.
- `refine_field(z, triinterpolator=None, **kwargs)` , where:
 - `z` array of field values (to refine) defined at the base triangulation nodes
 - `triinterpolator` is a `TriInterpolator` (optional)
 - the other optional keyword arguments `kwargs` are defined in each `TriRefiner` concrete implementation

and which returns (as a tuple) a refined triangular mesh and the interpolated values of the field at the refined triangulation nodes.

`class matplotlib.tri.UniformTriRefiner(triangulation)`

Bases: `matplotlib.tri.trirefine.TriRefiner`

Uniform mesh refinement by recursive subdivisions.

Parameters

`triangulation` [`Triangulation`] The encapsulated triangulation (to be refined)

`refine_field(z, triinterpolator=None, subdiv=3)`

Refines a field defined on the encapsulated triangulation.

Returns `refi_tri` (refined triangulation), `refi_z` (interpolated values of the field at the node of the refined triangulation).

Parameters

`z` [1d-array-like of length `n_points`] Values of the field to refine, defined at the nodes of the encapsulated triangulation. (`n_points` is the number of points in the initial triangulation)

`triinterpolator` [`TriInterpolator`, optional] Interpolator used for field interpolation. If not specified, a `CubicTriInterpolator` will be used.

`subdiv` [integer, optional] Recursion level for the subdivision. Defaults to 3. Each triangle will be divided into `4**subdiv` child triangles.

Returns

`refi_tri` [`Triangulation` object] The returned refined triangulation

`refi_z` [1d array of length: `refi_tri` node count.] The returned interpolated field (at `refi_tri` nodes)

`refine_triangulation(return_tri_index=False, subdiv=3)`

Computes an uniformly refined triangulation `refi_triangulation` of the encapsulated triangulation.

This function refines the encapsulated triangulation by splitting each father triangle into 4 child sub-triangles built on the edges midside nodes, recursively (level of recursion `subdiv`). In the end, each triangle is hence divided into $4^{**}subdiv$ child triangles. The default value for `subdiv` is 3 resulting in 64 refined subtriangles for each triangle of the initial triangulation.

Parameters

`return_tri_index` [boolean, optional] Boolean indicating whether an index table indicating the father triangle index of each point will be returned. Default value False.

`subdiv` [integer, optional] Recursion level for the subdivision. Defaults value 3. Each triangle will be divided into $4^{**}subdiv$ child triangles.

Returns

`refi_triangulation` [Triangulation] The returned refined triangulation

`found_index` [array-like of integers] Index of the initial triangulation containing triangle, for each point of `refi_triangulation`. Returned only if `return_tri_index` is set to True.

`class matplotlib.tri.TriAnalyzer(triangulation)`

Define basic tools for triangular mesh analysis and improvement.

A TriAnalyzer encapsulates a Triangulation object and provides basic tools for mesh analysis and mesh improvement.

Parameters

`triangulation` [Triangulation object] The encapsulated triangulation to analyze.

Attributes

`'scale_factors'`

`circle_ratios(rescale=True)`

Returns a measure of the triangulation triangles flatness.

The ratio of the incircle radius over the circumcircle radius is a widely used indicator of a triangle flatness. It is always ≤ 0.5 and $= 0.5$ only for equilateral triangles. Circle ratios below 0.01 denote very flat triangles.

To avoid unduly low values due to a difference of scale between the 2 axis, the triangular mesh can first be rescaled to fit inside a unit square with `scale_factors` (Only if `rescale` is True, which is its default value).

Parameters

`rescale` [boolean, optional] If True, a rescaling will be internally performed (based on `scale_factors`, so that the (unmasked) triangles fit exactly inside a unit square mesh. Default is True.

Returns

`circle_ratios` [masked array] Ratio of the incircle radius over the circumcircle radius, for each 'rescaled' triangle of the encapsulated triangulation. Values corresponding to masked triangles are masked out.

`get_flat_tri_mask(min_circle_ratio=0.01, rescale=True)`

Eliminates excessively flat border triangles from the triangulation.

Returns a mask `new_mask` which allows to clean the encapsulated triangulation from its border-located flat triangles (according to their `circle_ratios()`). This mask is meant to be subsequently applied to the triangulation using `matplotlib.tri.Triangulation.set_mask()`. `new_mask` is an extension of the initial triangulation mask in the sense that an initially masked triangle will remain masked.

The `new_mask` array is computed recursively ; at each step flat triangles are removed only if they share a side with the current mesh border. Thus no new holes in the triangulated domain will be created.

Parameters

`min_circle_ratio` [float, optional] Border triangles with incircle/circumcircle radii ratio r/R will be removed if $r/R < \text{min_circle_ratio}$. Default value: 0.01

`rescale` [boolean, optional] If True, a rescaling will first be internally performed (based on `scale_factors`), so that the (unmasked) triangles fit exactly inside a unit square mesh. This rescaling accounts for the difference of scale which might exist between the 2 axis. Default (and recommended) value is True.

Returns

`new_mask` [array-like of booleans] Mask to apply to encapsulated triangulation. All the initially masked triangles remain masked in the `new_mask`.

Notes

The rationale behind this function is that a Delaunay triangulation - of an unstructured set of points - sometimes contains almost flat triangles at its border, leading to artifacts in plots (especially for high-resolution contouring). Masked with computed `new_mask`, the encapsulated triangulation would contain no more unmasked border triangles with a circle ratio below `min_circle_ratio`, thus improving the mesh quality for subsequent plots or interpolation.

`scale_factors`

Factors to rescale the triangulation into a unit square.

Returns k, tuple of 2 scale factors.

Returns

k [tuple of 2 floats (kx, ky)] Tuple of floats that would rescale the triangulation : [triangulation.x * kx, triangulation.y * ky] fits exactly inside a unit square.

TYPE1FONT

72.1 matplotlib.type1font

This module contains a class representing a Type 1 font.

This version reads pfa and pfb files and splits them for embedding in pdf files. It also supports SlantFont and ExtendFont transformations, similarly to pdfTeX and friends. There is no support yet for subsetting.

Usage:

```
>>> font = Type1Font(filename)
>>> clear_part, encrypted_part, finale = font.parts
>>> slanted_font = font.transform({'slant': 0.167})
>>> extended_font = font.transform({'extend': 1.2})
```

Sources:

- Adobe Technical Note #5040, Supporting Downloadable PostScript Language Fonts.
- Adobe Type 1 Font Format, Adobe Systems Incorporated, third printing, v1.1, 1993. ISBN 0-201-57044-0.

class matplotlib.type1font.Type1Font(input)

Bases: `object`

A class representing a Type-1 font, for use by backends.

Attributes

parts [tuple] A 3-tuple of the cleartext part, the encrypted part, and the finale of zeros.

prop [Dict[str, Any]] A dictionary of font properties.

Initialize a Type-1 font. input can be either the file name of a pfb file or a 3-tuple of already-decoded Type-1 font parts.

parts

`prop`

`transform(effects)`

Transform the font by slanting or extending. `effects` should be a dict where `effects['slant']` is the tangent of the angle that the font is to be slanted to the right (so negative values slant to the left) and `effects['extend']` is the multiplier by which the font is to be extended (so values less than 1.0 condense). Returns a new `Type1Font` object.

`matplotlib.type1font.ord(x)`

73.1 matplotlib.units

The classes here provide support for using custom classes with Matplotlib, e.g., those that do not expose the array interface but know how to convert themselves to arrays. It also supports classes with units and units conversion. Use cases include converters for custom objects, e.g., a list of datetime objects, as well as for objects that are unit aware. We don't assume any particular units implementation; rather a units implementation must provide the register with the Registry converter dictionary and a ConversionInterface. For example, here is a complete implementation which supports plotting with native datetime objects:

```
import matplotlib.units as units
import matplotlib.dates as dates
import matplotlib.ticker as ticker
import datetime

class DateConverter(units.ConversionInterface):

    @staticmethod
    def convert(value, unit, axis):
        'Convert a datetime value to a scalar or array'
        return dates.date2num(value)

    @staticmethod
    def axisinfo(unit, axis):
        'Return major and minor tick locators and formatters'
        if unit != 'date': return None
        majloc = dates.AutoDateLocator()
        majfmt = dates.AutoDateFormatter(majloc)
        return AxisInfo(majloc=majloc,
                        majfmt=majfmt,
                        label='date')

    @staticmethod
    def default__units(x, axis):
```

(continues on next page)

(continued from previous page)

```
'Return the default unit for x or None'
return 'date'

# Finally we register our object type with the Matplotlib units registry.
units.registry[datetime.date] = DateConverter()
```

```
class matplotlib.units.AxisInfo(majloc=None, minloc=None, majfmt=None, minfmt=None,
                                label=None, default_limits=None)
```

Bases: `object`

Information to support default axis labeling, tick labeling, and default limits. An instance of this class must be returned by `ConversionInterface.axisinfo()`.

Parameters

`majloc, minloc` [`Locator`, optional] Tick locators for the major and minor ticks.

`majfmt, minfmt` [`Formatter`, optional] Tick formatters for the major and minor ticks.

`label` [`str`, optional] The default axis label.

`default_limits` [optional] The default min and max limits of the axis if no data has been plotted.

Notes

If any of the above are `None`, the axis will simply use the default value.

```
class matplotlib.units.ConversionInterface
```

Bases: `object`

The minimal interface for a converter to take custom data types (or sequences) and convert them to values Matplotlib can use.

```
static axisinfo(unit, axis)
```

Return an `AxisInfo` instance for the axis with the specified units.

```
static convert(obj, unit, axis)
```

Convert `obj` using `unit` for the specified axis. If `obj` is a sequence, return the converted sequence. The output must be a sequence of scalars that can be used by the numpy array layer.

```
static default_units(x, axis)
```

Return the default unit for `x` or `None` for the given axis.

```
static is_numlike(x)
```

The Matplotlib `datalim`, `autoscaling`, `locators` etc work with scalars which are the units converted to floats given the current unit. The converter may be passed these floats, or arrays of them, even when units are set.

class matplotlib.units.Registry

Bases: [dict](#)

A register that maps types to conversion interfaces.

`get_converter(x)`

Get the converter for data that has the same type as x. If no converters are registered for x, returns None.

WIDGETS

74.1 matplotlib.widgets

74.1.1 GUI neutral widgets

Widgets that are designed to work for any of the GUI backends. All of these widgets require you to predefine a `matplotlib.axes.Axes` instance and pass that as the first arg. `matplotlib` doesn't try to be too smart with respect to layout – you will have to figure out how wide and tall you want your `Axes` to be to accommodate your widget.

```
class matplotlib.widgets.AxesWidget(ax)
```

Bases: `matplotlib.widgets.Widget`

Widget that is connected to a single `Axes`.

To guarantee that the widget remains responsive and not garbage-collected, a reference to the object should be maintained by the user.

This is necessary because the callback registry maintains only weak-refs to the functions, which are member functions of the widget. If there are no references to the widget object it may be garbage collected which will disconnect the callbacks.

Attributes:

`ax` [`Axes`] The parent axes for the widget

`canvas` [`FigureCanvasBase` subclass] The parent figure canvas for the widget.

`active` [`bool`] If `False`, the widget does not respond to events.

```
connect_event(event, callback)
```

Connect callback with an event.

This should be used in lieu of `figure.canvas.mpl_connect` since this function stores callback ids for later clean up.

```
disconnect_events()
```

Disconnect all events created by this widget.

```
class matplotlib.widgets.Button(ax, label, image=None, color='0.85', hovercolor='0.95')
```

Bases: `matplotlib.widgets.AxesWidget`

A GUI neutral button.

For the button to remain responsive you must keep a reference to it. Call `on_clicked()` to connect to the button.

Attributes

`ax` : The `matplotlib.axes.Axes` the button renders into.

`label` : A `matplotlib.text.Text` instance.

`color` : The color of the button when not hovering.

`hovercolor` : The color of the button when hovering.

Parameters

`ax` [`matplotlib.axes.Axes`] The `matplotlib.axes.Axes` instance the button will be placed into.

`label` [`str`] The button text. Accepts string.

`image` [`array`, `mpl image`, `Pillow Image`] The image to place in the button, if not `None`. Can be any legal arg to `imshow` (numpy array, `matplotlib Image` instance, or `Pillow Image`).

`color` [`color`] The color of the button when not activated

`hovercolor` [`color`] The color of the button when the mouse is over it

`disconnect(cid)`

remove the observer with connection id `cid`

`on_clicked(func)`

When the button is clicked, call this func with event.

A connection id is returned. It can be used to disconnect the button from its callback.

`class matplotlib.widgets.CheckButtons(ax, labels, actives)`

Bases: `matplotlib.widgets.AxesWidget`

A GUI neutral set of check buttons.

For the check buttons to remain responsive you must keep a reference to this object.

The following attributes are exposed

`ax` The `matplotlib.axes.Axes` instance the buttons are located in

`labels` List of `matplotlib.text.Text` instances

`lines` List of (`line1`, `line2`) tuples for the x's in the check boxes. These lines exist for each box, but have `set_visible(False)` when its box is not checked.

`rectangles` List of `matplotlib.patches.Rectangle` instances

Connect to the `CheckButtons` with the `on_clicked()` method

Add check buttons to `matplotlib.axes.Axes` instance `ax`

`labels` A `len(buttons)` list of labels as strings

`actives`

A `len(buttons)` list of booleans indicating whether the button is active

`disconnect(cid)`

remove the observer with connection id `cid`

`get_status()`

returns a tuple of the status (`True/False`) of all of the check buttons

`on_clicked(func)`

When the button is clicked, call `func` with button label

A connection id is returned which can be used to disconnect

`set_active(index)`

Directly (de)activate a check button by index.

`index` is an index into the original label list that this object was constructed with.

Raises `ValueError` if `index` is invalid.

Callbacks will be triggered if `eventson` is `True`.

```
class matplotlib.widgets.Cursor(ax, horizOn=True, vertOn=True, useblit=False, **line-
                                props)
```

Bases: `matplotlib.widgets.AxesWidget`

A horizontal and vertical line that spans the axes and moves with the pointer. You can turn off the `hline` or `vline` respectively with the following attributes:

`horizOn` Controls the visibility of the horizontal line

`vertOn` Controls the visibility of the horizontal line

and the visibility of the cursor itself with the `visible` attribute.

For the cursor to remain responsive you must keep a reference to it.

Add a cursor to `ax`. If `useblit=True`, use the backend-dependent blitting features for faster updates. `lineprops` is a dictionary of line properties.

`clear(event)`

clear the cursor

`onmove(event)`

on mouse motion draw the cursor if visible

```
class matplotlib.widgets.EllipseSelector(ax, onselect, drawtype='box', minspanx=None,
                                         minspany=None, useblit=False, lineprops=None,
                                         rectprops=None, spancoords='data', button=None,
                                         maxdist=10, marker_props=None, interac-
                                         tive=False, state_modifier_keys=None)
```

Bases: `matplotlib.widgets.RectangleSelector`

Select an elliptical region of an axes.

For the cursor to remain responsive you must keep a reference to it.

Example usage:

```
from matplotlib.widgets import EllipseSelector
from pylab import *

def onselect(eclick, erelease):
    'eclick and erelease are matplotlib events at press and release'
    print(' startposition : (%f, %f)' % (eclick.xdata, eclick.ydata))
    print(' endposition   : (%f, %f)' % (erelease.xdata, erelease.ydata))
    print(' used button   : ', eclick.button)

def toggle_selector(event):
    print(' Key pressed.')
    if event.key in ['Q', 'q'] and toggle_selector.ES.active:
        print(' EllipseSelector deactivated.')
        toggle_selector.RS.set_active(False)
    if event.key in ['A', 'a'] and not toggle_selector.ES.active:
        print(' EllipseSelector activated.')
        toggle_selector.ES.set_active(True)

x = arange(100)/(99.0)
y = sin(x)
fig = figure
ax = subplot(111)
ax.plot(x,y)

toggle_selector.ES = EllipseSelector(ax, onselect, drawtype='line')
connect('key_press_event', toggle_selector)
show()
```

Create a selector in ax. When a selection is made, clear the span and call onselect with:

```
onselect(pos_1, pos_2)
```

and clear the drawn box/line. The pos_1 and pos_2 are arrays of length 2 containing the x- and y-coordinate.

If minspanx is not None then events smaller than minspanx in x direction are ignored (it's the same for y).

The rectangle is drawn with rectprops; default:

```
rectprops = dict(facecolor='red', edgecolor = 'black',
                  alpha=0.2, fill=True)
```

The line is drawn with lineprops; default:

```
lineprops = dict(color='black', linestyle='-',
                 linewidth = 2, alpha=0.5)
```

Use drawtype if you want the mouse to draw a line, a box or nothing between click and actual position by setting

drawtype = 'line', drawtype='box' or drawtype = 'none'. Drawing a line would result in a line from vertex A to vertex C in a rectangle ABCD.

spancoords is one of 'data' or 'pixels'. If 'data', minspanx and minspany will be interpreted in the same coordinates as the x and y axis. If 'pixels', they are in pixels.

button is a list of integers indicating which mouse buttons should be used for rectangle selection. You can also specify a single integer if only a single button is desired. Default is None, which does not limit which button can be used.

Note, typically: 1 = left mouse button 2 = center mouse button (scroll wheel) 3 = right mouse button

interactive will draw a set of handles and allow you interact with the widget after it is drawn.

state_modifier_keys are keyboard modifiers that affect the behavior of the widget.

The defaults are: dict(move=' ', clear='escape', square='shift', center='ctrl')

Keyboard modifiers, which: 'move': Move the existing shape. 'clear': Clear the current shape. 'square': Makes the shape square. 'center': Make the initial point the center of the shape. 'square' and 'center' can be combined.

draw_shape(extents)

```
class matplotlib.widgets.Lasso(ax, xy, callback=None, useblit=True)
```

Bases: matplotlib.widgets.AxesWidget

Selection curve of an arbitrary shape.

The selected path can be used in conjunction with contains_point() to select data points from an image.

Unlike LassoSelector, this must be initialized with a starting point xy, and the Lasso events are destroyed upon release.

Parameters

ax [Axes] The parent axes for the widget.

xy [array] Coordinates of the start of the lasso.

callback [callable] Whenever the lasso is released, the callback function is called and passed the vertices of the selected path.

onmove(event)

onrelease(event)

```
class matplotlib.widgets.LassoSelector(ax, onselect=None, useblit=True, lineprops=None,
                                       button=None)
    Bases: matplotlib.widgets._SelectorWidget
```

Selection curve of an arbitrary shape.

For the selector to remain responsive you must keep a reference to it.

The selected path can be used in conjunction with `contains__point` to select data points from an image.

In contrast to Lasso, LassoSelector is written with an interface similar to RectangleSelector and SpanSelector, and will continue to interact with the axes until disconnected.

Example usage:

```
ax = subplot(111)
ax.plot(x,y)

def onselect(verts):
    print(verts)
lasso = LassoSelector(ax, onselect)
```

Parameters

`ax` [Axes] The parent axes for the widget.

`onselect` [function] Whenever the lasso is released, the `onselect` function is called and passed the vertices of the selected path.

`button` [List[Int], optional] A list of integers indicating which mouse buttons should be used for rectangle selection. You can also specify a single integer if only a single button is desired. Default is `None`, which does not limit which button can be used.

Note, typically:

- 1 = left mouse button
- 2 = center mouse button (scroll wheel)
- 3 = right mouse button

onpress(event)

onrelease(event)

```
class matplotlib.widgets.LockDraw
    Bases: object
```

Some widgets, like the cursor, draw onto the canvas, and this is not desirable under all circumstances, like when the toolbar is in zoom-to-rect mode and drawing a rectangle. The module level "lock" allows someone to grab the lock and prevent other widgets from drawing. Use `matplotlib.widgets.lock(someobj)` to prevent other widgets from drawing while you're interacting with the canvas.

`available(o)`
drawing is available to o

`isowner(o)`
Return True if o owns this lock

`locked()`
Return True if the lock is currently held by an owner

`release(o)`
release the lock

`class matplotlib.widgets.MultiCursor(canvas, axes, useblit=True, horizOn=False, vertOn=True, **lineprops)`

Bases: `matplotlib.widgets.Widget`

Provide a vertical (default) and/or horizontal line cursor shared between multiple axes.

For the cursor to remain responsive you must keep a reference to it.

Example usage:

```
from matplotlib.widgets import MultiCursor
from pylab import figure, show, np

t = np.arange(0.0, 2.0, 0.01)
s1 = np.sin(2*np.pi*t)
s2 = np.sin(4*np.pi*t)
fig = figure()
ax1 = fig.add_subplot(211)
ax1.plot(t, s1)

ax2 = fig.add_subplot(212, sharex=ax1)
ax2.plot(t, s2)

multi = MultiCursor(fig.canvas, (ax1, ax2), color='r', lw=1,
                    horizOn=False, vertOn=True)
show()
```

`clear(event)`
clear the cursor

`connect()`
connect events

`disconnect()`
disconnect events

`onmove(event)`

```
class matplotlib.widgets.PolygonSelector(ax, onselect, useblit=False, lineprops=None, markerprops=None, vertex_select_radius=15)
Bases: matplotlib.widgets._SelectorWidget
```

Select a polygon region of an axes.

Place vertices with each mouse click, and make the selection by completing the polygon (clicking on the first vertex). Hold the ctrl key and click and drag a vertex to reposition it (the ctrl key is not necessary if the polygon has already been completed). Hold the shift key and click and drag anywhere in the axes to move all vertices. Press the esc key to start a new polygon.

For the selector to remain responsive you must keep a reference to it.

Parameters

`ax` [Axes] The parent axes for the widget.

`onselect` [function] When a polygon is completed or modified after completion, the `onselect` function is called and passed a list of the vertices as (xdata, ydata) tuples.

`useblit` [bool, optional]

`lineprops` [dict, optional] The line for the sides of the polygon is drawn with the properties given by `lineprops`. The default is `dict(color='k', linestyle='-', linewidth=2, alpha=0.5)`.

`markerprops` [dict, optional] The markers for the vertices of the polygon are drawn with the properties given by `markerprops`. The default is `dict(marker='o', markersize=7, mec='k', mfc='k', alpha=0.5)`.

`vertex_select_radius` [float, optional] A vertex is selected (to complete the polygon or to move a vertex) if the mouse click is within `vertex_select_radius` pixels of the vertex. The default radius is 15 pixels.

Examples

`/gallery/widgets/polygon_selector_demo`

`onmove(event)`

Cursor move event handler and validator

`verts`

Get the polygon vertices.

Returns

`list` A list of the vertices of the polygon as (xdata, ydata) tuples.

```
class matplotlib.widgets.RadioButtons(ax, labels, active=0, activecolor='blue')
Bases: matplotlib.widgets.AxesWidget
```

A GUI neutral radio button.

For the buttons to remain responsive you must keep a reference to this object.

The following attributes are exposed:

- `ax` The matplotlib.axes.Axes instance the buttons are in
- `activecolor` The color of the button when clicked
- `labels` A list of matplotlib.text.Text instances
- `circles` A list of matplotlib.patches.Circle instances
- `value_selected` A string listing the current value selected

Connect to the RadioButtons with the `on_clicked()` method

Add radio buttons to matplotlib.axes.Axes instance `ax`

`labels` A `len(buttons)` list of labels as strings

`active` The index into labels for the button that is active

`activecolor` The color of the button when clicked

`disconnect(cid)`
remove the observer with connection id `cid`

`on_clicked(func)`
When the button is clicked, call `func` with button label
A connection id is returned which can be used to disconnect

`set_active(index)`
Trigger which radio button to make active.
`index` is an index into the original label list that this object was constructed with.
Raise `ValueError` if the index is invalid.

Callbacks will be triggered if `eventson` is `True`.

```
class matplotlib.widgets.RectangleSelector(ax, onselect, drawtype='box', minspanx=None,
                                         minspany=None, useblit=False, lineprops=None,
                                         rectprops=None, spancoords='data', but-
                                         ton=None, maxdist=10, marker_props=None,
                                         interactive=False, state_modifier_keys=None)
```

Bases: `matplotlib.widgets._SelectorWidget`

Select a rectangular region of an axes.

For the cursor to remain responsive you must keep a reference to it.

Example usage:

```
from matplotlib.widgets import RectangleSelector
from pylab import *
```

(continues on next page)

(continued from previous page)

```

def onselect(eclick, erelease):
    'eclick and erelease are matplotlib events at press and release'
    print(' startposition : (%f, %f)' % (eclick.xdata, eclick.ydata))
    print(' endposition   : (%f, %f)' % (erelease.xdata, erelease.ydata))
    print(' used button   : ', eclick.button)

def toggle_selector(event):
    print(' Key pressed.')
    if event.key in ['Q', 'q'] and toggle_selector.RS.active:
        print(' RectangleSelector deactivated.')
        toggle_selector.RS.set_active(False)
    if event.key in ['A', 'a'] and not toggle_selector.RS.active:
        print(' RectangleSelector activated.')
        toggle_selector.RS.set_active(True)

x = arange(100)/(99.0)
y = sin(x)
fig = figure
ax = subplot(111)
ax.plot(x,y)

toggle_selector.RS = RectangleSelector(ax, onselect, drawtype='line')
connect('key_press_event', toggle_selector)
show()

```

Create a selector in ax. When a selection is made, clear the span and call onselect with:

```
onselect(pos_1, pos_2)
```

and clear the drawn box/line. The pos_1 and pos_2 are arrays of length 2 containing the x- and y-coordinate.

If minspanx is not None then events smaller than minspanx in x direction are ignored (it's the same for y).

The rectangle is drawn with rectprops; default:

```
rectprops = dict(facecolor='red', edgecolor = 'black',
                 alpha=0.2, fill=True)
```

The line is drawn with lineprops; default:

```
lineprops = dict(color='black', linestyle='-',
                 linewidth = 2, alpha=0.5)
```

Use drawtype if you want the mouse to draw a line, a box or nothing between click and actual position by setting

`drawtype = 'line'`, `drawtype='box'` or `drawtype = 'none'`. Drawing a line would result in a line from vertex A to vertex C in a rectangle ABCD.

`spancoords` is one of `'data'` or `'pixels'`. If `'data'`, `minspanx` and `minspany` will be interpreted in the same coordinates as the x and y axis. If `'pixels'`, they are in pixels.

`button` is a list of integers indicating which mouse buttons should be used for rectangle selection. You can also specify a single integer if only a single button is desired. Default is `None`, which does not limit which button can be used.

Note, typically: 1 = left mouse button 2 = center mouse button (scroll wheel) 3 = right mouse button

`interactive` will draw a set of handles and allow you interact with the widget after it is drawn.

`state_modifier_keys` are keyboard modifiers that affect the behavior of the widget.

The defaults are: `dict(move=' ', clear='escape', square='shift', center='ctrl')`

Keyboard modifiers, which: `'move'`: Move the existing shape. `'clear'`: Clear the current shape. `'square'`: Makes the shape square. `'center'`: Make the initial point the center of the shape. `'square'` and `'center'` can be combined.

`center`

Center of rectangle

`corners`

Corners of rectangle from lower left, moving clockwise.

`draw_shape(extents)`

`edge_centers`

Midpoint of rectangle edges from left, moving clockwise.

`extents`

Return (xmin, xmax, ymin, ymax).

`geometry`

Returns `numpy.ndarray` of shape (2,5) containing x (`RectangleSelector.geometry[1,:]`) and y (`RectangleSelector.geometry[0,:]`) coordinates of the four corners of the rectangle starting and ending in the top left corner.

```
class matplotlib.widgets.Slider(ax, label, valmin, valmax, valinit=0.5, valfmt='%1.2f',
                                closedmin=True, closedmax=True, slidermin=None, slider-
                                max=None, dragging=True, valstep=None, **kwargs)
```

Bases: `matplotlib.widgets.AxesWidget`

A slider representing a floating point range.

Create a slider from `valmin` to `valmax` in axes `ax`. For the slider to remain responsive you must maintain a reference to it. Call `on_changed()` to connect to the slider event.

Attributes

`val` [float] Slider value.

Parameters

- `ax` [Axes] The Axes to put the slider in.
- `label` [str] Slider label.
- `valmin` [float] The minimum value of the slider.
- `valmax` [float] The maximum value of the slider.
- `valinit` [float, optional, default: 0.5] The slider initial position.
- `valfmt` [str, optional, default: "%1.2f"] Used to format the slider value, fprint format string.
- `closedmin` [bool, optional, default: True] Indicate whether the slider interval is closed on the bottom.
- `closedmax` [bool, optional, default: True] Indicate whether the slider interval is closed on the top.
- `slidermin` [Slider, optional, default: None] Do not allow the current slider to have a value less than the value of the Slider `slidermin`.
- `slidermax` [Slider, optional, default: None] Do not allow the current slider to have a value greater than the value of the Slider `slidermax`.
- `dragging` [bool, optional, default: True] If True the slider can be dragged by the mouse.
- `valstep` [float, optional, default: None] If given, the slider will snap to multiples of `valstep`.

Notes

Additional kwargs are passed on to `self.poly` which is the `Rectangle` that draws the slider knob. See the `Rectangle` documentation for valid property names (e.g., `facecolor`, `edgecolor`, `alpha`).

`disconnect(cid)`

Remove the observer with connection id `cid`

Parameters

- `cid` [int] Connection id of the observer to be removed

`on__changed(func)`

When the slider value is changed call `func` with the new slider value

Parameters

- `func` [callable] Function to call when slider is changed. The function must accept a single float as its arguments.

Returns

- `cid` [int] Connection id (which can be used to disconnect `func`)

`reset()`
Reset the slider to the initial value

`set_val(val)`
Set slider value to `val`

Parameters

`val` [float]

```
class matplotlib.widgets.SpanSelector(ax, onselect, direction, minspan=None, use-
                                     blit=False, rectprops=None, onmove_callback=None,
                                     span_stays=False, button=None)
```

Bases: `matplotlib.widgets._SelectorWidget`

Visually select a min/max range on a single axis and call a function with those values.

To guarantee that the selector remains responsive, keep a reference to it.

In order to turn off the `SpanSelector`, set `span_selector.active=False`. To turn it back on, set `span_selector.active=True`.

Parameters

`ax` [`matplotlib.axes.Axes` object]

`onselect` [`func(min, max)`, min/max are floats]

`direction` ["horizontal" or "vertical"] The axis along which to draw the span selector

`minspan` [float, default is None] If selection is less than `minspan`, do not call `onselect`

`useblit` [bool, default is False] If True, use the backend-dependent blitting features for faster canvas updates.

`rectprops` [dict, default is None] Dictionary of `matplotlib.patches.Patch` properties

`onmove_callback` [`func(min, max)`, min/max are floats, default is None]
Called on mouse move while the span is being selected

`span_stays` [bool, default is False] If True, the span stays visible after the mouse is released

`button` [int or list of ints]

Determines which mouse buttons activate the span selector 1 = left mouse button

2 = center mouse button (scroll wheel)

3 = right mouse button

Examples

```
>>> import matplotlib.pyplot as plt
>>> import matplotlib.widgets as mwidgets
>>> fig, ax = plt.subplots()
>>> ax.plot([1, 2, 3], [10, 50, 100])
>>> def onselect(vmin, vmax):
>>>     print(vmin, vmax)
>>> rectprops = dict(facecolor='blue', alpha=0.5)
>>> span = mwidgets.SpanSelector(ax, onselect, 'horizontal',
>>>                               rectprops=rectprops)
>>> fig.show()
```

See also: [/gallery/widgets/span_selector](#)

`ignore(event)`
return True if event should be ignored

`new_axes(ax)`
Set `SpanSelector` to operate on a new `Axes`

`class matplotlib.widgets.SubplotTool(targetfig, toolfig)`
Bases: `matplotlib.widgets.Widget`

A tool to adjust the subplot params of a `matplotlib.figure.Figure`.

`targetfig` The figure instance to adjust.

`toolfig` The figure instance to embed the subplot tool into. If `None`, a default figure will be created. If you are using this from the GUI

`funcbottom(val)`

`funcspace(val)`

`funcleft(val)`

`funcright(val)`

`functop(val)`

`funcwspace(val)`

`class matplotlib.widgets.TextBox(ax, label, initial="", color='.95', hovercolor='1', label_pad=0.01)`

Bases: `matplotlib.widgets.AxesWidget`

A GUI neutral text input box.

For the text box to remain responsive you must keep a reference to it.

The following attributes are accessible:

`ax` The matplotlib.axes.Axes the button renders into.

`label` A matplotlib.text.Text instance.

`color` The color of the text box when not hovering.

`hovercolor` The color of the text box when hovering.

Call `on_text_change()` to be updated whenever the text changes.

Call `on_submit()` to be updated whenever the user hits enter or leaves the text entry field.

Parameters

`ax` [matplotlib.axes.Axes] The matplotlib.axes.Axes instance the button will be placed into.

`label` [str] Label for this text box. Accepts string.

`initial` [str] Initial value in the text box

`color` [color] The color of the box

`hovercolor` [color] The color of the box when the mouse is over it

`label_pad` [float] the distance between the label and the right side of the textbox

`begin_typing(x)`

`disconnect(cid)`

remove the observer with connection id cid

`on_submit(func)`

When the user hits enter or leaves the submission box, call this func with event.

A connection id is returned which can be used to disconnect.

`on_text_change(func)`

When the text changes, call this func with event.

A connection id is returned which can be used to disconnect.

`position_cursor(x)`

`set_val(val)`

`stop_typing()`

```
class matplotlib.widgets.ToolHandles(ax, x, y, marker='o', marker_props=None, use-
                                     blit=True)
```

Bases: `object`

Control handles for canvas tools.

Parameters

`ax` [matplotlib.axes.Axes] Matplotlib axes where tool handles are displayed.

`x, y` [1D arrays] Coordinates of control handles.

`marker` [str] Shape of marker used to display handle. See `matplotlib.pyplot.plot`.

`marker_props` [dict] Additional marker properties. See `matplotlib.lines.Line2D`.

`closest(x, y)`

Return index and pixel distance to closest index.

`set__animated(val)`

`set__data(pts, y=None)`

Set x and y positions of handles

`set__visible(val)`

`x`

`y`

class `matplotlib.widgets.Widget`

Bases: `object`

Abstract base class for GUI neutral widgets

`active`

Is the widget active?

`drawon = True`

`eventson = True`

`get__active()`

Get whether the widget is active.

`ignore(event)`

Return True if event should be ignored.

This method (or a version of it) should be called at the beginning of any event callback.

`set__active(active)`

Set whether the widget is active.

pyplot

matplotlib.pyplot is a state-based interface to
matplotlib.

MATPLOTLIB.PYPLOT

matplotlib.pyplot is a state-based interface to matplotlib. It provides a MATLAB-like way of plotting.

pyplot is mainly intended for interactive plots and simple cases of programmatic plot generation:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 5, 0.1)
y = np.sin(x)
plt.plot(x, y)
```

The object-oriented API is recommended for more complex plots.

75.1 Functions

<code>acorr(x[, hold, data])</code>	Plot the autocorrelation of x.
<code>angle_spectrum(x[, Fs, Fc, window, pad_to, ...])</code>	Plot the angle spectrum.
<code>annotate(*args, **kwargs)</code>	Annotate the point xy with text s.
<code>arrow(x, y, dx, dy[, hold])</code>	Add an arrow to the axes.
<code>autoscale([enable, axis, tight])</code>	Autoscale the axis view to the data (toggle).
<code>autumn()</code>	Set the colormap to "autumn".
<code>axes([arg])</code>	Add an axes to the current figure and make it the current axes.
<code>axhline([y, xmin, xmax, hold])</code>	Add a horizontal line across the axis.
<code>axhspan(ymin, ymax[, xmin, xmax, hold])</code>	Add a horizontal span (rectangle) across the axis.
<code>axis(*v, **kwargs)</code>	Convenience method to get or set axis properties.
<code>axvline([x, ymin, ymax, hold])</code>	Add a vertical line across the axes.
<code>axvspan(xmin, xmax[, ymin, ymax, hold])</code>	Add a vertical span (rectangle) across the axes.
<code>bar(*args, **kwargs)</code>	Make a bar plot.

Continued on next page

Table 1 – continued from previous page

<code>barbs(*args, **kw)</code>	Plot a 2-D field of barbs.
<code>barh(*args, **kwargs)</code>	Make a horizontal bar plot.
<code>bone()</code>	Set the colormap to "bone".
<code>box([on])</code>	Turn the axes box on or off on the current axes.
<code>boxplot(x[, notch, sym, vert, whis, ...])</code>	Make a box and whisker plot.
<code>broken_barh(xranges, yrange[, hold, data])</code>	Plot a horizontal sequence of rectangles.
<code>cla()</code>	Clear the current axes.
<code>clabel(CS, *args, **kwargs)</code>	Label a contour plot.
<code>clf()</code>	Clear the current figure.
<code>clim([vmin, vmax])</code>	Set the color limits of the current image.
<code>close(*args)</code>	Close a figure window.
<code>cohere(x, y[, NFFT, Fs, Fc, detrend, ...])</code>	Plot the coherence between x and y.
<code>colorbar([mappable, cax, ax])</code>	Add a colorbar to a plot.
<code>colors()</code>	Deprecated since version 2.1.
<code>connect(s, func)</code>	Connect event with string s to func.
<code>contour(*args, **kwargs)</code>	Plot contours.
<code>contourf(*args, **kwargs)</code>	Plot contours.
<code>cool()</code>	Set the colormap to "cool".
<code>copper()</code>	Set the colormap to "copper".
<code>csd(x, y[, NFFT, Fs, Fc, detrend, window, ...])</code>	Plot the cross-spectral density.
<code>delaxes([ax])</code>	Remove the given Axes ax from the current figure.
<code>disconnect(cid)</code>	Disconnect callback id cid
<code>draw()</code>	Redraw the current figure.
<code>errorbar(x, y[, yerr, xerr, fmt, ecolor, ...])</code>	Plot y versus x as lines and/or markers with attached errorbars.
<code>eventplot(positions[, orientation, ...])</code>	Plot identical parallel lines at the given positions.
<code>figimage(*args, **kwargs)</code>	Add a non-resampled image to the figure.
<code>figlegend(*args, **kwargs)</code>	Place a legend in the figure.
<code>fignum_exists(num)</code>	
<code>figtext(*args, **kwargs)</code>	Add text to figure.
<code>figure([num, figsize, dpi, facecolor, ...])</code>	Creates a new figure.
<code>fill(*args, **kwargs)</code>	Plot filled polygons.
<code>fill_between(x, y1[, y2, where, ...])</code>	Fill the area between two horizontal curves.
<code>fill_betweenx(y, x1[, x2, where, step, ...])</code>	Fill the area between two vertical curves.
<code>findobj([o, match, include_self])</code>	Find artist objects.
<code>flag()</code>	Set the colormap to "flag".
<code>gca(**kwargs)</code>	Get the current Axes instance on the current figure matching the given keyword args, or create one.
<code>gcf()</code>	Get a reference to the current figure.
<code>gci()</code>	Get the current colorable artist.

Continued on next page

Table 1 – continued from previous page

<code>get_current_fig_manager()</code>	
<code>get_figlabels()</code>	Return a list of existing figure labels.
<code>get_fignums()</code>	Return a list of existing figure numbers.
<code>get_plot_commands()</code>	Get a sorted list of all of the plotting commands.
<code>ginput(*args, **kwargs)</code>	Blocking call to interact with a figure.
<code>gray()</code>	Set the colormap to "gray".
<code>grid([b, which, axis])</code>	Turn the axes grids on or off.
<code>hexbin(x, y[, C, gridsize, bins, xscale, ...])</code>	Make a hexagonal binning plot.
<code>hist(x[, bins, range, density, weights, ...])</code>	Plot a histogram.
<code>hist2d(x, y[, bins, range, normed, weights, ...])</code>	Make a 2D histogram plot.
<code>hlines(y, xmin, xmax[, colors, linestyles, ...])</code>	Plot horizontal lines at each y from xmin to xmax.
<code>hold([b])</code>	Deprecated since version 2.0.
<code>hot()</code>	Set the colormap to "hot".
<code>hsv()</code>	Set the colormap to "hsv".
<code>imread(*args, **kwargs)</code>	Read an image from a file into an array.
<code>imsave(*args, **kwargs)</code>	Save an array as in image file.
<code>imshow(X[, cmap, norm, aspect, ...])</code>	Display an image on the axes.
<code>inferno()</code>	Set the colormap to "inferno".
<code>install_repl_displayhook()</code>	Install a repl display hook so that any stale figure are automatically redrawn when control is returned to the repl.
<code>ioff()</code>	Turn interactive mode off.
<code>ion()</code>	Turn interactive mode on.
<code>ishold()</code>	Deprecated since version 2.0.
<code>isinteractive()</code>	Return status of interactive mode.
<code>jet()</code>	Set the colormap to "jet".
<code>legend(*args, **kwargs)</code>	Places a legend on the axes.
<code>locator_params([axis, tight])</code>	Control behavior of tick locators.
<code>loglog(*args, **kwargs)</code>	Make a plot with log scaling on both the x and y axis.
<code>magma()</code>	Set the colormap to "magma".
<code>magnitude_spectrum(x[, Fs, Fc, window, ...])</code>	Plot the magnitude spectrum.
<code>margins(*args, **kw)</code>	Set or retrieve autoscaling margins.
<code>matshow(A[, fignum])</code>	Display an array as a matrix in a new figure window.
<code>minorticks_off()</code>	Remove minor ticks from the current plot.
<code>minorticks_on()</code>	Display minor ticks on the current plot.
<code>nipy_spectral()</code>	Set the colormap to "nipy_spectral".
<code>over(func, *args, **kwargs)</code>	Deprecated since version 2.0.
<code>pause(interval)</code>	Pause for interval seconds.

Continued on next page

Table 1 – continued from previous page

<code>pcolor(*args, **kwargs)</code>	Create a pseudocolor plot with a non-regular rectangular grid.
<code>pcolormesh(*args, **kwargs)</code>	Create a pseudocolor plot with a non-regular rectangular grid.
<code>phase_spectrum(x[, Fs, Fc, window, pad_to, ...])</code>	Plot the phase spectrum.
<code>pie(x[, explode, labels, colors, autopct, ...])</code>	Plot a pie chart.
<code>pink()</code>	Set the colormap to "pink".
<code>plasma()</code>	Set the colormap to "plasma".
<code>plot(*args, **kwargs)</code>	Plot y versus x as lines and/or markers.
<code>plot_date(x, y[, fmt, tz, xdate, ydate, ...])</code>	Plot data that contains dates.
<code>plotfile(fname[, cols, plotfuncs, comments, ...])</code>	Plot the data in a file.
<code>polar(*args, **kwargs)</code>	Make a polar plot.
<code>prism()</code>	Set the colormap to "prism".
<code>psd(x[, NFFT, Fs, Fc, detrend, window, ...])</code>	Plot the power spectral density.
<code>quiver(*args, **kw)</code>	Plot a 2-D field of arrows.
<code>quiverkey(*args, **kw)</code>	Add a key to a quiver plot.
<code>rc(*args, **kwargs)</code>	Set the current rc params.
<code>rc_context([rc, fname])</code>	Return a context manager for managing rc settings.
<code>rcdefaults()</code>	Restore the rc params from Matplotlib's internal defaults.
<code>rgrids(*args, **kwargs)</code>	Get or set the radial gridlines on a polar plot.
<code>savefig(*args, **kwargs)</code>	Save the current figure.
<code>sca(ax)</code>	Set the current Axes instance to ax.
<code>scatter(x, y[, s, c, marker, cmap, norm, ...])</code>	A scatter plot of y vs x with varying marker size and/or color.
<code>sci(im)</code>	Set the current image.
<code>semilogx(*args, **kwargs)</code>	Make a plot with log scaling on the x axis.
<code>semilogy(*args, **kwargs)</code>	Make a plot with log scaling on the y axis.
<code>set_cmap(cmap)</code>	Set the default colormap.
<code>setp(*args, **kwargs)</code>	Set a property on an artist object.
<code>show(*args, **kw)</code>	Display a figure.
<code>specgram(x[, NFFT, Fs, Fc, detrend, window, ...])</code>	Plot a spectrogram.
<code>spectral()</code>	Set the colormap to "spectral".
<code>spring()</code>	Set the colormap to "spring".
<code>spy(Z[, precision, marker, markersize, aspect])</code>	Plot the sparsity pattern on a 2-D array.
<code>stackplot(x, *args, **kwargs)</code>	Draws a stacked area plot.
<code>stem(*args, **kwargs)</code>	Create a stem plot.
<code>step(x, y, *args, **kwargs)</code>	Make a step plot.
<code>streamplot(x, y, u, v[, density, linewidth, ...])</code>	Draws streamlines of a vector flow.
<code>subplot(*args, **kwargs)</code>	Return a subplot axes at the given grid position.

Continued on next page

Table 1 – continued from previous page

<code>subplot2grid(shape, loc[, rowspan, colspan, fig])</code>	Create an axis at specific location inside a regular grid.
<code>subplot_tool([targetfig])</code>	Launch a subplot tool window for a figure.
<code>subplots([nrows, ncols, sharex, sharey, ...])</code>	Create a figure and a set of subplots
<code>subplots_adjust(*args, **kwargs)</code>	Tune the subplot layout.
<code>summer()</code>	Set the colormap to "summer".
<code>suptitle(*args, **kwargs)</code>	Add a centered title to the figure.
<code>switch_backend(newbackend)</code>	Switch the default backend.
<code>table(**kwargs)</code>	Add a table to the current axes.
<code>text(x, y, s[, fontdict, withdash])</code>	Add text to the axes.
<code>thetagrids(*args, **kwargs)</code>	Get or set the theta locations of the gridlines in a polar plot.
<code>tick_params([axis])</code>	Change the appearance of ticks, tick labels, and gridlines.
<code>ticklabel_format(**kwargs)</code>	Change the ScalarFormatter used by default for linear axes.
<code>tight_layout([pad, h_pad, w_pad, rect])</code>	Automatically adjust subplot parameters to give specified padding.
<code>title(s, *args, **kwargs)</code>	Set a title of the current axes.
<code>tricontour(*args, **kwargs)</code>	Draw contours on an unstructured triangular grid.
<code>tricontourf(*args, **kwargs)</code>	Draw contours on an unstructured triangular grid.
<code>tripcolor(*args, **kwargs)</code>	Create a pseudocolor plot of an unstructured triangular grid.
<code>triplot(*args, **kwargs)</code>	Draw a unstructured triangular grid as lines and/or markers.
<code>twinx([ax])</code>	Make a second axes that shares the x-axis.
<code>twiny([ax])</code>	Make a second axes that shares the y-axis.
<code>uninstall_repl_displayhook()</code>	Uninstalls the matplotlib display hook.
<code>violinplot(dataset[, positions, vert, ...])</code>	Make a violin plot.
<code>viridis()</code>	Set the colormap to "viridis".
<code>vlines(x, ymin, ymax[, colors, linestyles, ...])</code>	Plot vertical lines.
<code>waitforbuttonpress(*args, **kwargs)</code>	Blocking call to interact with the figure.
<code>winter()</code>	Set the colormap to "winter".
<code>xcorr(x, y[, normed, detrend, usevlines, ...])</code>	Plot the cross correlation between x and y.
<code>xkcd([scale, length, randomness])</code>	Turns on <code>xkcd</code> sketch-style drawing mode. This will only have effect on things drawn after this function is called..
<code>xlabel(s, *args, **kwargs)</code>	Set the x-axis label of the current axes.
<code>xlim(*args, **kwargs)</code>	Get or set the x limits of the current axes.
<code>xscale(*args, **kwargs)</code>	Set the scaling of the x-axis.
<code>xticks(*args, **kwargs)</code>	Get or set the current tick locations and labels of the x-axis.
<code>ylabel(s, *args, **kwargs)</code>	Set the y-axis label of the current axes.

Continued on next page

Table 1 – continued from previous page

<code>ylim(*args, **kwargs)</code>	Get or set the y-limits of the current axes.
<code>yscale(*args, **kwargs)</code>	Set the scaling of the y-axis.
<code>yticks(*args, **kwargs)</code>	Get or set the current tick locations and labels of the y-axis.

75.1.1 matplotlib.pyplot.acorr

`matplotlib.pyplot.acorr(x, hold=None, data=None, **kwargs)`

Plot the autocorrelation of x.

Parameters

`x` [sequence of scalar]

`hold` [bool, optional, deprecated, default: True]

`detrend` [callable, optional, default: `mlab.detrend_none`] x is detrended by the detrend callable. Default is no normalization.

`normed` [bool, optional, default: True] If True, input vectors are normalised to unit length.

`usevlines` [bool, optional, default: True] If True, `Axes.vlines` is used to plot the vertical lines from the origin to the `acorr`. Otherwise, `Axes.plot` is used.

`maxlags` [integer, optional, default: 10] Number of lags to show. If None, will return all $2 * \text{len}(x) - 1$ lags.

Returns

`lags` [array (length $2 * \text{maxlags} + 1$)] lag vector.

`c` [array (length $2 * \text{maxlags} + 1$)] auto correlation vector.

`line` [LineCollection or Line2D] Artist added to the axes of the correlation.
LineCollection if `usevlines` is True Line2D if `usevlines` is False

`b` [Line2D or None] Horizontal line at 0 if `usevlines` is True None if `usevlines` is False

Other Parameters

`linestyle` [Line2D prop, optional, default: None] Only used if `usevlines` is False.

`marker` [string, optional, default: 'o']

Notes

The cross correlation is performed with `numpy.correlate()` with `mode = 2`.

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: `'x'`.
-

75.1.2 matplotlib.pyplot.angle_spectrum

`matplotlib.pyplot.angle_spectrum(x, Fs=None, Fc=None, window=None, pad_to=None, sides=None, hold=None, data=None, **kwargs)`

Plot the angle spectrum.

Call signature:

```
angle_spectrum(x, Fs=2, Fc=0, window=mlab.window_hanning,
               pad_to=None, sides='default', **kwargs)
```

Compute the angle spectrum (wrapped phase spectrum) of `x`. Data is padded to a length of `pad_to` and the windowing function `window` is applied to the signal.

Parameters

- `x` [1-D array or sequence] Array or sequence containing the data
- `Fs` [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.
- `window` [callable or ndarray] A function or a vector of length `NFFT`. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.
- `sides` [[`'default'` | `'onesided'` | `'twosided'`]] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. `'onesided'` forces the return of a one-sided spectrum, while `'twosided'` forces two-sided.
- `pad_to` [integer] The number of points to which the data segment is padded when performing the FFT. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` equal to the length of the input signal (i.e. no padding).
- `Fc` [integer] The center frequency of `x` (defaults to 0), which offsets the `x` extents of the plot to reflect the frequency range used when a signal is acquired and then filtered and downsampled to baseband.

Returns

- spectrum [1-D array] The values for the angle spectrum in radians (real valued)
- freqs [1-D array] The frequencies corresponding to the elements in spectrum
- line [a Line2D instance] The line created by this function

Other Parameters

****kwargs** : Keyword arguments control the Line2D properties:

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) float array
alpha		float (0.0 transparent through 1.0 opaque)
animated		bool
antialiased or aa		bool
clip_box		a Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
color or c		any matplotlib color
contains		a callable function
dash_capstyle		['butt' 'round' 'projecting']
dash_joinstyle		['miter' 'round' 'bevel']
dashes		sequence of on/off ink in points
drawstyle		['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']
figure		a Figure instance
fillstyle		['full' 'left' 'right' 'bottom' 'top' 'none']
gid		an id string
label		object
linestyle or ls		['solid' 'dashed' 'dashdot' 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']
linewidth or lw		float value in points
marker		A valid marker style
markeredgecolor or mec		any matplotlib color
markeredgewidth or mew		float value in points
markerfacecolor or mfc		any matplotlib color
markerfacecoloralt or mfcalt		any matplotlib color
markersize or ms		float
markevery		[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]
path_effects		AbstractPathEffect
picker		float distance in points or callable pick function fn(artist, event)
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
solid_capstyle		['butt' 'round' 'projecting']
solid_joinstyle		['miter' 'round' 'bevel']

Continued on next page

Table 2 – continued from previous page

	Property	Description
transform		a matplotlib.transforms.Transform instance
url		a url string
visible		bool
xdata		1D array
ydata		1D array
zorder		float

See also:

`magnitude_spectrum()` `angle_spectrum()` plots the magnitudes of the corresponding frequencies.

`phase_spectrum()` `phase_spectrum()` plots the unwrapped version of this function.

`specgram()` `specgram()` can plot the angle spectrum of segments within the signal in a colormap.

Notes

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'x'.
-

75.1.3 matplotlib.pyplot.annotate

`matplotlib.pyplot.annotate(*args, **kwargs)`

Annotate the point `xy` with text `s`.

Additional kwargs are passed to `Text`.

Parameters

`s` [str] The text of the annotation

`xy` [iterable] Length 2 sequence specifying the (x,y) point to annotate

`xytext` [iterable, optional] Length 2 sequence specifying the (x,y) to place the text at. If None, defaults to `xy`.

`xycoords` [str, Artist, Transform, callable or tuple, optional] The coordinate system that `xy` is given in.

For a `str` the allowed values are:

Property	Description
'figure points'	points from the lower left of the figure
'figure pixels'	pixels from the lower left of the figure
'figure fraction'	fraction of figure from lower left
'axes points'	points from lower left corner of axes
'axes pixels'	pixels from lower left corner of axes
'axes fraction'	fraction of axes from lower left
'data'	use the coordinate system of the object being annotated (default)
'polar'	(theta,r) if not native 'data' coordinates

If a Artist object is passed in the units are fraction if it's bounding box.

If a Transform object is passed in use that to transform xy to screen coordinates

If a callable it must take a RendererBase object as input and return a Transform or Bbox object

If a [tuple](#) must be length 2 tuple of str, Artist, Transform or callable objects. The first transform is used for the x coordinate and the second for y.

See [Advanced Annotation](#) for more details.

Defaults to 'data'

`textcoords` [str, Artist, Transform, callable or tuple, optional] The coordinate system that xytext is given, which may be different than the coordinate system used for xy.

All xycoords values are valid as well as the following strings:

Property	Description
'offset points'	offset (in points) from the xy value
'offset pixels'	offset (in pixels) from the xy value

defaults to the input of xycoords

`arrowprops` [dict, optional] If not None, properties used to draw a FancyArrowPatch arrow between xy and xytext.

If arrowprops does not contain the key 'arrowstyle' the allowed keys are:

Key	Description
width	the width of the arrow in points
headwidth	the width of the base of the arrow head in points
headlength	the length of the arrow head in points
shrink	fraction of total length to 'shrink' from both ends
?	any key to matplotlib.patches.FancyArrowPatch

If the arrowprops contains the key 'arrowstyle' the above keys are forbidden.
The allowed values of 'arrowstyle' are:

Name	Attrs
'-'	None
'->'	head_length=0.4,head_width=0.2
'-['	widthB=1.0,lengthB=0.2,angleB=None
' -'	widthA=1.0,widthB=1.0
'-> '	head_length=0.4,head_width=0.2
'<-'	head_length=0.4,head_width=0.2
'<->'	head_length=0.4,head_width=0.2
'< -'	head_length=0.4,head_width=0.2
'< ->'	head_length=0.4,head_width=0.2
'fancy'	head_length=0.4,head_width=0.4,tail_width=0.4
'simple'	head_length=0.5,head_width=0.5,tail_width=0.2
'wedge'	tail_width=0.3,shrink_factor=0.5

Valid keys for FancyArrowPatch are:

Key	Description
arrowstyle	the arrow style
connectionstyle	the connection style
relpos	default is (0.5, 0.5)
patchA	default is bounding box of the text
patchB	default is None
shrinkA	default is 2 points
shrinkB	default is 2 points
mutation_scale	default is text size (in points)
mutation_aspect	default is 1.
?	any key for matplotlib.patches.PathPatch

Defaults to None

annotation_clip [bool, optional] Controls the visibility of the annotation when it goes outside the axes area.

If **True**, the annotation will only be drawn when the xy is inside the axes.
If **False**, the annotation will always be drawn regardless of its position.

The default is **None**, which behave as **True** only if xycoords is "data".

Returns

Annotation

Examples using `matplotlib.pyplot.annotate`

- [sphx_glr_gallery_text_labels_and_annotations_usetex_demo.py](#)
- [sphx_glr_gallery_text_labels_and_annotations_mathtext_examples.py](#)
- [sphx_glr_gallery_showcase_xkcd.py](#)
- [Pyplot tutorial](#)

75.1.4 `matplotlib.pyplot.arrow`

`matplotlib.pyplot.arrow(x, y, dx, dy, hold=None, **kwargs)`

Add an arrow to the axes.

This draws an arrow from (x, y) to $(x+dx, y+dy)$.

Parameters

`x, y` [float] The x/y-coordinate of the arrow base.

`dx, dy` [float] The length of the arrow along x/y-direction.

Returns

`arrow` [FancyArrow] The created FancyArrow object.

Other Parameters

`**kwargs` Optional kwargs (inherited from FancyArrow patch) control the arrow construction and properties:

Constructor arguments

`width`: float (default: 0.001) width of full arrow tail

`length_includes_head`: bool (default: False) True if head is to be counted in calculating the length.

`head_width`: float or None (default: 3*width) total width of the full arrow head

`head_length`: float or None (default: 1.5 * head_width) length of arrow head

`shape`: ['full', 'left', 'right'] (default: 'full') draw the left-half, right-half, or full arrow

`overhang`: float (default: 0) fraction that the arrow is swept back (0 overhang means triangular shape). Can be negative or greater than one.

`head_starts_at_zero`: bool (default: False) if True, the head starts being drawn at coordinate 0 instead of ending at coordinate 0.

Other valid kwargs (inherited from :class:'Patch') are:

Property	Description
<code>agg_filter</code>	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
<code>alpha</code>	float or None
<code>animated</code>	bool
<code>antialiased</code> or <code>aa</code>	bool or None
<code>capstyle</code>	['butt' 'round' 'projecting']
<code>clip_box</code>	a Bbox instance
<code>clip_on</code>	bool
<code>clip_path</code>	[(Path, Transform) Patch None]
<code>color</code>	matplotlib color spec
<code>contains</code>	a callable function
<code>edgecolor</code> or <code>ec</code>	mpl color spec, None, 'none', or 'auto'
<code>facecolor</code> or <code>fc</code>	mpl color spec, or None for default, or 'none' for no color
<code>figure</code>	a Figure instance
<code>fill</code>	bool
<code>gid</code>	an id string
<code>hatch</code>	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
<code>joinstyle</code>	['miter' 'round' 'bevel']
<code>label</code>	object
<code>linestyle</code> or <code>ls</code>	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
<code>linewidth</code> or <code>lw</code>	float or None for default
<code>path_effects</code>	AbstractPathEffect
<code>picker</code>	[None bool float callable]
<code>rasterized</code>	bool or None
<code>sketch_params</code>	(scale: float, length: float, randomness: float)
<code>snap</code>	bool or None
<code>transform</code>	Transform
<code>url</code>	a url string
<code>visible</code>	bool
<code>zorder</code>	float

Notes

The resulting arrow is affected by the axes aspect ratio and limits. This may produce an arrow whose head is not square with its stem. To create an arrow whose head is square with its stem, use `annotate()` for example:

```
>>> ax.annotate("", xy=(0.5, 0.5), xytext=(0, 0),  
...             arrowprops=dict(arrowstyle="->"))
```

Examples using `matplotlib.pyplot.arrow`

- `sphinx_glr_gallery_text_labels_and_annotations_arrow_demo.py`

75.1.5 `matplotlib.pyplot.autoscale`

`matplotlib.pyplot.autoscale(enable=True, axis='both', tight=None)`

Autoscale the axis view to the data (toggle).

Convenience method for simple axis view autoscaling. It turns autoscaling on or off, and then, if autoscaling for either axis is on, it performs the autoscaling on the specified axis or axes.

Parameters

`enable` [bool or None, optional] True (default) turns autoscaling on, False turns it off. None leaves the autoscaling state unchanged.

`axis` [['both' | 'x' | 'y'], optional] which axis to operate on; default is 'both'

`tight`: bool or None, optional If True, set view limits to data limits; if False, let the locator and margins expand the view limits; if None, use tight scaling if the only artist is an image, otherwise treat tight as False. The tight setting is retained for future autoscaling until it is explicitly changed.

75.1.6 `matplotlib.pyplot.autumn`

`matplotlib.pyplot.autumn()`

Set the colormap to "autumn".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.7 `matplotlib.pyplot.axes`

`matplotlib.pyplot.axes(arg=None, **kwargs)`

Add an axes to the current figure and make it the current axes.

Parameters

arg [None or 4-tuple or Axes] The exact behavior of this function depends on the type:

- None: A new full window axes is added using `subplot(111, **kwargs)`
- 4-tuple of floats `rect = [left, bottom, width, height]`. A new axes is added with dimensions `rect` in normalized (0, 1) units using `add_axes` on the current figure.
- Axes: This is equivalent to `pyplot.sca`. It sets the current axes to `arg`. Note: This implicitly changes the current figure to the parent of `arg`.

Note: The use of an Axes as an argument is deprecated and will be removed in v3.0. Please use `pyplot.sca` instead.

Returns

axes [Axes] The created or activated axes.

Other Parameters

kwargs : For allowed keyword arguments see `pyplot.subplot` and `Figure.add_axes` respectively. Some common keyword arguments are listed below:

kwarg	Ac-cepts	Description
face-color	color	the axes background color
frameon	bool	whether to display the frame
sharex	otherax	share x-axis with otherax
sharey	otherax	share y-axis with otherax
polar	bool	whether to use polar axes
aspect	[str num]	['equal', 'auto'] or a number. If a number, the ratio of y-unit/x-unit in screen-space. See also <code>set_aspect</code> .

Examples

Creating a new full window axes:

```
>>> plt.axes()
```

Creating a new axes with specified dimensions and some kwargs:

```
>>> plt.axes((left, bottom, width, height), facecolor='w')
```

Examples using `matplotlib.pyplot.axes`

- `sphinx_glr_gallery_subplots_axes_and_figures_subplots_adjust.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_axes_demo.py`
- `sphinx_glr_gallery_lines_bars_and_markers_scatter_hist.py`
- `sphinx_glr_gallery_text_labels_and_annotations_arrow_simple_demo.py`
- `sphinx_glr_gallery_text_labels_and_annotations_mathtext_examples.py`
- `sphinx_glr_gallery_axes_grid1_demo_axes_hbox_divider.py`
- `sphinx_glr_gallery_axes_grid1_make_room_for_ylabel_using_axesgrid.py`
- `sphinx_glr_gallery_event_handling_lasso_demo.py`
- `sphinx_glr_gallery_misc_tight_bbox_test.py`
- `sphinx_glr_gallery_widgets_textbox.py`
- `sphinx_glr_gallery_widgets_check_buttons.py`
- `sphinx_glr_gallery_widgets_buttons.py`
- `sphinx_glr_gallery_widgets_radio_buttons.py`
- `sphinx_glr_gallery_widgets_slider_demo.py`

75.1.8 `matplotlib.pyplot.axhline`

`matplotlib.pyplot.axhline(y=0, xmin=0, xmax=1, hold=None, **kwargs)`

Add a horizontal line across the axis.

Parameters

`y` [scalar, optional, default: 0] y position in data coordinates of the horizontal line.

`xmin` [scalar, optional, default: 0] Should be between 0 and 1, 0 being the far left of the plot, 1 the far right of the plot.

`xmax` [scalar, optional, default: 1] Should be between 0 and 1, 0 being the far left of the plot, 1 the far right of the plot.

Returns

`line` [Line2D]

Other Parameters

`**kwargs` : Valid kwargs are Line2D properties, with the exception of 'transform':

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) float (0.0 transparent through 1.0 opaque)
alpha		float (0.0 transparent through 1.0 opaque)
animated		bool
antialiased or aa		bool
clip_box		a Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
color or c		any matplotlib color
contains		a callable function
dash_capstyle		['butt' 'round' 'projecting']
dash_joinstyle		['miter' 'round' 'bevel']
dashes		sequence of on/off ink in points
drawstyle		['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']
figure		a Figure instance
fillstyle		['full' 'left' 'right' 'bottom' 'top' 'none']
gid		an id string
label		object
linestyle or ls		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']
linewidth or lw		float value in points
marker		A valid marker style
markeredgecolor or mec		any matplotlib color
markeredgewidth or mew		float value in points
markerfacecolor or mfc		any matplotlib color
markerfacecoloralt or mfcalt		any matplotlib color
markersize or ms		float
markevery		[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]
path_effects		AbstractPathEffect
picker		float distance in points or callable pick function fn(artist, event)
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
solid_capstyle		['butt' 'round' 'projecting']
solid_joinstyle		['miter' 'round' 'bevel']
transform		a matplotlib.transforms.Transform instance
url		a url string
visible		bool
xdata		1D array
ydata		1D array
zorder		float

See also:

`hlines` Add horizontal lines in data coordinates.

`axhspan` Add a horizontal span (rectangle) across the axis.

Examples

- draw a thick red hline at 'y' = 0 that spans the xrange:

```
>>> axhline(linewidth=4, color='r')
```

- draw a default hline at 'y' = 1 that spans the xrange:

```
>>> axhline(y=1)
```

- draw a default hline at 'y' = .5 that spans the middle half of the xrange:

```
>>> axhline(y=.5, xmin=0.25, xmax=0.75)
```

Examples using matplotlib.pyplot.axhline

- sphx_glr_gallery_subplots_axes_and_figures_axhspan_demo.py
- sphx_glr_gallery_color_color_cycle_default.py
- sphx_glr_gallery_text_labels_and_annotations_multiline.py
- sphx_glr_gallery_misc_zorder_demo.py

75.1.9 matplotlib.pyplot.axhspan

matplotlib.pyplot.axhspan(ymin, ymax, xmin=0, xmax=1, hold=None, **kwargs)

Add a horizontal span (rectangle) across the axis.

Draw a horizontal span (rectangle) from ymin to ymax. With the default values of xmin = 0 and xmax = 1, this always spans the xrange, regardless of the xlim settings, even if you change them, e.g., with the set_xlim() command. That is, the horizontal extent is in axes coords: 0=left, 0.5=middle, 1.0=right but the y location is in data coordinates.

Parameters

ymin [float] Lower limit of the horizontal span in data units.

ymax [float] Upper limit of the horizontal span in data units.

xmin [float, optional, default: 0] Lower limit of the vertical span in axes (relative 0-1) units.

xmax [float, optional, default: 1] Upper limit of the vertical span in axes (relative 0-1) units.

Returns

Polygon [Polygon]

Other Parameters

****kwargs** [Polygon properties.]

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
an- tialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '.' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

See also:

`axvspan` Add a vertical span across the axes.

Examples using `matplotlib.pyplot.ahspan`

- `sphx_glr_gallery_subplots_axes_and_figures_ahspan_demo.py`

75.1.10 `matplotlib.pyplot.axis`

`matplotlib.pyplot.axis(*v, **kwargs)`

Convenience method to get or set axis properties.

Calling with no arguments:

```
>>> axis()
```

returns the current axes limits `[xmin, xmax, ymin, ymax]`..

```
>>> axis(v)
```

sets the min and max of the x and y axes, with `v = [xmin, xmax, ymin, ymax]`..

```
>>> axis('off')
```

turns off the axis lines and labels.:

```
>>> axis('equal')
```

changes limits of x or y axis so that equal increments of x and y have the same length; a circle is circular.:

```
>>> axis('scaled')
```

achieves the same result by changing the dimensions of the plot box instead of the axis data limits.:

```
>>> axis('tight')
```

changes x and y axis limits such that all data is shown. If all data is already shown, it will move it to the center of the figure without modifying `(xmax - xmin)` or `(ymax - ymin)`. Note this is slightly different than in MATLAB.:

```
>>> axis('image')
```

is 'scaled' with the axis limits equal to the data limits.:

```
>>> axis('auto')
```

and:

```
>>> axis('normal')
```

are deprecated. They restore default behavior; axis limits are automatically scaled to make the data fit comfortably within the plot box.

if `len(*v)==0`, you can pass in `xmin`, `xmax`, `ymin`, `ymax` as kwargs selectively to alter just those limits without changing the others.

```
>>> axis('square')
```

changes the limit ranges (`xmax-xmin`) and (`ymax-ymin`) of the x and y axes to be the same, and have the same scaling, resulting in a square plot.

The `xmin`, `xmax`, `ymin`, `ymax` tuple is returned

See also:

`xlim()`, `ylim()` For setting the x- and y-limits individually.

Examples using `matplotlib.pyplot.axis`

- `sphinx_glr_gallery_pyplots_pyplot_formatstr.py`
- `sphinx_glr_gallery_pyplots_pyplot_text.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_axhspan_demo.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_axes_demo.py`
- `sphinx_glr_gallery_shapes_and_collections_artist_reference.py`
- `sphinx_glr_gallery_text_labels_and_annotations_autowrap.py`
- `sphinx_glr_gallery_text_labels_and_annotations_stix_fonts_demo.py`
- `sphinx_glr_gallery_text_labels_and_annotations_font_table_ttf_sgskip.py`
- `sphinx_glr_gallery_text_labels_and_annotations_fonts_demo_kw.py`
- `sphinx_glr_gallery_text_labels_and_annotations_text_alignment.py`
- `sphinx_glr_gallery_text_labels_and_annotations_fonts_demo.py`
- `sphinx_glr_gallery_event_handling_ginput_manual_clabel_sgskip.py`
- `sphinx_glr_gallery_misc_contour_manual.py`
- `sphinx_glr_gallery_misc_cursor_demo_sgskip.py`
- `sphinx_glr_gallery_specialty_plots_anscombe.py`
- `sphinx_glr_gallery_widgets_slider_demo.py`
- [Pyplot tutorial](#)

75.1.11 matplotlib.pyplot.axvline

matplotlib.pyplot.axvline(x=0, ymin=0, ymax=1, hold=None, **kwargs)

Add a vertical line across the axes.

Parameters

- x [scalar, optional, default: 0] x position in data coordinates of the vertical line.
- ymin [scalar, optional, default: 0] Should be between 0 and 1, 0 being the bottom of the plot, 1 the top of the plot.
- ymax [scalar, optional, default: 1] Should be between 0 and 1, 0 being the bottom of the plot, 1 the top of the plot.

Returns

line [Line2D]

Other Parameters

****kwargs** : Valid kwargs are Line2D properties, with the exception of 'transform':

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)
alpha		float (0.0 transparent through 1.0 opaque)
animated		bool
antialiased or aa		bool
clip_box		a Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
color or c		any matplotlib color
contains		a callable function
dash_capstyle		['butt' 'round' 'projecting']
dash_joinstyle		['miter' 'round' 'bevel']
dashes		sequence of on/off ink in points
drawstyle		['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']
figure		a Figure instance
fillstyle		['full' 'left' 'right' 'bottom' 'top' 'none']
gid		an id string
label		object
linestyle or ls		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' ''
linewidth or lw		float value in points
marker		A valid marker style
markeredgecolor or mec		any matplotlib color
markeredgewidth or mew		float value in points
markerfacecolor or mfc		any matplotlib color
markerfacecoloralt or mfcalt		any matplotlib color

Continued on next page

Table 4 – continued from previous page

	Property	Description
markersize or ms	float	
markevery	[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]	
path_effects	AbstractPathEffect	
picker	float distance in points or callable pick function fn(artist, event)	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
solid_capstyle	['butt' 'round' 'projecting']	
solid_joinstyle	['miter' 'round' 'bevel']	
transform	a matplotlib.transforms.Transform instance	
url	a url string	
visible	bool	
xdata	1D array	
ydata	1D array	
zorder	float	

See also:

`vlines` Add vertical lines in data coordinates.

`axvspan` Add a vertical span (rectangle) across the axis.

Examples

- draw a thick red vline at $x = 0$ that spans the yrange:

```
>>> axvline(linewidth=4, color='r')
```

- draw a default vline at $x = 1$ that spans the yrange:

```
>>> axvline(x=1)
```

- draw a default vline at $x = .5$ that spans the middle half of the yrange:

```
>>> axvline(x=.5, ymin=0.25, ymax=0.75)
```

Examples using `matplotlib.pyplot.axvline`

- `sphx_glr_gallery_subplots_axes_and_figures_axhspan_demo.py`
- `sphx_glr_gallery_color_color_cycle_default.py`

75.1.12 matplotlib.pyplot.axvspan

`matplotlib.pyplot.axvspan(xmin, xmax, ymin=0, ymax=1, hold=None, **kwargs)`

Add a vertical span (rectangle) across the axes.

Draw a vertical span (rectangle) from `xmin` to `xmax`. With the default values of `ymin = 0` and `ymax = 1`. This always spans the yrange, regardless of the `ylim` settings, even if you change them, e.g., with the `set_ylim()` command. That is, the vertical extent is in axes coords: 0=bottom, 0.5=middle, 1.0=top but the `y` location is in data coordinates.

Parameters

`xmin` [scalar] Number indicating the first X-axis coordinate of the vertical span rectangle in data units.

`xmax` [scalar] Number indicating the second X-axis coordinate of the vertical span rectangle in data units.

`ymin` [scalar, optional] Number indicating the first Y-axis coordinate of the vertical span rectangle in relative Y-axis units (0-1). Default to 0.

`ymax` [scalar, optional] Number indicating the second Y-axis coordinate of the vertical span rectangle in relative Y-axis units (0-1). Default to 1.

Returns

`rectangle` [matplotlib.patches.Polygon] Vertical span (rectangle) from (`xmin`, `ymin`) to (`xmax`, `ymax`).

Other Parameters

`**kwargs` Optional parameters are properties of the class `matplotlib.patches.Polygon`.

See also:

`axhspan` Add a horizontal span across the axes.

Examples

Draw a vertical, green, translucent rectangle from `x = 1.25` to `x = 1.55` that spans the yrange of the axes.

```
>>> axvspan(1.25, 1.55, facecolor='g', alpha=0.5)
```

Examples using `matplotlib.pyplot.axvspan`

- `sphx_glr_gallery_subplots_axes_and_figures_axhspan_demo.py`

75.1.13 matplotlib.pyplot.bar

matplotlib.pyplot.bar(*args, **kwargs)

Make a bar plot.

Call signatures:

```
bar(x, height, *, align='center', **kwargs)
bar(x, height, width, *, align='center', **kwargs)
bar(x, height, width, bottom, *, align='center', **kwargs)
```

The bars are positioned at *x* with the given alignment. Their dimensions are given by *width* and *height*. The vertical baseline is *bottom* (default 0).

Each of *x*, *height*, *width*, and *bottom* may either be a scalar applying to all bars, or it may be a sequence of length *N* providing a separate value for each bar.

Parameters

x [sequence of scalars] The *x* coordinates of the bars. See also *align* for the alignment of the bars to the coordinates.

height [scalar or sequence of scalars] The height(s) of the bars.

width [scalar or array-like, optional] The width(s) of the bars (default: 0.8).

bottom [scalar or array-like, optional] The *y* coordinate(s) of the bars bases (default: 0).

align [{'center', 'edge'}], optional, default: 'center' Alignment of the bars to the *x* coordinates:

- 'center': Center the base on the *x* positions.
- 'edge': Align the left edges of the bars with the *x* positions.

To align the bars on the right edge pass a negative width and *align*='edge'.

Returns

container [BarContainer] Container with all the bars and optionally errorbars.

Other Parameters

color [scalar or array-like, optional] The colors of the bar faces.

edgecolor [scalar or array-like, optional] The colors of the bar edges.

linewidth [scalar or array-like, optional] Width of the bar edge(s). If 0, don't draw edges.

tick_label [string or array-like, optional] The tick labels of the bars. Default: None (Use default numeric labels.)

xerr, *yerr* [scalar or array-like of shape(*N*,) or shape(2,*N*), optional] If not None, add horizontal / vertical errorbars to the bar tips. The values are +/- sizes relative to the data:

- scalar: symmetric +/- values for all bars
- shape(N,): symmetric +/- values for each bar
- shape(2,N): Separate - and + values for each bar. First row contains the lower errors, the second row contains the upper errors.
- None: No errorbar. (Default)

See /gallery/statistics/errorbar_features for an example on the usage of `xerr` and `yerr`.

`ecolor` [scalar or array-like, optional, default: 'black'] The line color of the errorbars.

`capsize` [scalar, optional] The length of the error bar caps in points. Default: None, which will take the value from `rcParams["errorbar.capsize"]`.

`error_kw` [dict, optional] Dictionary of kwargs to be passed to the errorbar method. Values of `ecolor` or `capsize` defined here take precedence over the independent kwargs.

`log` [bool, optional, default: False] If True, set the y-axis to be log scale.

`orientation` [{'vertical', 'horizontal'}, optional] This is for internal use only. Please use `barh` for horizontal bar plots. Default: 'vertical'.

See also:

`barh` Plot a horizontal bar plot.

Notes

The optional arguments `color`, `edgecolor`, `linewidth`, `xerr`, and `yerr` can be either scalars or sequences of length equal to the number of bars. This enables you to use `bar` as the basis for stacked bar charts, or candlestick plots. Detail: `xerr` and `yerr` are passed directly to `errorbar()`, so they can also have shape `2xN` for independent specification of lower and upper errors.

Other optional kwargs:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float or None
animated	bool
antialiased or aa	bool or None
capstyle	['butt' 'round' 'projecting']
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	matplotlib color spec
contains	a callable function
edgecolor or ec	mpl color spec, None, 'none', or 'auto'
facecolor or fc	mpl color spec, or None for default, or 'none' for no color
figure	a Figure instance
fill	bool
gid	an id string
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '.' '*']
joinstyle	['miter' 'round' 'bevel']
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float or None for default
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All arguments with the following names: 'bottom', 'color', 'ecolor', 'edgecolor', 'height', 'left', 'linewidth', 'tick_label', 'width', 'x', 'xerr', 'y', 'yerr'.
 - All positional arguments.
-

Examples using `matplotlib.pyplot.bar`

- [sphx_glr_gallery_lines_bars_and_markers_bar_stacked.py](#)
- [sphx_glr_gallery_shapes_and_collections_hatch_demo.py](#)
- [sphx_glr_gallery_pie_and_polar_charts_polar_bar.py](#)
- [sphx_glr_gallery_pie_and_polar_charts_nested_pie.py](#)
- [sphx_glr_gallery_misc_table_demo.py](#)
- [sphx_glr_gallery_specialty_plots_system_monitor.py](#)
- [sphx_glr_gallery_ticks_and_spines_custom_ticker1.py](#)
- [Pyplot tutorial](#)

75.1.14 `matplotlib.pyplot.barbs`

`matplotlib.pyplot.barbs(*args, **kw)`

Plot a 2-D field of barbs.

Call signatures:

```
barb(U, V, **kw)
barb(U, V, C, **kw)
barb(X, Y, U, V, **kw)
barb(X, Y, U, V, C, **kw)
```

Arguments:

X, Y: The x and y coordinates of the barb locations (default is head of barb; see `pivot` kwarg)

U, V: Give the x and y components of the barb shaft

C: An optional array used to map colors to the barbs

All arguments may be 1-D or 2-D arrays or sequences. If X and Y are absent, they will be generated as a uniform grid. If U and V are 2-D arrays but X and Y are 1-D, and if `len(X)` and `len(Y)` match the column and row dimensions of U, then X and Y will be expanded with [numpy.meshgrid\(\)](#).

U, V, C may be masked arrays, but masked X, Y are not supported at present.

Keyword arguments:

length: Length of the barb in points; the other parts of the barb are scaled against this. Default is 7.

pivot: ['tip' | 'middle' | float] The part of the arrow that is at the grid point; the arrow rotates about this point, hence the name pivot. Default is 'tip'. Can also be a number, which shifts the start of the barb that many points from the origin.

barbcolor: [color | color sequence] Specifies the color all parts of the barb except any flags. This parameter is analogous to the `edgecolor` parameter for polygons, which can be used instead. However this parameter will override `facecolor`.

flagcolor: [color | color sequence] Specifies the color of any flags on the barb. This parameter is analogous to the `facecolor` parameter for polygons, which can be used instead. However this parameter will override `facecolor`. If this is not set (and `C` has not either) then `flagcolor` will be set to match `barbcolor` so that the barb has a uniform color. If `C` has been set, `flagcolor` has no effect.

sizes: A dictionary of coefficients specifying the ratio of a given feature to the length of the barb. Only those values one wishes to override need to be included. These features include:

- `'spacing'` - space between features (flags, full/half barbs)
- `'height'` - height (distance from shaft to top) of a flag or full barb
- `'width'` - width of a flag, twice the width of a full barb
- `'emptybarb'` - radius of the circle used for low magnitudes

fill_empty: A flag on whether the empty barbs (circles) that are drawn should be filled with the flag color. If they are not filled, they will be drawn such that no color is applied to the center. Default is `False`

rounding: A flag to indicate whether the vector magnitude should be rounded when allocating barb components. If `True`, the magnitude is rounded to the nearest multiple of the half-barb increment. If `False`, the magnitude is simply truncated to the next lowest multiple. Default is `True`

barb_increments: A dictionary of increments specifying values to associate with different parts of the barb. Only those values one wishes to override need to be included.

- `'half'` - half barbs (Default is 5)
- `'full'` - full barbs (Default is 10)
- `'flag'` - flags (default is 50)

flip_barb: Either a single boolean flag or an array of booleans. Single boolean indicates whether the lines and flags should point opposite to normal for all barbs. An array (which should be the same size as the other data arrays) indicates whether to flip for each individual barb. Normal behavior is for the barbs and lines to point right (comes from wind barbs having these features point towards low pressure in the Northern Hemisphere.) Default is `False`

Barbs are traditionally used in meteorology as a way to plot the speed and direction of wind observations, but can technically be used to plot any two dimensional vector quantity. As opposed to arrows, which give vector magnitude by the length of the arrow, the barbs give more quantitative information about the vector magnitude by putting slanted lines or a triangle for various increments in magnitude, as show schematically below:



The largest increment is given by a triangle (or "flag"). After those come full lines (barbs). The smallest increment is a half line. There is only, of course, ever at most 1 half line. If the magnitude is small and only needs a single half-line and no full lines or triangles, the half-line is offset from the end of the barb so that it can be easily distinguished from barbs with a single full line. The magnitude for the barb shown above would nominally be 65, using the standard increments of 50, 10, and 5.

linewidths and edgecolors can be used to customize the barb. Additional PolyCollection keyword arguments:

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a
alpha		float or None
animated		bool
antialiased or antialiaseds		Boolean or sequence of booleans
array		ndarray
capstyle		unknown
clim		a length 2 sequence of floats; may be overridden in methods that have vmin and vmax
clip_box		a Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
cmap		a colormap or registered colormap name
color		matplotlib color arg or sequence of rgba tuples
contains		a callable function
edgecolor or edgecolors		matplotlib color spec or sequence of specs
facecolor or facecolors		matplotlib color spec or sequence of specs
figure		a Figure instance
gid		an id string
hatch		['/' '\' '.' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle		unknown
label		object
linestyle or dashes or linestyles		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ...]
linewidth or linewidths or lw		float or sequence of floats
norm		Normalize
offset_position		['screen' 'data']
offsets		float or sequence of floats
path_effects		AbstractPathEffect
picker		[None bool float callable]
pickradius		float distance in points

Continued on next page

Table 5 – continued from previous page

	Property	Description
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
transform		Transform
url		a url string
urls		List[str] or None
visible		bool
zorder		float

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All positional and all keyword arguments.
-

Examples using matplotlib.pyplot.barbs

- sphx_glr_gallery_images_contours_and_fields_barb_demo.py

75.1.15 matplotlib.pyplot.barh

matplotlib.pyplot.barh(*args, **kwargs)

Make a horizontal bar plot.

Call signatures:

```
bar(y, width, *, align='center', **kwargs)
bar(y, width, height, *, align='center', **kwargs)
bar(y, width, height, left, *, align='center', **kwargs)
```

The bars are positioned at y with the given align. Their dimensions are given by width and height. The horizontal baseline is left (default 0).

Each of y, width, height, and left may either be a scalar applying to all bars, or it may be a sequence of length N providing a separate value for each bar.

Parameters

y [scalar or array-like] The y coordinates of the bars. See also align for the alignment of the bars to the coordinates.

width [scalar or array-like] The width(s) of the bars.

height [sequence of scalars, optional, default: 0.8] The heights of the bars.

`left` [sequence of scalars] The x coordinates of the left sides of the bars (default: 0).

`align` [{'center', 'edge'}], optional, default: 'center'] Alignment of the base to the y coordinates*:

- 'center': Center the bars on the y positions.
- 'edge': Align the bottom edges of the bars with the y positions.

To align the bars on the top edge pass a negative height and `align='edge'`.

Returns

`container` [BarContainer] Container with all the bars and optionally errorbars.

Other Parameters

`color` [scalar or array-like, optional] The colors of the bar faces.

`edgecolor` [scalar or array-like, optional] The colors of the bar edges.

`linewidth` [scalar or array-like, optional] Width of the bar edge(s). If 0, don't draw edges.

`tick_label` [string or array-like, optional] The tick labels of the bars. Default: None (Use default numeric labels.)

`xerr`, `yerr` [scalar or array-like of shape(N,) or shape(2,N), optional] If not None, add horizontal / vertical errorbars to the bar tips. The values are +/- sizes relative to the data:

- scalar: symmetric +/- values for all bars
- shape(N,): symmetric +/- values for each bar
- shape(2,N): Separate - and + values for each bar. First row contains the lower errors, the second row contains the upper errors.
- None: No errorbar. (default)

See /gallery/statistics/errorbar_features for an example on the usage of `xerr` and `yerr`.

`ecolor` [scalar or array-like, optional, default: 'black'] The line color of the errorbars.

`capsize` [scalar, optional] The length of the error bar caps in points. Default: None, which will take the value from `rcParams["errorbar.capsize"]`.

`error_kw` [dict, optional] Dictionary of kwargs to be passed to the errorbar method. Values of `ecolor` or `capsize` defined here take precedence over the independent kwargs.

`log` [bool, optional, default: False] If True, set the x-axis to be log scale.

See also:

`bar` Plot a vertical bar plot.

Notes

The optional arguments `color`, `edgecolor`, `linewidth`, `xerr`, and `yerr` can be either scalars or sequences of length equal to the number of bars. This enables you to use `bar` as the basis for stacked bar charts, or candlestick plots. Detail: `xerr` and `yerr` are passed directly to `errorbar()`, so they can also have shape `2xN` for independent specification of lower and upper errors.

Other optional kwargs:

Property	Description
<code>agg_filter</code>	a filter function, which takes a $(m, n, 3)$ float array and a dpi value, and returns a $(m, n, 3)$ array
<code>alpha</code>	float or None
<code>animated</code>	bool
<code>antialiased</code> or <code>aa</code>	bool or None
<code>capstyle</code>	['butt' 'round' 'projecting']
<code>clip_box</code>	a Bbox instance
<code>clip_on</code>	bool
<code>clip_path</code>	[(Path, Transform) Patch None]
<code>color</code>	matplotlib color spec
<code>contains</code>	a callable function
<code>edgecolor</code> or <code>ec</code>	mpl color spec, None, 'none', or 'auto'
<code>facecolor</code> or <code>fc</code>	mpl color spec, or None for default, or 'none' for no color
<code>figure</code>	a Figure instance
<code>fill</code>	bool
<code>gid</code>	an id string
<code>hatch</code>	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '.' '*']
<code>joinstyle</code>	['miter' 'round' 'bevel']
<code>label</code>	object
<code>linestyle</code> or <code>ls</code>	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
<code>linewidth</code> or <code>lw</code>	float or None for default
<code>path_effects</code>	AbstractPathEffect
<code>picker</code>	[None bool float callable]
<code>rasterized</code>	bool or None
<code>sketch_params</code>	(scale: float, length: float, randomness: float)
<code>snap</code>	bool or None
<code>transform</code>	Transform
<code>url</code>	a url string
<code>visible</code>	bool
<code>zorder</code>	float

75.1.16 matplotlib.pyplot.bone

`matplotlib.pyplot.bone()`

Set the colormap to "bone".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.17 matplotlib.pyplot.box

`matplotlib.pyplot.box(on=None)`

Turn the axes box on or off on the current axes.

Parameters

`on` [bool or None] The new Axes box state. If None, toggle the state.

See also:

`matplotlib.axes.Axes.set_frame_on()`, `matplotlib.axes.Axes.get_frame_on()`

75.1.18 matplotlib.pyplot.boxplot

`matplotlib.pyplot.boxplot(x, notch=None, sym=None, vert=None, whis=None, positions=None, widths=None, patch_artist=None, bootstrap=None, usermedians=None, conf_intervals=None, meanline=None, showmeans=None, showcaps=None, showbox=None, showfliers=None, boxprops=None, labels=None, flierprops=None, medianprops=None, meanprops=None, capprops=None, whiskerprops=None, manage_xticks=True, autorange=False, zorder=None, hold=None, data=None)`

Make a box and whisker plot.

Make a box and whisker plot for each column of `x` or each vector in sequence `x`. The box extends from the lower to upper quartile values of the data, with a line at the median. The whiskers extend from the box to show the range of the data. Flier points are those past the end of the whiskers.

Parameters

`x` [Array or a sequence of vectors.] The input data.

`notch` [bool, optional (False)] If `True`, will produce a notched box plot. Otherwise, a rectangular boxplot is produced. The notches represent the confidence interval (CI) around the median. See the entry for the `bootstrap` parameter for information regarding how the locations of the notches are computed.

Note: In cases where the values of the CI are less than the lower quartile or greater than the upper quartile, the notches will extend beyond the box,

giving it a distinctive "flipped" appearance. This is expected behavior and consistent with other statistical visualization packages.

- sym** [str, optional] The default symbol for flier points. Enter an empty string (") if you don't want to show fliers. If `None`, then the fliers default to 'b+' If you want more control use the `flierprops` kwarg.
- vert** [bool, optional (True)] If `True` (default), makes the boxes vertical. If `False`, everything is drawn horizontally.
- whis** [float, sequence, or string (default = 1.5)] As a float, determines the reach of the whiskers to the beyond the first and third quartiles. In other words, where IQR is the interquartile range (Q3-Q1), the upper whisker will extend to last datum less than $Q3 + whis * IQR$. Similarly, the lower whisker will extend to the first datum greater than $Q1 - whis * IQR$. Beyond the whiskers, data are considered outliers and are plotted as individual points. Set this to an unreasonably high value to force the whiskers to show the min and max values. Alternatively, set this to an ascending sequence of percentile (e.g., [5, 95]) to set the whiskers at specific percentiles of the data. Finally, `whis` can be the string 'range' to force the whiskers to the min and max of the data.
- bootstrap** [int, optional] Specifies whether to bootstrap the confidence intervals around the median for notched boxplots. If `bootstrap` is `None`, no bootstrapping is performed, and notches are calculated using a Gaussian-based asymptotic approximation (see McGill, R., Tukey, J.W., and Larsen, W.A., 1978, and Kendall and Stuart, 1967). Otherwise, `bootstrap` specifies the number of times to bootstrap the median to determine its 95% confidence intervals. Values between 1000 and 10000 are recommended.
- usermedians** [array-like, optional] An array or sequence whose first dimension (or length) is compatible with `x`. This overrides the medians computed by matplotlib for each element of `usermedians` that is not `None`. When an element of `usermedians` is `None`, the median will be computed by matplotlib as normal.
- conf_intervals** [array-like, optional] Array or sequence whose first dimension (or length) is compatible with `x` and whose second dimension is 2. When the an element of `conf_intervals` is not `None`, the notch locations computed by matplotlib are overridden (provided `notch` is `True`). When an element of `conf_intervals` is `None`, the notches are computed by the method specified by the other kwargs (e.g., `bootstrap`).
- positions** [array-like, optional] Sets the positions of the boxes. The ticks and limits are automatically set to match the positions. Defaults to `range(1, N+1)` where `N` is the number of boxes to be drawn.
- widths** [scalar or array-like] Sets the width of each box either with a scalar or a sequence. The default is 0.5, or $0.15 * (\text{distance between extreme positions})$, if that is smaller.

`patch_artist` [bool, optional (False)] If `False` produces boxes with the `Line2D` artist. Otherwise, boxes and drawn with `Patch` artists.

`labels` [sequence, optional] Labels for each dataset. Length must be compatible with dimensions of `x`.

`manage_xticks` [bool, optional (True)] If the function should adjust the `xlim` and `xtick` locations.

`autorange` [bool, optional (False)] When `True` and the data are distributed such that the 25th and 75th percentiles are equal, `whis` is set to 'range' such that the whisker ends are at the minimum and maximum of the data.

`meanline` [bool, optional (False)] If `True` (and `showmeans` is `True`), will try to render the mean as a line spanning the full width of the box according to `meanprops` (see below). Not recommended if `shownotches` is also `True`. Otherwise, means will be shown as points.

`zorder` [scalar, optional (None)] Sets the `zorder` of the boxplot.

Returns

`result` [dict] A dictionary mapping each component of the boxplot to a list of the `matplotlib.lines.Line2D` instances created. That dictionary has the following keys (assuming vertical boxplots):

- `boxes`: the main body of the boxplot showing the quartiles and the median's confidence intervals if enabled.
- `medians`: horizontal lines at the median of each box.
- `whiskers`: the vertical lines extending to the most extreme, non-outlier data points.
- `caps`: the horizontal lines at the ends of the whiskers.
- `fliers`: points representing data that extend beyond the whiskers (fliers).
- `means`: points or lines representing the means.

Other Parameters

`showcaps` [bool, optional (True)] Show the caps on the ends of whiskers.

`showbox` [bool, optional (True)] Show the central box.

`showfliers` [bool, optional (True)] Show the outliers beyond the caps.

`showmeans` [bool, optional (False)] Show the arithmetic means.

`capprops` [dict, optional (None)] Specifies the style of the caps.

`boxprops` [dict, optional (None)] Specifies the style of the box.

`whiskerprops` [dict, optional (None)] Specifies the style of the whiskers.

`flierprops` [dict, optional (None)] Specifies the style of the fliers.

`medianprops` [dict, optional (None)] Specifies the style of the median.

meanprops [dict, optional (None)] Specifies the style of the mean.

Notes

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All positional and all keyword arguments.

75.1.19 matplotlib.pyplot.broken_barh

matplotlib.pyplot.broken_barh(xranges, yrange, hold=None, data=None, **kwargs)

Plot a horizontal sequence of rectangles.

A rectangle is drawn for each element of xrange. All rectangles have the same vertical position and size defined by yrange.

This is a convenience function for instantiating a BrokenBarHCollection, adding it to the axes and autoscaling the view.

Parameters

xranges [sequence of tuples (xmin, xwidth)] The x-positions and extends of the rectangles. For each tuple (xmin, xwidth) a rectangle is drawn from xmin to xmin + xwidth.

yrange [(ymin, ymax)] The y-position and extend for all the rectangles.

Returns

collection [A BrokenBarHCollection]

Other Parameters

**kwargs [BrokenBarHCollection properties] Each kwarg can be either a single argument applying to all rectangles, e.g.:

```
facecolors='black'
```

or a sequence of arguments over which is cycled, e.g.:

```
facecolors=('black', 'blue')
```

would create interleaving black and blue rectangles.

Supported keywords:

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a
alpha		float or None
animated		bool
antialiased or antialiaseds		Boolean or sequence of booleans
array		ndarray
capstyle		unknown
clim		a length 2 sequence of floats; may be overridden in methods that have vmin and vm
clip_box		a Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
cmap		a colormap or registered colormap name
color		matplotlib color arg or sequence of rgba tuples
contains		a callable function
edgecolor or edgecolors		matplotlib color spec or sequence of specs
facecolor or facecolors		matplotlib color spec or sequence of specs
figure		a Figure instance
gid		an id string
hatch		['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle		unknown
label		object
linestyle or dashes or linestyles		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ...]
linewidth or linewidths or lw		float or sequence of floats
norm		Normalize
offset_position		['screen' 'data']
offsets		float or sequence of floats
path_effects		AbstractPathEffect
picker		[None bool float callable]
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
transform		Transform
url		a url string
urls		List[str] or None
visible		bool
zorder		float

Notes

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All positional and all keyword arguments.

75.1.20 matplotlib.pyplot.cla

matplotlib.pyplot.cla()

Clear the current axes.

75.1.21 matplotlib.pyplot.clabel

matplotlib.pyplot.clabel(CS, *args, **kwargs)

Label a contour plot.

Call signature:

```
clabel(cs, **kwargs)
```

Adds labels to line contours in cs, where cs is a ContourSet object returned by contour.

```
clabel(cs, v, **kwargs)
```

only labels contours listed in v.

Parameters

fontsize [string or float, optional] Size in points or relative size e.g., 'smaller', 'x-large'. See Text.set_size for accepted string values.

colors : Color of each label

- if None, the color of each label matches the color of the corresponding contour
- if one string color, e.g., colors = 'r' or colors = 'red', all labels will be plotted in this color
- if a tuple of matplotlib color args (string, float, rgb, etc), different labels will be plotted in different colors in the order specified

inline [bool, optional] If True the underlying contour is removed where the label is placed. Default is True.

inline_spacing [float, optional] Space in pixels to leave on each side of label when placing inline. Defaults to 5.

This spacing will be exact for labels at locations where the contour is straight, less so for labels on curved contours.

fmt [string or dict, optional] A format string for the label. Default is '%1.3f'

Alternatively, this can be a dictionary matching contour levels with arbitrary strings to use for each contour level (i.e., `fmt[level]=string`), or it can be any callable, such as a Formatter instance, that returns a string when called with a numeric contour level.

`manual` [bool or iterable, optional] If True, contour labels will be placed manually using mouse clicks. Click the first button near a contour to add a label, click the second button (or potentially both mouse buttons at once) to finish adding labels. The third button can be used to remove the last label added, but only if labels are not inline. Alternatively, the keyboard can be used to select label locations (enter to end label placement, delete or backspace act like the third mouse button, and any other key will select a label location).

`manual` can also be an iterable object of x,y tuples. Contour labels will be created as if mouse is clicked at each x,y positions.

`rightside_up` [bool, optional] If True, label rotations will always be plus or minus 90 degrees from level. Default is True.

`use_clabeltext` [bool, optional] If True, `ClabelText` class (instead of `Text`) is used to create labels. `ClabelText` recalculates rotation angles of texts during the drawing time, therefore this can be used if aspect of the axes changes. Default is False.

Examples using `matplotlib.pyplot.clabel`

- `sphx_glr_gallery_images_contours_and_fields_contour_label_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_contour_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_contourf_demo.py`
- `sphx_glr_gallery_event_handling_ginput_manual_clabel_sgskip.py`

75.1.22 `matplotlib.pyplot.clf`

`matplotlib.pyplot.clf()`

Clear the current figure.

Examples using `matplotlib.pyplot.clf`

- `sphx_glr_gallery_shapes_and_collections_fancybox_demo.py`
- `sphx_glr_gallery_event_handling_ginput_manual_clabel_sgskip.py`

75.1.23 `matplotlib.pyplot.clim`

`matplotlib.pyplot.clim(vmin=None, vmax=None)`

Set the color limits of the current image.

To apply `clim` to all axes images do:

```
clim(0, 0.5)
```

If either vmin or vmax is None, the image min/max respectively will be used for color scaling.

If you want to set the clim of multiple images, use, for example:

```
for im in gca().get_images():
    im.set_clim(0, 0.05)
```

75.1.24 matplotlib.pyplot.close

matplotlib.pyplot.close(*args)

Close a figure window.

close() by itself closes the current figure

close(fig) closes the Figure instance fig

close(num) closes the figure number num

close(name) where name is a string, closes figure with that label

close('all') closes all the figure windows

Examples using matplotlib.pyplot.close

- sphx_glr_gallery_subplots_axes_and_figures_subplots_demo.py
- sphx_glr_gallery_event_handling_pipong.py
- sphx_glr_gallery_misc_multipage_pdf.py
- sphx_glr_gallery_misc_multiprocess_sgskip.py
- [Tight Layout guide](#)

75.1.25 matplotlib.pyplot.cohere

matplotlib.pyplot.cohere(x, y, NFFT=256, Fs=2, Fc=0, detrend=<function detrend_none>, window=<function window_hanning>, noverlap=0, pad_to=None, sides='default', scale_by_freq=None, hold=None, data=None, **kwargs)

Plot the coherence between x and y.

Plot the coherence between x and y. Coherence is the normalized cross spectral density:

$$C_{xy} = \frac{|P_{xy}|^2}{P_{xx}P_{yy}}$$

Parameters

- Fs** [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.
- window** [callable or ndarray] A function or a vector of length `NFFT`. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.
- sides** [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.
- pad_to** [integer] The number of points to which the data segment is padded when performing the FFT. This can be different from `NFFT`, which specifies the number of data points used. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` equal to `NFFT`.
- NFFT** [integer] The number of data points used in each block for the FFT. A power 2 is most efficient. The default value is 256. This should NOT be used to get zero padding, or the scaling of the result will be incorrect. Use `pad_to` for this instead.
- detrend** [{ 'default', 'constant', 'mean', 'linear', 'none' } or callable] The function applied to each segment before `fft`-ing, designed to remove the mean or linear trend. Unlike in MATLAB, where the `detrend` parameter is a vector, in `matplotlib` it is a function. The `pylab` module defines `detrend_none()`, `detrend_mean()`, and `detrend_linear()`, but you can use a custom function as well. You can also use a string to choose one of the functions. 'default', 'constant', and 'mean' call `detrend_mean()`. 'linear' calls `detrend_linear()`. 'none' calls `detrend_none()`.
- scale_by_freq** [boolean, optional] Specifies whether the resulting density values should be scaled by the scaling frequency, which gives density in units of Hz^{-1} . This allows for integration over the returned frequency values. The default is `True` for MATLAB compatibility.
- noverlap** [integer] The number of points of overlap between blocks. The default value is 0 (no overlap).
- Fc** [integer] The center frequency of `x` (defaults to 0), which offsets the `x` extents of the plot to reflect the frequency range used when a signal is acquired and then filtered and downsampled to baseband.

Returns

The return value is a tuple (`*Cxy*`, `*f*`), where `*f*` are the

frequencies of the coherence vector.

kwargs are applied to the lines.

Other Parameters

****kwargs** : Keyword arguments control the Line2D properties:

	Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)	
alpha	float (0.0 transparent through 1.0 opaque)	
animated	bool	
antialiased or aa	bool	
clip_box	a Bbox instance	
clip_on	bool	
clip_path	[(Path, Transform) Patch None]	
color or c	any matplotlib color	
contains	a callable function	
dash_capstyle	['butt' 'round' 'projecting']	
dash_joinstyle	['miter' 'round' 'bevel']	
dashes	sequence of on/off ink in points	
drawstyle	['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']	
figure	a Figure instance	
fillstyle	['full' 'left' 'right' 'bottom' 'top' 'none']	
gid	an id string	
label	object	
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']	
linewidth or lw	float value in points	
marker	A valid marker style	
markeredgecolor or mec	any matplotlib color	
markeredgewidth or mew	float value in points	
markerfacecolor or mfc	any matplotlib color	
markerfacecoloralt or mfcalt	any matplotlib color	
markersize or ms	float	
markevery	[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]	
path_effects	AbstractPathEffect	
picker	float distance in points or callable pick function fn(artist, event)	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
solid_capstyle	['butt' 'round' 'projecting']	
solid_joinstyle	['miter' 'round' 'bevel']	
transform	a matplotlib.transforms.Transform instance	
url	a url string	
visible	bool	
xdata	1D array	

Continued on next page

Table 7 – continued from previous page

	Property	Description
ydata	1D array	
zorder	float	

References

Bendat & Piersol – Random Data: Analysis and Measurement Procedures, John Wiley & Sons (1986)

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All arguments with the following names: 'x', 'y'.

75.1.26 matplotlib.pyplot.colorbar

matplotlib.pyplot.colorbar(mappable=None, cax=None, ax=None, **kw)

Add a colorbar to a plot.

Function signatures for the pyplot interface; all but the first are also method signatures for the colorbar() method:

```
colorbar(**kwargs)
colorbar(mappable, **kwargs)
colorbar(mappable, cax=cax, **kwargs)
colorbar(mappable, ax=ax, **kwargs)
```

Parameters

mappable : The Image, ContourSet, etc. to which the colorbar applies; this argument is mandatory for the Figure colorbar() method but optional for the pyplot colorbar() function, which sets the default to the current image.

cax [Axes object, optional] Axes into which the colorbar will be drawn.

ax [Axes, list of Axes, optional] Parent axes from which space for a new colorbar axes will be stolen. If a list of axes is given they will all be resized to make room for the colorbar axes.

use_gridspec [bool, optional] If cax is None, a new cax is created as an instance of Axes. If ax is an instance of Subplot and use_gridspec is True, cax is created as an instance of Subplot using the grid_spec module.

Returns

:class: '~matplotlib.colorbar.Colorbar' instance. See also its base class, ColorbarBase. Call the `set_label()` method to label the colorbar.

Notes

Additional keyword arguments are of two kinds:

axes properties:

Property	Description
orientation	vertical or horizontal
fraction	0.15; fraction of original axes to use for colorbar
pad	0.05 if vertical, 0.15 if horizontal; fraction of original axes between colorbar and new image axes
shrink	1.0; fraction by which to multiply the size of the colorbar
aspect	20; ratio of long to short dimensions
anchor	(0.0, 0.5) if vertical; (0.5, 1.0) if horizontal; the anchor point of the colorbar axes
panchor	(1.0, 0.5) if vertical; (0.5, 0.0) if horizontal; the anchor point of the colorbar parent axes. If False, the parent axes' anchor will be unchanged

colorbar properties:

Property	Description
extend	['neither' 'both' 'min' 'max'] If not 'neither', make pointed end(s) for out-of- range values. These are set for a given colormap using the colormap set_under and set_over methods.
extendfrac	[None 'auto' length lengths] If set to None, both the minimum and maximum triangular colorbar extensions will have a length of 5% of the interior colorbar length (this is the default setting). If set to 'auto', makes the triangular colorbar extensions the same lengths as the interior boxes (when spacing is set to 'uniform') or the same lengths as the respective adjacent interior boxes (when spacing is set to 'proportional'). If a scalar, indicates the length of both the minimum and maximum triangular colorbar extensions as a fraction of the interior colorbar length. A two-element sequence of fractions may also be given, indicating the lengths of the minimum and maximum colorbar extensions respectively as a fraction of the interior colorbar length.
extendrect	bool If False the minimum and maximum colorbar extensions will be triangular (the default). If True the extensions will be rectangular.
spacing	['uniform' 'proportional'] Uniform spacing gives each discrete color the same space; proportional makes the space proportional to the data interval.
ticks	[None list of ticks Locator object] If None, ticks are determined automatically from the input.
format	[None format string Formatter object] If None, the ScalarFormatter is used. If a format string is given, e.g., '%.3f', that is used. An alternative Formatter object may be given instead.
drawedges	bool Whether to draw lines at color boundaries.

The following will probably be useful only in the context of indexed colors (that is, when the mappable has `norm=NoNorm()`), or other unusual circumstances.

Property	Description
boundaries	None or a sequence
values	None or a sequence which must be of length 1 less than the sequence of boundaries. For each region delimited by adjacent entries in boundaries, the color mapped to the corresponding value in values will be used.

If mappable is a `ContourSet`, its `extend` kwarg is included automatically.

The `shrink` kwarg provides a simple way to scale the colorbar with respect to the axes. Note that if `cax` is specified it determines the size of the colorbar and `shrink` and `aspect` kwargs are

ignored.

For more precise control, you can manually specify the positions of the axes objects in which the mappable and the colorbar are drawn. In this case, do not use any of the axes properties kwargs.

It is known that some vector graphics viewer (svg and pdf) renders white gaps between segments of the colorbar. This is due to bugs in the viewers not matplotlib. As a workaround the colorbar can be rendered with overlapping segments:

```
cbar = colorbar()
cbar.solids.set_edgecolor("face")
draw()
```

However this has negative consequences in other circumstances. Particularly with semi transparent images ($\alpha < 1$) and colorbar extensions and is not enabled by default see (issue #1188).

Examples using matplotlib.pyplot.colorbar

- sphx_glr_gallery_subplots_axes_and_figures_subplots_adjust.py
- sphx_glr_gallery_color_colorbar_basics.py
- sphx_glr_gallery_images_contours_and_fields_contourf_hatching.py
- sphx_glr_gallery_images_contours_and_fields_contourf_log.py
- sphx_glr_gallery_images_contours_and_fields_pcolormesh_levels.py
- sphx_glr_gallery_images_contours_and_fields_multi_image.py
- sphx_glr_gallery_images_contours_and_fields_image_masked.py
- sphx_glr_gallery_images_contours_and_fields_pcolor_demo.py
- sphx_glr_gallery_images_contours_and_fields_contour_demo.py
- sphx_glr_gallery_images_contours_and_fields_contour_image.py
- sphx_glr_gallery_images_contours_and_fields_contourf_demo.py
- sphx_glr_gallery_images_contours_and_fields_custom_cmap.py
- sphx_glr_gallery_images_contours_and_fields_image_annotated_heatmap.py
- sphx_glr_gallery_shapes_and_collections_ellipse_collection.py
- sphx_glr_gallery_shapes_and_collections_line_collection.py
- sphx_glr_gallery_axes_grid1_simple_colorbar.py
- sphx_glr_gallery_axes_grid1_demo_colorbar_with_inset_locator.py
- sphx_glr_gallery_axes_grid1_demo_axes_divider.py
- sphx_glr_gallery_misc_contour_manual.py

- [Image tutorial](#)
- [Tight Layout guide](#)

75.1.27 matplotlib.pyplot.colors

matplotlib.pyplot.colors()

Deprecated since version 2.1: The colors function was deprecated in version 2.1.

This is a do-nothing function to provide you with help on how matplotlib handles colors.

Commands which take color arguments can use several formats to specify the colors. For the basic built-in colors, you can use a single letter

Alias	Color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

For a greater range of colors, you have two options. You can specify the color using an html hex string, as in:

```
color = '#eeefff'
```

or you can pass an R,G,B tuple, where each of R,G,B are in the range [0,1].

You can also use any legal html name for a color, for example:

```
color = 'red'  
color = 'burlywood'  
color = 'chartreuse'
```

The example below creates a subplot with a dark slate gray background:

```
subplot(111, facecolor=(0.1843, 0.3098, 0.3098))
```

Here is an example that creates a pale turquoise title:

```
title('Is this the best color?', color='#afeeee')
```

75.1.28 matplotlib.pyplot.connect

matplotlib.pyplot.connect(s, func)

Connect event with string s to func. The signature of func is:

```
def func(event)
```

where event is a matplotlib.backend_bases.Event. The following events are recognized

- 'button_press_event'
- 'button_release_event'
- 'draw_event'
- 'key_press_event'
- 'key_release_event'
- 'motion_notify_event'
- 'pick_event'
- 'resize_event'
- 'scroll_event'
- 'figure_enter_event',
- 'figure_leave_event',
- 'axes_enter_event',
- 'axes_leave_event'
- 'close_event'

For the location events (button and key press/release), if the mouse is over the axes, the variable event.inaxes will be set to the Axes the event occurs is over, and additionally, the variables event.xdata and event.ydata will be defined. This is the mouse location in data coords. See KeyEvent and MouseEvent for more info.

Return value is a connection id that can be used with mpl_disconnect().

Examples

Usage:

```
def on_press(event):
    print('you pressed', event.button, event.xdata, event.ydata)

cid = canvas.mpl_connect('button_press_event', on_press)
```

Examples using `matplotlib.pyplot.connect`

- `sphx_glr_gallery_event_handling_coords_demo.py`
- `sphx_glr_gallery_misc_cursor_demo_sgskip.py`
- `sphx_glr_gallery_user_interfaces_pylab_with_gtk_sgskip.py`
- `sphx_glr_gallery_widgets_rectangle_selector.py`

75.1.29 `matplotlib.pyplot.contour``matplotlib.pyplot.contour(*args, **kwargs)`

Plot contours.

Call signature:

`contour([X, Y,] Z, [levels], **kwargs)`

`contour()` and `contourf()` draw contour lines and filled contours, respectively. Except as noted, function signatures and return values are the same for both versions.

Parameters

`X, Y` [array-like, optional] The coordinates of the values in `Z`.

`X` and `Y` must both be 2-D with the same shape as `Z` (e.g. created via `numpy.meshgrid()`), or they must both be 1-D such that `len(X) == M` is the number of columns in `Z` and `len(Y) == N` is the number of rows in `Z`.

If not given, they are assumed to be integer indices, i.e. `X = range(M)`, `Y = range(N)`.

`Z` [array-like(`N, M`)] The height values over which the contour is drawn.

`levels` [int or array-like, optional] Determines the number and positions of the contour lines / regions.

If an int `n`, use `n` data intervals; i.e. draw `n+1` contour lines. The level heights are automatically chosen.

If array-like, draw contour lines at the specified levels. The values must be in increasing order.

Returns

:class:`~matplotlib.contour.QuadContourSet`

Other Parameters

`corner_mask` [bool, optional] Enable/disable corner masking, which only has an effect if `Z` is a masked array. If False, any quad touching a masked point is masked out. If True, only the triangular corners of quads nearest those points are always masked out, other triangular corners comprising three unmasked points are contoured as usual.

Defaults to `rcParams['contour.corner_mask']`, which defaults to `True`.

colors [color string or sequence of colors, optional] The colors of the levels, i.e. the lines for contour and the areas for `contourf`.

The sequence is cycled for the levels in ascending order. If the sequence is shorter than the number of levels, it's repeated.

As a shortcut, single color strings may be used in place of one-element lists, i.e. `'red'` instead of `['red']` to color all levels with the same color. This shortcut does only work for color strings, not for other ways of specifying colors.

By default (value `None`), the colormap specified by `cmap` will be used.

alpha [float, optional] The alpha blending value, between 0 (transparent) and 1 (opaque).

cmap [str or Colormap, optional] A Colormap instance or registered colormap name. The colormap maps the level values to colors. Defaults to `rcParams["image.cmap"]`.

If given, colors take precedence over `cmap`.

norm [Normalize, optional] If a colormap is used, the Normalize instance scales the level values to the canonical colormap range `[0, 1]` for mapping to colors. If not given, the default linear scaling is used.

vmin, vmax [float, optional] If not `None`, either or both of these values will be supplied to the Normalize instance, overriding the default color scaling based on levels.

origin [{`None`, `'upper'`, `'lower'`, `'image'`}, optional] Determines the orientation and exact position of `Z` by specifying the position of `Z[0, 0]`. This is only relevant, if `X`, `Y` are not given.

- `None`: `Z[0, 0]` is at `X=0`, `Y=0` in the lower left corner.
- `'lower'`: `Z[0, 0]` is at `X=0.5`, `Y=0.5` in the lower left corner.
- `'upper'`: `Z[0, 0]` is at `X=N+0.5`, `Y=0.5` in the upper left corner.
- `'image'`: Use the value from `rcParams["image.origin"]`. Note: The value `None` in the rcParam is currently handled as `'lower'`.

extent [(`x0`, `x1`, `y0`, `y1`), optional] If origin is not `None`, then extent is interpreted as in `matplotlib.pyplot.imshow()`: it gives the outer pixel boundaries. In this case, the position of `Z[0,0]` is the center of the pixel, not a corner. If origin is `None`, then (`x0`, `y0`) is the position of `Z[0,0]`, and (`x1`, `y1`) is the position of `Z[-1,-1]`.

This keyword is not active if `X` and `Y` are specified in the call to `contour`.

locator [`ticker.Locator` subclass, optional] The locator is used to determine the contour levels if they are not given explicitly via levels. Defaults to `MaxNLocator`.

`extend` [{`'neither'`, `'both'`, `'min'`, `'max'`}, optional] Unless this is `'neither'`, contour levels are automatically added to one or both ends of the range so that all data are included. These added ranges are then mapped to the special colormap values which default to the ends of the colormap range, but can be set via `matplotlib.colors.Colormap.set_under()` and `matplotlib.colors.Colormap.set_over()` methods.

`xunits, yunits` [registered units, optional] Override axis units by specifying an instance of a `matplotlib.units.ConversionInterface`.

`antialiased` [bool, optional] Enable antialiasing, overriding the defaults. For filled contours, the default is `True`. For line contours, it is taken from `rcParams["lines.antialiased"]`.

`Nchunk` [int ≥ 0 , optional] If 0, no subdivision of the domain. Specify a positive integer to divide the domain into subdomains of `nchunk` by `nchunk` quads. Chunking reduces the maximum length of polygons generated by the contouring algorithm which reduces the rendering workload passed on to the backend and also requires slightly less RAM. It can however introduce rendering artifacts at chunk boundaries depending on the backend, the `antialiased` flag and value of `alpha`.

`linewidths` [float or sequence of float, optional] Only applies to contour.

The line width of the contour lines.

If a number, all levels will be plotted with this linewidth.

If a sequence, the levels in ascending order will be plotted with the `linewidths` in the order specified.

Defaults to `rcParams["lines.linewidth"]`.

`linestyles` [{`None`, `'solid'`, `'dashed'`, `'dashdot'`, `'dotted'`}, optional] Only applies to contour.

If `linestyles` is `None`, the default is `'solid'` unless the lines are monochrome. In that case, negative contours will take their linestyle from `rcParams["contour.negative_linestyle"]` setting.

`linestyles` can also be an iterable of the above strings specifying a set of linestyles to be used. If this iterable is shorter than the number of contour levels it will be repeated as necessary.

`hatches` [List[str], optional] Only applies to contourf.

A list of cross hatch patterns to use on the filled areas. If `None`, no hatching will be added to the contour. Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Notes

1. `contourf()` differs from the MATLAB version in that it does not draw the polygon edges.

To draw edges, add line contours with calls to `contour()`.

2. `contourf` fills intervals that are closed at the top; that is, for boundaries `z1` and `z2`, the filled region is:

$$z1 < Z \leq z2$$

There is one exception: if the lowest boundary coincides with the minimum value of the `Z` array, then that minimum value will be included in the lowest interval.

Examples using `matplotlib.pyplot.contour`

- `sphx_glr_gallery_images_contours_and_fields_contour_corner_mask.py`
- `sphx_glr_gallery_images_contours_and_fields_contourf_hatching.py`
- `sphx_glr_gallery_images_contours_and_fields_contour_label_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_irregulardatagrid.py`
- `sphx_glr_gallery_images_contours_and_fields_contour_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_image_transparency_blend.py`
- `sphx_glr_gallery_images_contours_and_fields_contour_image.py`
- `sphx_glr_gallery_images_contours_and_fields_contourf_demo.py`
- `sphx_glr_gallery_event_handling_ginput_manual_clabel_sgskip.py`

75.1.30 `matplotlib.pyplot.contourf`

`matplotlib.pyplot.contourf(*args, **kwargs)`

Plot contours.

Call signature:

```
contour([X, Y], Z, [levels], **kwargs)
```

`contour()` and `contourf()` draw contour lines and filled contours, respectively. Except as noted, function signatures and return values are the same for both versions.

Parameters

`X, Y` [array-like, optional] The coordinates of the values in `Z`.

`X` and `Y` must both be 2-D with the same shape as `Z` (e.g. created via `numpy.meshgrid()`), or they must both be 1-D such that `len(X) == M` is the number of columns in `Z` and `len(Y) == N` is the number of rows in `Z`.

If not given, they are assumed to be integer indices, i.e. `X = range(M)`, `Y = range(N)`.

`Z` [array-like(N, M)] The height values over which the contour is drawn.

levels [int or array-like, optional] Determines the number and positions of the contour lines / regions.

If an int *n*, use *n* data intervals; i.e. draw *n*+1 contour lines. The level heights are automatically chosen.

If array-like, draw contour lines at the specified levels. The values must be in increasing order.

Returns

:class:~matplotlib.contour.QuadContourSet

Other Parameters

corner_mask [bool, optional] Enable/disable corner masking, which only has an effect if *Z* is a masked array. If False, any quad touching a masked point is masked out. If True, only the triangular corners of quads nearest those points are always masked out, other triangular corners comprising three unmasked points are contoured as usual.

Defaults to rcParams['contour.corner_mask'], which defaults to True.

colors [color string or sequence of colors, optional] The colors of the levels, i.e. the lines for contour and the areas for contourf.

The sequence is cycled for the levels in ascending order. If the sequence is shorter than the number of levels, it's repeated.

As a shortcut, single color strings may be used in place of one-element lists, i.e. 'red' instead of ['red'] to color all levels with the same color. This shortcut does only work for color strings, not for other ways of specifying colors.

By default (value None), the colormap specified by *cmap* will be used.

alpha [float, optional] The alpha blending value, between 0 (transparent) and 1 (opaque).

cmap [str or Colormap, optional] A Colormap instance or registered colormap name. The colormap maps the level values to colors. Defaults to rcParams["image.cmap"].

If given, colors take precedence over *cmap*.

norm [Normalize, optional] If a colormap is used, the Normalize instance scales the level values to the canonical colormap range [0, 1] for mapping to colors. If not given, the default linear scaling is used.

vmin, vmax [float, optional] If not None, either or both of these values will be supplied to the Normalize instance, overriding the default color scaling based on levels.

origin [{None, 'upper', 'lower', 'image'}, optional] Determines the orientation and exact position of *Z* by specifying the position of *Z*[0, 0]. This is only relevant, if *X*, *Y* are not given.

- None: $Z[0, 0]$ is at $X=0$, $Y=0$ in the lower left corner.
- 'lower': $Z[0, 0]$ is at $X=0.5$, $Y=0.5$ in the lower left corner.
- 'upper': $Z[0, 0]$ is at $X=N+0.5$, $Y=0.5$ in the upper left corner.
- 'image': Use the value from `rcParams["image.origin"]`. Note: The value None in the rcParam is currently handled as 'lower'.

extent [(x0, x1, y0, y1), optional] If origin is not None, then extent is interpreted as in `matplotlib.pyplot.imshow()`: it gives the outer pixel boundaries. In this case, the position of $Z[0,0]$ is the center of the pixel, not a corner. If origin is None, then (x0, y0) is the position of $Z[0,0]$, and (x1, y1) is the position of $Z[-1,-1]$.

This keyword is not active if X and Y are specified in the call to `contour`.

locator [`ticker.Locator` subclass, optional] The locator is used to determine the contour levels if they are not given explicitly via levels. Defaults to `MaxNLocator`.

extend [{ 'neither', 'both', 'min', 'max' }, optional] Unless this is 'neither', contour levels are automatically added to one or both ends of the range so that all data are included. These added ranges are then mapped to the special colormap values which default to the ends of the colormap range, but can be set via `matplotlib.colors.Colormap.set_under()` and `matplotlib.colors.Colormap.set_over()` methods.

xunits, yunits [registered units, optional] Override axis units by specifying an instance of a `matplotlib.units.ConversionInterface`.

antialiased [bool, optional] Enable antialiasing, overriding the defaults. For filled contours, the default is True. For line contours, it is taken from `rcParams["lines.antialiased"]`.

Nchunk [int ≥ 0 , optional] If 0, no subdivision of the domain. Specify a positive integer to divide the domain into subdomains of `nchunk` by `nchunk` quads. Chunking reduces the maximum length of polygons generated by the contouring algorithm which reduces the rendering workload passed on to the backend and also requires slightly less RAM. It can however introduce rendering artifacts at chunk boundaries depending on the backend, the antialiased flag and value of alpha.

linewidths [float or sequence of float, optional] Only applies to `contour`.

The line width of the contour lines.

If a number, all levels will be plotted with this linewidth.

If a sequence, the levels in ascending order will be plotted with the linewidths in the order specified.

Defaults to `rcParams["lines.linewidth"]`.

linestyles [{None, 'solid', 'dashed', 'dashdot', 'dotted'}, optional] Only applies to `contour`.

If `linestyles` is `None`, the default is `'solid'` unless the lines are monochrome. In that case, negative contours will take their linestyle from `rcParams["contour.negative_linestyle"]` setting.

`linestyles` can also be an iterable of the above strings specifying a set of linestyles to be used. If this iterable is shorter than the number of contour levels it will be repeated as necessary.

`hatches` [List[str], optional] Only applies to `contourf`.

A list of cross hatch patterns to use on the filled areas. If `None`, no hatching will be added to the contour. Hatching is supported in the PostScript, PDF, SVG and Agg backends only.

Notes

1. `contourf()` differs from the MATLAB version in that it does not draw the polygon edges. To draw edges, add line contours with calls to `contour()`.
2. `contourf` fills intervals that are closed at the top; that is, for boundaries `z1` and `z2`, the filled region is:

$z1 < Z \leq z2$

There is one exception: if the lowest boundary coincides with the minimum value of the `Z` array, then that minimum value will be included in the lowest interval.

Examples using `matplotlib.pyplot.contourf`

- `sphinx_glr_gallery_images_contours_and_fields_contour_corner_mask.py`
- `sphinx_glr_gallery_images_contours_and_fields_contourf_hatching.py`
- `sphinx_glr_gallery_images_contours_and_fields_contourf_log.py`
- `sphinx_glr_gallery_images_contours_and_fields_pcolormesh_levels.py`
- `sphinx_glr_gallery_images_contours_and_fields_irregulardatagrid.py`
- `sphinx_glr_gallery_images_contours_and_fields_triinterp_demo.py`
- `sphinx_glr_gallery_images_contours_and_fields_contourf_demo.py`

75.1.31 `matplotlib.pyplot.cool`

`matplotlib.pyplot.cool()`

Set the colormap to `"cool"`.

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.32 matplotlib.pyplot.copper

matplotlib.pyplot.copper()

Set the colormap to "copper".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.33 matplotlib.pyplot.csd

matplotlib.pyplot.csd(x, y, NFFT=None, Fs=None, Fc=None, detrend=None, window=None, noverlap=None, pad_to=None, sides=None, scale_by_freq=None, return_line=None, hold=None, data=None, **kwargs)

Plot the cross-spectral density.

Call signature:

```
csd(x, y, NFFT=256, Fs=2, Fc=0, detrend=mlab.detrend_none,
    window=mlab.window_hanning, noverlap=0, pad_to=None,
    sides='default', scale_by_freq=None, return_line=None, **kwargs)
```

The cross spectral density P_{xy} by Welch's average periodogram method. The vectors x and y are divided into NFFT length segments. Each segment is detrended by function `detrend` and windowed by function `window`. `noverlap` gives the length of the overlap between segments. The product of the direct FFTs of x and y are averaged over each segment to compute P_{xy} , with a scaling to correct for power loss due to windowing.

If `len(x) < NFFT` or `len(y) < NFFT`, they will be zero padded to NFFT.

Parameters

`x, y` [1-D arrays or sequences] Arrays or sequences containing the data

`Fs` [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.

`window` [callable or ndarray] A function or a vector of length NFFT. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.

`sides` [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.

`pad_to` [integer] The number of points to which the data segment is padded when performing the FFT. This can be different from NFFT, which speci-

fies the number of data points used. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` equal to `NFFT`

NFFT [integer] The number of data points used in each block for the FFT. A power 2 is most efficient. The default value is 256. This should NOT be used to get zero padding, or the scaling of the result will be incorrect. Use `pad_to` for this instead.

detrend [{`'default'`, `'constant'`, `'mean'`, `'linear'`, `'none'`} or callable] The function applied to each segment before `fft`-ing, designed to remove the mean or linear trend. Unlike in MATLAB, where the `detrend` parameter is a vector, in matplotlib is it a function. The `pylab` module defines `detrend_none()`, `detrend_mean()`, and `detrend_linear()`, but you can use a custom function as well. You can also use a string to choose one of the functions. `'default'`, `'constant'`, and `'mean'` call `detrend_mean()`. `'linear'` calls `detrend_linear()`. `'none'` calls `detrend_none()`.

scale_by_freq [boolean, optional] Specifies whether the resulting density values should be scaled by the scaling frequency, which gives density in units of Hz^{-1} . This allows for integration over the returned frequency values. The default is `True` for MATLAB compatibility.

noverlap [integer] The number of points of overlap between segments. The default value is 0 (no overlap).

Fc [integer] The center frequency of `x` (defaults to 0), which offsets the `x` extents of the plot to reflect the frequency range used when a signal is acquired and then filtered and downsampled to baseband.

return_line [bool] Whether to include the line object plotted in the returned values. Default is `False`.

Returns

Pxy [1-D array] The values for the cross spectrum $P_{\{xy\}}$ before scaling (complex valued)

freqs [1-D array] The frequencies corresponding to the elements in `Pxy`

line [a `Line2D` instance] The line created by this function. Only returned if `return_line` is `True`.

Other Parameters

****kwargs** : Keyword arguments control the `Line2D` properties:

	Property	Description
<code>agg_filter</code>		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)
<code>alpha</code>		float (0.0 transparent through 1.0 opaque)

Continued on next page

Table 8 – continued from previous page

	Property	Description
animated	bool	
antialiased or aa	bool	
clip_box	a Bbox instance	
clip_on	bool	
clip_path	[(Path, Transform) Patch None]	
color or c	any matplotlib color	
contains	a callable function	
dash_capstyle	['butt' 'round' 'projecting']	
dash_joinstyle	['miter' 'round' 'bevel']	
dashes	sequence of on/off ink in points	
drawstyle	['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']	
figure	a Figure instance	
fillstyle	['full' 'left' 'right' 'bottom' 'top' 'none']	
gid	an id string	
label	object	
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']	
linewidth or lw	float value in points	
marker	A valid marker style	
markeredgewidth or mec	any matplotlib color	
markeredgecolor or mew	float value in points	
markerfacecolor or mfc	any matplotlib color	
markerfacecoloralt or mfcalt	any matplotlib color	
markersize or ms	float	
markevery	[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]	
path_effects	AbstractPathEffect	
picker	float distance in points or callable pick function fn(artist, event)	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
solid_capstyle	['butt' 'round' 'projecting']	
solid_joinstyle	['miter' 'round' 'bevel']	
transform	a matplotlib.transforms.Transform instance	
url	a url string	
visible	bool	
xdata	1D array	
ydata	1D array	
zorder	float	

See also:

`psd()` `psd()` is the equivalent to setting `y=x`.

Notes

For plotting, the power is plotted as $10 \log_{10}(P_{xy})$ for decibels, though $P_{\{xy\}}$ itself is returned.

References

Bendat & Piersol – Random Data: Analysis and Measurement Procedures, John Wiley & Sons (1986)

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'x', 'y'.
-

75.1.34 matplotlib.pyplot.delaxes

`matplotlib.pyplot.delaxes(ax=None)`

Remove the given Axes `ax` from the current figure. If `ax` is `None`, the current axes is removed. A `KeyError` is raised if the axes doesn't exist.

75.1.35 matplotlib.pyplot.disconnect

`matplotlib.pyplot.disconnect(cid)`

Disconnect callback id `cid`

Examples

Usage:

```
cid = canvas.mpl_connect('button_press_event', on_press)
#...later
canvas.mpl_disconnect(cid)
```

Examples using `matplotlib.pyplot.disconnect`

- `sphinx_glr_gallery_event_handling_coords_demo.py`

75.1.36 matplotlib.pyplot.draw

`matplotlib.pyplot.draw()`

Redraw the current figure.

This is used to update a figure that has been altered, but not automatically re-drawn. If interactive mode is on (`ion()`), this should be only rarely needed, but there may be ways to modify the state of a figure without marking it as stale. Please report these cases as bugs.

A more object-oriented alternative, given any Figure instance, `fig`, that was created using a `pyplot` function, is:

```
fig.canvas.draw_idle()
```

Examples using `matplotlib.pyplot.draw`

- `sphinx_glr_gallery_pyplots_whats_new_99_axes_grid.py`
- `sphinx_glr_gallery_shapes_and_collections_fancybox_demo.py`
- `sphinx_glr_gallery_text_labels_and_annotations_fancytextbox_demo.py`
- `sphinx_glr_gallery_text_labels_and_annotations_multiline.py`
- `sphinx_glr_gallery_text_labels_and_annotations_fancyarrow_demo.py`
- `sphinx_glr_gallery_text_labels_and_annotations_usetex_baseline_test.py`
- `sphinx_glr_gallery_text_labels_and_annotations_demo_text_path.py`
- `sphinx_glr_gallery_text_labels_and_annotations_arrow_demo.py`
- `sphinx_glr_gallery_axes_grid1_simple_axisline4.py`
- `sphinx_glr_gallery_axes_grid1_simple_axesgrid2.py`
- `sphinx_glr_gallery_axes_grid1_demo_colorbar_with_inset_locator.py`
- `sphinx_glr_gallery_axes_grid1_parasite_simple2.py`
- `sphinx_glr_gallery_axes_grid1_scatter_hist_locatable_axes.py`
- `sphinx_glr_gallery_axes_grid1_demo_edge_colorbar.py`
- `sphinx_glr_gallery_axes_grid1_demo_axes_divider.py`
- `sphinx_glr_gallery_axes_grid1_demo_axes_grid2.py`
- `sphinx_glr_gallery_axes_grid1_demo_axes_grid.py`
- `sphinx_glr_gallery_axisartist_demo_parasite_axes2.py`
- `sphinx_glr_gallery_axisartist_demo_curvelinear_grid.py`
- `sphinx_glr_gallery_event_handling_ginput_manual_clabel_sgskip.py`
- `sphinx_glr_gallery_misc_cursor_demo_sgskip.py`

- sphx_glr_gallery_misc_anchored_artists.py
- sphx_glr_gallery_mplot3d_rotate_axes3d.py
- sphx_glr_gallery_scales_aspect_loglog.py
- sphx_glr_gallery_widgets_textbox.py
- sphx_glr_gallery_widgets_check_buttons.py
- sphx_glr_gallery_widgets_buttons.py
- sphx_glr_gallery_widgets_radio_buttons.py

75.1.37 matplotlib.pyplot.errorbar

```
matplotlib.pyplot.errorbar(x, y, yerr=None, xerr=None, fmt="", ecolor=None,
                           linewidth=None, capsize=None, barsabove=False, lolims=False,
                           uplims=False, xlolims=False, xuplims=False, errorevery=1,
                           capthick=None, hold=None, data=None, **kwargs)
```

Plot y versus x as lines and/or markers with attached errorbars.

x, y define the data locations, xerr, yerr define the errorbar sizes. By default, this draws the data markers/lines as well the errorbars. Use `fmt='none'` to draw errorbars without any data markers.

Parameters

x, y [scalar or array-like] The data positions.

xerr, yerr [scalar or array-like, shape(N,) or shape(2,N), optional] The errorbar sizes:

- scalar: Symmetric +/- values for all data points.
- shape(N,): Symmetric +/-values for each data point.
- shape(2,N): Separate - and + values for each bar. First row contains the lower errors, the second row contains the upper errors.
- None: No errorbar.

See `/gallery/statistics/errorbar_features` for an example on the usage of xerr and yerr.

fmt [plot format string, optional, default: ""] The format for the data points / data lines. See `plot` for details.

Use 'none' (case insensitive) to plot errorbars without any data markers.

ecolor [mpl color, optional, default: None] A matplotlib color arg which gives the color the errorbar lines. If None, use the color of the line connecting the markers.

linewidth [scalar, optional, default: None] The linewidth of the errorbar lines. If None, the linewidth of the current style is used.

- `capsize` [scalar, optional, default: None] The length of the error bar caps in points. If None, it will take the value from `rcParams["errorbar.capsize"]`.
- `capthick` [scalar, optional, default: None] An alias to the keyword argument `markedgedwidth` (a.k.a. `mew`). This setting is a more sensible name for the property that controls the thickness of the error bar cap in points. For backwards compatibility, if `mew` or `markedgedwidth` are given, then they will over-ride `capthick`. This may change in future releases.
- `barsabove` [bool, optional, default: False] If True, will plot the errorbars above the plot symbols. Default is below.
- `lolims`, `uplims`, `xlolims`, `xuplims` [bool, optional, default: None] These arguments can be used to indicate that a value gives only upper/lower limits. In that case a caret symbol is used to indicate this. `lims`-arguments may be of the same type as `xerr` and `yerr`. To use limits with inverted axes, `set_xlim()` or `set_ylim()` must be called before `errorbar()`.
- `errorevery` [positive integer, optional, default: 1] Subsamples the errorbars. e.g., if `errorevery=5`, errorbars for every 5-th datapoint will be plotted. The data plot itself still shows all data points.

Returns

- `container` [ErrorbarContainer] The container contains:
- `plotline`: Line2D instance of x, y plot markers and/or line.
 - `caplines`: A tuple of Line2D instances of the error bar caps.
 - `barlinecols`: A tuple of LineCollection with the horizontal and vertical error ranges.

Other Parameters

****kwargs** : All other keyword arguments are passed on to the plot command for the markers. For example, this code makes big red squares with thick green edges:

```
x,y,yerr = rand(3,10)
errorbar(x, y, yerr, marker='s', mfc='red',
         mec='green', ms=20, mew=4)
```

where `mfc`, `mec`, `ms` and `mew` are aliases for the longer property names, `markerfacecolor`, `markedgedcolor`, `markersize` and `markedgedwidth`.

Valid kwargs for the marker properties are Lines2D properties:

	Property	Description
<code>agg_filter</code>		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)
<code>alpha</code>		float (0.0 transparent through 1.0 opaque)
<code>animated</code>		bool

Continued on next page

Table 9 – continued from previous page

	Property	Description
antialiased or aa	bool	
clip_box	a Bbox instance	
clip_on	bool	
clip_path	[(Path, Transform) Patch None]	
color or c	any matplotlib color	
contains	a callable function	
dash_capstyle	['butt' 'round' 'projecting']	
dash_joinstyle	['miter' 'round' 'bevel']	
dashes	sequence of on/off ink in points	
drawstyle	['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']	
figure	a Figure instance	
fillstyle	['full' 'left' 'right' 'bottom' 'top' 'none']	
gid	an id string	
label	object	
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']	
linewidth or lw	float value in points	
marker	A valid marker style	
markeredgecolor or mec	any matplotlib color	
markeredgewidth or mew	float value in points	
markerfacecolor or mfc	any matplotlib color	
markerfacecoloralt or mfcalt	any matplotlib color	
markersize or ms	float	
markevery	[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]	
path_effects	AbstractPathEffect	
picker	float distance in points or callable pick function fn(artist, event)	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
solid_capstyle	['butt' 'round' 'projecting']	
solid_joinstyle	['miter' 'round' 'bevel']	
transform	a matplotlib.transforms.Transform instance	
url	a url string	
visible	bool	
xdata	1D array	
ydata	1D array	
zorder	float	

Notes

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All arguments with the following names: 'x', 'xerr', 'y', 'yerr'.

Examples using `matplotlib.pyplot.errorbar`

- `sphx_glr_gallery_lines_bars_and_markers_errorbar_limits_simple.py`

75.1.38 `matplotlib.pyplot.eventplot`

`matplotlib.pyplot.eventplot(positions, orientation='horizontal', lineoffsets=1, linelengths=1, linewidths=None, colors=None, linestyle='solid', hold=None, data=None, **kwargs)`

Plot identical parallel lines at the given positions.

`positions` should be a 1D or 2D array-like object, with each row corresponding to a row or column of lines.

This type of plot is commonly used in neuroscience for representing neural events, where it is usually called a spike raster, dot raster, or raster plot.

However, it is useful in any situation where you wish to show the timing or position of multiple sets of discrete events, such as the arrival times of people to a business on each day of the month or the date of hurricanes each year of the last century.

Parameters

`positions` [1D or 2D array-like object] Each value is an event. If `positions` is a 2D array-like, each row corresponds to a row or a column of lines (depending on the `orientation` parameter).

`orientation` [{'horizontal', 'vertical'}, optional] Controls the direction of the event collections:

- 'horizontal' : the lines are arranged horizontally in rows, and are vertical.
- 'vertical' : the lines are arranged vertically in columns, and are horizontal.

`lineoffsets` [scalar or sequence of scalars, optional, default: 1] The offset of the center of the lines from the origin, in the direction orthogonal to `orientation`.

`linelengths` [scalar or sequence of scalars, optional, default: 1] The total height of the lines (i.e. the lines stretches from `lineoffset - linelength/2` to `lineoffset + linelength/2`).

`linewidths` [scalar, scalar sequence or None, optional, default: None] The line width(s) of the event lines, in points. If it is None, defaults to its `rcParams` setting.

`colors` [color, sequence of colors or None, optional, default: None] The color(s) of the event lines. If it is None, defaults to its `rcParams` setting.

`linestyles` [str or tuple or a sequence of such values, optional] Default is 'solid'. Valid strings are ['solid', 'dashed', 'dashdot', 'dotted', '-', '—', '-.', ':']. Dash tuples should be of the form:

(offset, onoffseq),

where onoffseq is an even length tuple of on and off ink in points.

****kwargs** [optional] Other keyword arguments are line collection properties. See `LineCollection` for a list of the valid properties.

Returns

`list` [A list of `EventCollection` objects.] Contains the `EventCollection` that were added.

Notes

For `linelengths`, `linewidths`, `colors`, and `linestyles`, if only a single value is given, that value is applied to all lines. If an array-like is given, it must have the same length as `positions`, and each value will be applied to the corresponding row of the array.

Examples

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'colors', 'linelengths', 'lineoffsets', 'linestyles', 'linewidths', 'positions'.
-

75.1.39 matplotlib.pyplot.figimage

`matplotlib.pyplot.figimage(*args, **kwargs)`

Add a non-resampled image to the figure.

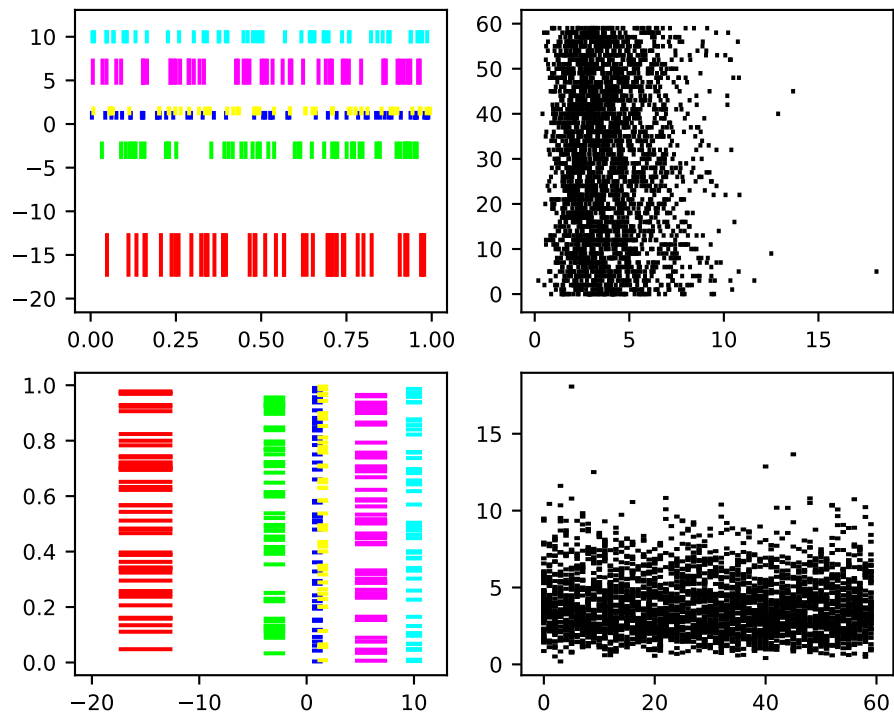
The image is attached to the lower or upper left corner depending on origin.

Parameters

X The image data. This is an array of one of the following shapes:

- `MxN`: luminance (grayscale) values
- `MxNx3`: RGB values
- `MxNx4`: RGBA values

`xo, yo` [int] The x/y image offset in pixels.



`alpha` [None or float] The alpha blending value.

`norm` [matplotlib.colors.Normalize] A Normalize instance to map the luminance to the interval [0, 1].

`cmap` [str or matplotlib.colors.Colormap] The colormap to use. Default: `rcParams["image.cmap"]`.

`vmin`, `vmax` [scalar] If `norm` is not given, these values set the data limits for the colormap.

`origin` [{ 'upper', 'lower' }] Indicates where the [0, 0] index of the array is in the upper left or lower left corner of the axes. Defaults to `rcParams["image.origin"]`.

`resize` [bool] If True, resize the figure to match the given image size.

Returns

:class:'matplotlib.image.FigureImage'

Other Parameters

****kwargs** Additional kwargs are Artist kwargs passed on to FigureImage.

Notes

`figimage` complements the axes image (`imshow()`) which will be resampled to fit the current axes. If you want a resampled image to fill the entire figure, you can define an Axes with extent `[0,0,1,1]`.

Examples:

```
f = plt.figure()
nx = int(f.get_figwidth() * f.dpi)
ny = int(f.get_figheight() * f.dpi)
data = np.random.random((ny, nx))
f.figimage(data)
plt.show()
```

Examples using `matplotlib.pyplot.figimage`

- `sphx_glr_gallery_images_contours_and_fields_figimage_demo.py`

75.1.40 `matplotlib.pyplot.figlegend`

`matplotlib.pyplot.figlegend(*args, **kwargs)`

Place a legend in the figure.

`labels` a sequence of strings

`handles` a sequence of `Line2D` or `Patch` instances

`loc` can be a string or an integer specifying the legend location

A `matplotlib.legend.Legend` instance is returned.

Examples

To make a legend from existing artists on every axes:

```
figlegend()
```

To make a legend for a list of lines and labels:

```
figlegend( (line1, line2, line3),
           ('label1', 'label2', 'label3'),
           'upper right' )
```

See also:

`legend()`

75.1.41 matplotlib.pyplot.figure_exists

matplotlib.pyplot.figure_exists(num)

75.1.42 matplotlib.pyplot.figtext

matplotlib.pyplot.figtext(*args, **kwargs)

Add text to figure.

Call signature:

```
text(x, y, s, fontdict=None, **kwargs)
```

Add text to figure at location x, y (relative 0-1 coords). See text() for the meaning of the other arguments.

kwargs control the Text properties:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value
alpha	float (0.0 transparent through 1.0 opaque)
animated	bool
backgroundcolor	any matplotlib color
bbox	FancyBboxPatch prop dict
clip_box	a matplotlib.transforms.Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color	any matplotlib color
contains	a callable function
family or fontfamily or fontname or name	[FONTNAME 'serif' 'sans-serif' 'cursive' 'fantasy' 'monospace']
figure	a Figure instance
fontproperties or font_properties	a matplotlib.font_manager.FontProperties instance
gid	an id string
horizontalalignment or ha	['center' 'right' 'left']
label	object
linespacing	float (multiple of font size)
multialignment or ma	['left' 'right' 'center']
path_effects	AbstractPathEffect
picker	[None bool float callable]
position	(x,y)
rasterized	bool or None
rotation	[angle in degrees 'vertical' 'horizontal']
rotation_mode	[None "default" "anchor"]
size or fontsize	[size in points 'xx-small' 'x-small' 'small' 'medium' 'large' 'x-large']
sketch_params	(scale: float, length: float, randomness: float)

Continued on next page

Table 10 – continued from previous page

	Property	Description
snap		bool or None
stretch or fontstretch		[a numeric value in range 0-1000 'ultra-condensed' 'extra-condensed']
style or fontstyle		['normal' 'italic' 'oblique']
text		string or anything printable with '%s' conversion.
transform		Transform
url		a url string
usetex		bool or None
variant or fontvariant		['normal' 'small-caps']
verticalalignment or va		['center' 'top' 'bottom' 'baseline']
visible		bool
weight or fontweight		[a numeric value in range 0-1000 'ultralight' 'light' 'normal' 'bold' 'extra-bold' 'black']
wrap		bool
x		float
y		float
zorder		float

Examples using `matplotlib.pyplot.figtext`

- `sphx_glr_gallery_showcase_integral.py`

75.1.43 `matplotlib.pyplot.figure`

`matplotlib.pyplot.figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameon=True, FigureClass=<class 'matplotlib.figure.Figure'>, clear=False, **kwargs)`

Creates a new figure.

Parameters

num [integer or string, optional, default: none] If not provided, a new figure will be created, and the figure number will be incremented. The figure object holds this number in a `number` attribute. If `num` is provided, and a figure with this id already exists, make it active, and returns a reference to it. If this figure does not exist, create it and returns it. If `num` is a string, the window title will be set to this figure's `num`.

figsize [tuple of integers, optional, default: None] width, height in inches. If not provided, defaults to `rc figure.figsize`.

dpi [integer, optional, default: None] resolution of the figure. If not provided, defaults to `rc figure.dpi`.

facecolor : the background color. If not provided, defaults to `rc figure.facecolor`.

edgecolor : the border color. If not provided, defaults to rc figure.edgecolor.

frameon [bool, optional, default: True] If False, suppress drawing the figure frame.

FigureClass [class derived from matplotlib.figure.Figure] Optionally use a custom Figure instance.

clear [bool, optional, default: False] If True and the figure already exists, then it is cleared.

Returns

figure [Figure] The Figure instance returned will also be passed to new_figure_manager in the backends, which allows to hook custom Figure classes into the pylab interface. Additional kwargs will be passed to the figure init function.

Notes

If you are creating many figures, make sure you explicitly call "close" on the figures you are not using, because this will enable pylab to properly clean up the memory.

rcParams defines the default values, which can be modified in the matplotlibrc file

Examples using matplotlib.pyplot.figure

- sphx_glr_gallery_api_mathtext_asarray.py
- sphx_glr_gallery_api_sankey_links.py
- sphx_glr_gallery_api_sankey_basics.py
- sphx_glr_gallery_api_logos2.py
- sphx_glr_gallery_api_sankey_rankine.py
- sphx_glr_gallery_api_skewt.py
- sphx_glr_gallery_pyplots_fig_x.py
- sphx_glr_gallery_pyplots_pyplot_two_subplots.py
- sphx_glr_gallery_pyplots_whats_new_99_mplot3d.py
- sphx_glr_gallery_pyplots_fig_axes_labels_simple.py
- sphx_glr_gallery_pyplots_fig_axes_customize_simple.py
- sphx_glr_gallery_pyplots_annotation_polar.py
- sphx_glr_gallery_pyplots_whats_new_99_axes_grid.py
- sphx_glr_gallery_pyplots_text_commands.py
- sphx_glr_gallery_pyplots_whats_new_1_subplot3d.py

- sphx_glr_gallery_pyplots_whats_new_99_spines.py
- sphx_glr_gallery_pyplots_compound_path_demo.py
- sphx_glr_gallery_pyplots_pyplot_scales.py
- sphx_glr_gallery_pyplots_whats_new_98_4_fancy.py
- sphx_glr_gallery_pyplots_text_layout.py
- sphx_glr_gallery_subplots_axes_and_figures_multiple_figs_demo.py
- sphx_glr_gallery_subplots_axes_and_figures_geo_demo.py
- sphx_glr_gallery_subplots_axes_and_figures_custom_figure_class.py
- sphx_glr_gallery_subplots_axes_and_figures_align_labels_demo.py
- sphx_glr_gallery_subplots_axes_and_figures_demo_tight_layout.py
- sphx_glr_gallery_subplots_axes_and_figures_axes_zoom_effect.py
- sphx_glr_gallery_lines_bars_and_markers_errorbar_limits_simple.py
- sphx_glr_gallery_lines_bars_and_markers_scatter_hist.py
- sphx_glr_gallery_lines_bars_and_markers_eventcollection_demo.py
- sphx_glr_gallery_lines_bars_and_markers_linestyles.py
- sphx_glr_gallery_lines_bars_and_markers_markevery_demo.py
- sphx_glr_gallery_lines_bars_and_markers_psd_demo.py
- sphx_glr_gallery_images_contours_and_fields_figimage_demo.py
- sphx_glr_gallery_images_contours_and_fields_barcode_demo.py
- sphx_glr_gallery_images_contours_and_fields_layer_images.py
- sphx_glr_gallery_images_contours_and_fields_plot_streamplot.py
- sphx_glr_gallery_shapes_and_collections_fancybox_demo.py
- sphx_glr_gallery_text_labels_and_annotations_dfrac_demo.py
- sphx_glr_gallery_text_labels_and_annotations_autowrap.py
- sphx_glr_gallery_text_labels_and_annotations_multiline.py
- sphx_glr_gallery_text_labels_and_annotations_fancyarrow_demo.py
- sphx_glr_gallery_text_labels_and_annotations_demo_text_rotation_mode.py
- sphx_glr_gallery_text_labels_and_annotations_stix_fonts_demo.py
- sphx_glr_gallery_text_labels_and_annotations_font_table_ttf_sgskip.py
- sphx_glr_gallery_text_labels_and_annotations_usetex_baseline_test.py
- sphx_glr_gallery_text_labels_and_annotations_mathtext_examples.py
- sphx_glr_gallery_text_labels_and_annotations_demo_text_path.py

- sphx_glr_gallery_text_labels_and_annotations_arrow_demo.py
- sphx_glr_gallery_pie_and_polar_charts_polar_legend.py
- sphx_glr_gallery_pie_and_polar_charts_polar_scatter.py
- sphx_glr_gallery_showcase_xkcd.py
- sphx_glr_gallery_showcase_mandelbrot.py
- sphx_glr_gallery_showcase_anatomy.py
- sphx_glr_gallery_showcase_firefox.py
- Frame grabbing
- Animated image using a precomputed list of images
- MATPLOTLIB UNCHAINED
- Animated 3D random walk
- The double pendulum problem
- Rain simulation
- sphx_glr_gallery_axes_grid1_demo_imagegrid_aspect.py
- sphx_glr_gallery_axes_grid1_simple_axesgrid.py
- sphx_glr_gallery_axes_grid1_simple_rgb.py
- sphx_glr_gallery_axes_grid1_simple_axesgrid2.py
- sphx_glr_gallery_axes_grid1_simple_axes_divider1.py
- sphx_glr_gallery_axes_grid1_simple_axes_divider2.py
- sphx_glr_gallery_axes_grid1_simple_axes_divider3.py
- sphx_glr_gallery_axes_grid1_parasite_simple2.py
- sphx_glr_gallery_axes_grid1_demo_fixed_size_axes.py
- sphx_glr_gallery_axes_grid1_make_room_for_ylabel_using_axesgrid.py
- sphx_glr_gallery_axes_grid1_demo_axes_rgb.py
- sphx_glr_gallery_axes_grid1_demo_edge_colorbar.py
- sphx_glr_gallery_axes_grid1_demo_axes_divider.py
- sphx_glr_gallery_axes_grid1_demo_axes_grid2.py
- sphx_glr_gallery_axes_grid1_demo_axes_grid.py
- sphx_glr_gallery_axes_grid1_inset_locator_demo.py
- sphx_glr_gallery_axisartist_simple_axisline3.py
- sphx_glr_gallery_axisartist_simple_axis_direction01.py
- sphx_glr_gallery_axisartist_axis_direction_demo_step01.py

- sphx_glr_gallery_axisartist_simple_axisartist1.py
- sphx_glr_gallery_axisartist_simple_axis_direction03.py
- sphx_glr_gallery_axisartist_simple_axisline2.py
- sphx_glr_gallery_axisartist_demo_axisline_style.py
- sphx_glr_gallery_axisartist_simple_axisline.py
- sphx_glr_gallery_axisartist_demo_ticklabel_alignment.py
- sphx_glr_gallery_axisartist_axis_direction_demo_step02.py
- sphx_glr_gallery_axisartist_demo_ticklabel_direction.py
- sphx_glr_gallery_axisartist_axis_direction_demo_step03.py
- sphx_glr_gallery_axisartist_demo_parasite_axes.py
- sphx_glr_gallery_axisartist_axis_direction_demo_step04.py
- sphx_glr_gallery_axisartist_demo_curvelinear_grid2.py
- sphx_glr_gallery_axisartist_demo_floating_axis.py
- sphx_glr_gallery_axisartist_demo_axis_direction.py
- sphx_glr_gallery_axisartist_simple_axis_pad.py
- sphx_glr_gallery_axisartist_demo_curvelinear_grid.py
- sphx_glr_gallery_axisartist_demo_floating_axes.py
- sphx_glr_gallery_event_handling_close_event.py
- sphx_glr_gallery_misc_agg_buffer_to_array.py
- sphx_glr_gallery_misc_hyperlinks_sgskip.py
- sphx_glr_gallery_misc_zorder_demo.py
- sphx_glr_gallery_misc_contour_manual.py
- sphx_glr_gallery_misc_transoffset.py
- sphx_glr_gallery_misc_patheffect_demo.py
- sphx_glr_gallery_misc_multipage_pdf.py
- sphx_glr_gallery_misc_svg_filter_line.py
- sphx_glr_gallery_misc_svg_filter_pie.py
- sphx_glr_gallery_misc_demo_agg_filter.py
- sphx_glr_gallery_mplot3d_contourf3d.py
- sphx_glr_gallery_mplot3d_wire3d.py
- sphx_glr_gallery_mplot3d_contour3d_2.py
- sphx_glr_gallery_mplot3d_contour3d.py

- sphx_glr_gallery_mplot3d_rotate_axes3d.py
- sphx_glr_gallery_mplot3d_offset.py
- sphx_glr_gallery_mplot3d_lines3d.py
- sphx_glr_gallery_mplot3d_surface3d_2.py
- sphx_glr_gallery_mplot3d_3d_bars.py
- sphx_glr_gallery_mplot3d_quiver3d.py
- sphx_glr_gallery_mplot3d_surface3d_radial.py
- sphx_glr_gallery_mplot3d_voxels.py
- sphx_glr_gallery_mplot3d_surface3d.py
- sphx_glr_gallery_mplot3d_text3d.py
- sphx_glr_gallery_mplot3d_contour3d_3.py
- sphx_glr_gallery_mplot3d_contourf3d_2.py
- sphx_glr_gallery_mplot3d_trisurf3d.py
- sphx_glr_gallery_mplot3d_scatter3d.py
- sphx_glr_gallery_mplot3d_mixed_subplots.py
- sphx_glr_gallery_mplot3d_surface3d_3.py
- sphx_glr_gallery_mplot3d_hist3d.py
- sphx_glr_gallery_mplot3d_bars3d.py
- sphx_glr_gallery_mplot3d_voxels_rgb.py
- sphx_glr_gallery_mplot3d_tricontour3d.py
- sphx_glr_gallery_mplot3d_voxels_torus.py
- sphx_glr_gallery_mplot3d_subplot3d.py
- sphx_glr_gallery_mplot3d_wire3d_animation.py
- sphx_glr_gallery_mplot3d_tricontourf3d.py
- sphx_glr_gallery_mplot3d_voxels_numpy_logo.py
- sphx_glr_gallery_mplot3d_2dcollections3d.py
- sphx_glr_gallery_mplot3d_lorenz_attractor.py
- sphx_glr_gallery_mplot3d_polys3d.py
- sphx_glr_gallery_mplot3d_trisurf3d_2.py
- sphx_glr_gallery_mplot3d_pathpatch3d.py
- sphx_glr_gallery_recipes_create_subplots.py
- sphx_glr_gallery_specialty_plots_mri_with_eeg.py

- [sphx_glr_gallery_ticks_and_spines_spine_placement_demo.py](#)
- [sphx_glr_gallery_ticks_and_spines_tick-locators.py](#)
- [sphx_glr_gallery_ticks_and_spines_tick-formatters.py](#)
- [sphx_glr_gallery_units_ellipse_with_units.py](#)
- [sphx_glr_gallery_user_interfaces_toolmanager_sgskip.py](#)
- [sphx_glr_gallery_user_interfaces_svg_histogram_sgskip.py](#)
- [sphx_glr_gallery_userdemo_demo_gridspec05.py](#)
- [sphx_glr_gallery_userdemo_demo_gridspec01.py](#)
- [sphx_glr_gallery_userdemo_demo_gridspec02.py](#)
- [sphx_glr_gallery_userdemo_pgf_texsystem_sgskip.py](#)
- [sphx_glr_gallery_userdemo_pgf_fonts_sgskip.py](#)
- [sphx_glr_gallery_userdemo_demo_gridspec03.py](#)
- [sphx_glr_gallery_userdemo_demo_gridspec04.py](#)
- [sphx_glr_gallery_userdemo_pgf_preamble_sgskip.py](#)
- [sphx_glr_gallery_userdemo_demo_gridspec06.py](#)
- [sphx_glr_gallery_widgets_cursor.py](#)
- [sphx_glr_gallery_widgets_menu.py](#)
- [Image tutorial](#)
- [Usage Guide](#)
- [Pyplot tutorial](#)
- [Artist tutorial](#)
- [Customizing Figure Layouts Using GridSpec and Other Functions](#)
- [Tight Layout guide](#)
- [origin and extent in imshow](#)
- [Constrained Layout Guide](#)
- [Path effects guide](#)
- [Transformations Tutorial](#)
- [Specifying Colors](#)
- [Text properties and layout](#)
- [Text in Matplotlib Plots](#)

75.1.44 matplotlib.pyplot.fill

matplotlib.pyplot.fill(*args, **kwargs)

Plot filled polygons.

Parameters

args [sequence of x, y, [color]] Each polygon is defined by the lists of x and y positions of its nodes, optionally followed by a color specifier. See matplotlib.colors for supported color specifiers. The standard color cycle is used for polygons without a color specifier.

You can plot multiple polygons by providing multiple x, y, [color] groups.

For example, each of the following is legal:

ax.fill(x, y)	# a polygon with default color
ax.fill(x, y, "b")	# a blue polygon
ax.fill(x, y, x2, y2)	# two polygons
ax.fill(x, y, "b", x2, y2, "r")	# a blue and a red polygon

Returns

a list of :class:`~matplotlib.patches.Polygon`

Other Parameters

**kwargs [Polygon properties]

Notes

Use fill_between() if you would like to fill the region between two curves.

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All arguments with the following names: 'x', 'y'.
-

Examples using matplotlib.pyplot.fill

- sphx_glr_gallery_event_handling_ginput_manual_clabel_sgskip.py
- sphx_glr_gallery_misc_fill_spiral.py

75.1.45 matplotlib.pyplot.fill_between

`matplotlib.pyplot.fill_between(x, y1, y2=0, where=None, interpolate=False, step=None, hold=None, data=None, **kwargs)`

Fill the area between two horizontal curves.

The curves are defined by the points $(x, y1)$ and $(x, y2)$. This creates one or multiple polygons describing the filled area.

You may exclude some horizontal sections from filling using `where`.

By default, the edges connect the given points directly. Use `step` if the filling should be a step function, i.e. constant in between `x`.

Parameters

`x` [array (length N)] The x coordinates of the nodes defining the curves.

`y1` [array (length N) or scalar] The y coordinates of the nodes defining the first curve.

`y2` [array (length N) or scalar, optional, default: 0] The y coordinates of the nodes defining the second curve.

`where` [array of bool (length N), optional, default: None] Define where to exclude some horizontal regions from being filled. The filled regions are defined by the coordinates `x[where]`. More precisely, fill between `x[i]` and `x[i+1]` if `where[i]` and `where[i+1]`. Note that this definition implies that an isolated True value between two False values in `where` will not result in filling. Both sides of the True position remain unfilled due to the adjacent False values.

`interpolate` [bool, optional] This option is only relevant if `where` is used and the two curves are crossing each other.

Semantically, `where` is often used for $y1 > y2$ or similar. By default, the nodes of the polygon defining the filled region will only be placed at the positions in the `x` array. Such a polygon cannot describe the above semantics close to the intersection. The x-sections containing the intersection are simply clipped.

Setting `interpolate` to True will calculate the actual intersection point and extend the filled region up to this point.

`step` [{'pre', 'post', 'mid'}, optional] Define step if the filling should be a step function, i.e. constant in between `x`. The value determines where the step will occur:

- 'pre': The y value is continued constantly to the left from every x position, i.e. the interval $(x[i-1], x[i])$ has the value `y[i]`.
- 'post': The y value is continued constantly to the right from every x position, i.e. the interval $[x[i], x[i+1])$ has the value `y[i]`.
- 'mid': Steps occur half-way between the x positions.

Returns

‘PolyCollection’ A PolyCollection containing the plotted polygons.

Other Parameters

****kwargs** All other keyword arguments are passed on to PolyCollection. They control the Polygon properties:

	Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) float array	
alpha	float or None	
animated	bool	
antialiased or antialiaseds	Boolean or sequence of booleans	
array	ndarray	
capstyle	unknown	
clim	a length 2 sequence of floats; may be overridden in methods that have vmin and vmax	
clip_box	a Bbox instance	
clip_on	bool	
clip_path	[(Path, Transform) Patch None]	
cmap	a colormap or registered colormap name	
color	matplotlib color arg or sequence of rgba tuples	
contains	a callable function	
edgecolor or edgecolors	matplotlib color spec or sequence of specs	
facecolor or facecolors	matplotlib color spec or sequence of specs	
figure	a Figure instance	
gid	an id string	
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']	
joinstyle	unknown	
label	object	
linestyle or dashes or linestyles	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':']	
linewidth or linewidths or lw	float or sequence of floats	
norm	Normalize	
offset_position	['screen' 'data']	
offsets	float or sequence of floats	
path_effects	AbstractPathEffect	
picker	[None bool float callable]	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
transform	Transform	
url	a url string	
urls	List[str] or None	
visible	bool	
zorder	float	

See also:

`fill_betweenx` Fill between two sets of x-values.

Notes

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'where', 'x', 'y1', 'y2'.
-

Examples using `matplotlib.pyplot.fill_between`

- `sphx_glr_gallery_text_labels_and_annotations_mathtext_examples.py`

75.1.46 `matplotlib.pyplot.fill_betweenx`

`matplotlib.pyplot.fill_betweenx`(y, x1, x2=0, where=None, step=None, interpolate=False, hold=None, data=None, **kwargs)

Fill the area between two vertical curves.

The curves are defined by the points (x1, y) and (x2, y). This creates one or multiple polygons describing the filled area.

You may exclude some vertical sections from filling using `where`.

By default, the edges connect the given points directly. Use `step` if the filling should be a step function, i.e. constant in between y.

Parameters

`y` [array (length N)] The y coordinates of the nodes defining the curves.

`x1` [array (length N) or scalar] The x coordinates of the nodes defining the first curve.

`x2` [array (length N) or scalar, optional, default: 0] The x coordinates of the nodes defining the second curve.

`where` [array of bool (length N), optional, default: None] Define where to exclude some vertical regions from being filled. The filled regions are defined by the coordinates `y[where]`. More precisely, fill between `y[i]` and `y[i+1]` if `where[i]` and `where[i+1]`. Note that this definition implies that an isolated True value between two False values in `where` will not result in filling. Both sides of the True position remain unfilled due to the adjacent False values.

`interpolate` [bool, optional] This option is only relevant if `where` is used and the two curves are crossing each other.

Semantically, where is often used for $x_1 > x_2$ or similar. By default, the nodes of the polygon defining the filled region will only be placed at the positions in the y array. Such a polygon cannot describe the above semantics close to the intersection. The y -sections containing the intersection are simply clipped.

Setting `interpolate` to `True` will calculate the actual intersection point and extend the filled region up to this point.

`step` [{`'pre'`, `'post'`, `'mid'`}, optional] Define step if the filling should be a step function, i.e. constant in between y . The value determines where the step will occur:

- `'pre'`: The y value is continued constantly to the left from every x position, i.e. the interval $(x[i-1], x[i])$ has the value $y[i]$.
- `'post'`: The y value is continued constantly to the right from every x position, i.e. the interval $[x[i], x[i+1])$ has the value $y[i]$.
- `'mid'`: Steps occur half-way between the x positions.

Returns

`'PolyCollection'` A `PolyCollection` containing the plotted polygons.

Other Parameters

****kwargs** All other keyword arguments are passed on to `PolyCollection`. They control the `Polygon` properties:

	Property	Description
<code>agg_filter</code>		a filter function, which takes a $(m, n, 3)$ float array and a dpi value, and returns a (
<code>alpha</code>		float or None
<code>animated</code>		bool
<code>antialiased</code> or <code>antialiaseds</code>		Boolean or sequence of booleans
<code>array</code>		ndarray
<code>capstyle</code>		unknown
<code>clim</code>		a length 2 sequence of floats; may be overridden in methods that have <code>vmin</code> and <code>vm</code>
<code>clip_box</code>		a <code>Bbox</code> instance
<code>clip_on</code>		bool
<code>clip_path</code>		<code>[(Path, Transform) Patch None]</code>
<code>cmap</code>		a colormap or registered colormap name
<code>color</code>		matplotlib color arg or sequence of rgba tuples
<code>contains</code>		a callable function
<code>edgecolor</code> or <code>edgecolors</code>		matplotlib color spec or sequence of specs
<code>facecolor</code> or <code>facecolors</code>		matplotlib color spec or sequence of specs
<code>figure</code>		a <code>Figure</code> instance
<code>gid</code>		an id string
<code>hatch</code>		<code>['/' '\' ' ' '-' '+' 'x' 'o' 'O' '.' '*']</code>
<code>joinstyle</code>		unknown

Continued on next page

Table 12 – continued from previous page

	Property	Description
label	object	
linestyle or dashes or linestyles	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ...]	
linewidth or linewidths or lw	float or sequence of floats	
norm	Normalize	
offset_position	['screen' 'data']	
offsets	float or sequence of floats	
path_effects	AbstractPathEffect	
picker	[None bool float callable]	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
transform	Transform	
url	a url string	
urls	List[str] or None	
visible	bool	
zorder	float	

See also:

`fill_between` Fill between two sets of y-values.

Notes

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'where', 'x1', 'x2', 'y'.

75.1.47 matplotlib.pyplot.findobj

`matplotlib.pyplot.findobj(o=None, match=None, include_self=True)`

Find artist objects.

Recursively find all Artist instances contained in self.

match can be

- None: return all objects contained in artist.
- function with signature `boolean = match(artist)` used to filter matches
- class instance: e.g., `Line2D`. Only return artists of class type.

If `include_self` is `True` (default), include self in the list to be checked for a match.

75.1.48 `matplotlib.pyplot.flag`

`matplotlib.pyplot.flag()`

Set the colormap to "flag".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.49 `matplotlib.pyplot.gca`

`matplotlib.pyplot.gca(**kwargs)`

Get the current Axes instance on the current figure matching the given keyword args, or create one.

See also:

`matplotlib.figure.Figure.gca` The figure's `gca` method.

Examples

To get the current polar axes on the current figure:

```
plt.gca(projection='polar')
```

If the current axes doesn't exist, or isn't a polar one, the appropriate axes will be created and then returned.

Examples using `matplotlib.pyplot.gca`

- `sphinx_glr_gallery_pyplots_pyplot_scales.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_multiple_figs_demo.py`
- `sphinx_glr_gallery_images_contours_and_fields_image_annotated_heatmap.py`
- `sphinx_glr_gallery_text_labels_and_annotations_usetex_fonteffects.py`
- `sphinx_glr_gallery_text_labels_and_annotations_text_rotation_relative_to_line.py`
- `sphinx_glr_gallery_text_labels_and_annotations_rainbow_text.py`
- `sphinx_glr_gallery_text_labels_and_annotations_text_alignment.py`
- `sphinx_glr_gallery_text_labels_and_annotations_usetex_demo.py`
- `sphinx_glr_gallery_text_labels_and_annotations_mathtext_examples.py`
- `sphinx_glr_gallery_text_labels_and_annotations_arrow_demo.py`

- [sphx_glr_gallery_axes_grid1_simple_anchored_artists.py](#)
- [sphx_glr_gallery_event_handling_trifinder_event_demo.py](#)
- [sphx_glr_gallery_event_handling_ginput_manual_clabel_sgskip.py](#)
- [sphx_glr_gallery_misc_set_and_get.py](#)
- [sphx_glr_gallery_misc_contour_manual.py](#)
- [sphx_glr_gallery_misc_anchored_artists.py](#)
- [sphx_glr_gallery_scales_symlog_demo.py](#)
- [sphx_glr_gallery_scales_custom_scale.py](#)
- [sphx_glr_gallery_specialty_plots_hinton_demo.py](#)
- [sphx_glr_gallery_specialty_plots_anscombe.py](#)
- [Pyplot tutorial](#)
- [Legend guide](#)
- [Tight Layout guide](#)

75.1.50 `matplotlib.pyplot.gcf`

`matplotlib.pyplot.gcf()`

Get a reference to the current figure.

Examples using `matplotlib.pyplot.gcf`

- [sphx_glr_gallery_text_labels_and_annotations_arrow_demo.py](#)
- [sphx_glr_gallery_event_handling_trifinder_event_demo.py](#)
- [Tight Layout guide](#)

75.1.51 `matplotlib.pyplot.gci`

`matplotlib.pyplot.gci()`

Get the current colorable artist. Specifically, returns the current `ScalarMappable` instance (image or patch collection), or `None` if no images or patch collections have been defined. The commands `imshow()` and `figimage()` create `Image` instances, and the commands `pcolor()` and `scatter()` create `Collection` instances. The current image is an attribute of the current axes, or the nearest earlier axes in the current figure that contains an image.

75.1.52 `matplotlib.pyplot.get_current_fig_manager`

`matplotlib.pyplot.get_current_fig_manager()`

Examples using `matplotlib.pyplot.get_current_fig_manager`

- `sphinx_glr_gallery_misc_agg_buffer.py`
- `sphinx_glr_gallery_user_interfaces_pylab_with_gtk_sgskip.py`

75.1.53 `matplotlib.pyplot.get_figlabels`

`matplotlib.pyplot.get_figlabels()`

Return a list of existing figure labels.

75.1.54 `matplotlib.pyplot.get_fignums`

`matplotlib.pyplot.get_fignums()`

Return a list of existing figure numbers.

75.1.55 `matplotlib.pyplot.get_plot_commands`

`matplotlib.pyplot.get_plot_commands()`

Get a sorted list of all of the plotting commands.

75.1.56 `matplotlib.pyplot.ginput`

`matplotlib.pyplot.ginput(*args, **kwargs)`

Blocking call to interact with a figure.

Wait until the user clicks `n` times on the figure, and return the coordinates of each click in a list.

The buttons used for the various actions (adding points, removing points, terminating the inputs) can be overridden via the arguments `mouse_add`, `mouse_pop` and `mouse_stop`, that give the associated mouse button: 1 for left, 2 for middle, 3 for right.

Parameters

`n` [int, optional, default: 1] Number of mouse clicks to accumulate. If negative, accumulate clicks until the input is terminated manually.

`timeout` [scalar, optional, default: 30] Number of seconds to wait before timing out. If zero or negative will never timeout.

`show_clicks` [bool, optional, default: False] If True, show a red cross at the location of each click.

`mouse_add` [int, one of (1, 2, 3), optional, default: 1 (left click)] Mouse button used to add points.

`mouse_pop` [int, one of (1, 2, 3), optional, default: 3 (right click)] Mouse button used to remove the most recently added point.

`mouse_stop` [int, one of (1, 2, 3), optional, default: 2 (middle click)] Mouse button used to stop input.

Returns

`points` [list of tuples] A list of the clicked (x, y) coordinates.

Notes

The keyboard can also be used to select points in case your mouse does not have one or more of the buttons. The delete and backspace keys act like right clicking (i.e., remove last point), the enter key terminates input and any other key (not already used by the window manager) selects a point.

Examples using `matplotlib.pyplot.ginput`

- `sphinx_glr_gallery_event_handling_ginput_demo_sgskip.py`
- `sphinx_glr_gallery_event_handling_ginput_manual_clabel_sgskip.py`

75.1.57 `matplotlib.pyplot.gray`

`matplotlib.pyplot.gray()`

Set the colormap to "gray".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.58 `matplotlib.pyplot.grid`

`matplotlib.pyplot.grid(b=None, which='major', axis='both', **kwargs)`

Turn the axes grids on or off.

Set the axes grids on or off; `b` is a boolean.

If `b` is `None` and `len(kwargs)==0`, toggle the grid state. If `kwargs` are supplied, it is assumed that you want a grid and `b` is thus set to `True`.

`which` can be 'major' (default), 'minor', or 'both' to control whether major tick grids, minor tick grids, or both are affected.

`axis` can be 'both' (default), 'x', or 'y' to control which set of gridlines are drawn.

`kwargs` are used to set the grid line properties, e.g.,:

```
ax.grid(color='r', linestyle='-', linewidth=2)
```

Valid Line2D `kwargs` are

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) float array
alpha		float (0.0 transparent through 1.0 opaque)
animated		bool
antialiased or aa		bool
clip_box		a Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
color or c		any matplotlib color
contains		a callable function
dash_capstyle		['butt' 'round' 'projecting']
dash_joinstyle		['miter' 'round' 'bevel']
dashes		sequence of on/off ink in points
drawstyle		['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']
figure		a Figure instance
fillstyle		['full' 'left' 'right' 'bottom' 'top' 'none']
gid		an id string
label		object
linestyle or ls		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']
linewidth or lw		float value in points
marker		A valid marker style
markeredgecolor or mec		any matplotlib color
markeredgewidth or mew		float value in points
markerfacecolor or mfc		any matplotlib color
markerfacecoloralt or mfcalt		any matplotlib color
markersize or ms		float
markevery		[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]
path_effects		AbstractPathEffect
picker		float distance in points or callable pick function fn(artist, event)
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
solid_capstyle		['butt' 'round' 'projecting']
solid_joinstyle		['miter' 'round' 'bevel']
transform		a matplotlib.transforms.Transform instance
url		a url string
visible		bool
xdata		1D array
ydata		1D array
zorder		float

Examples using `matplotlib.pyplot.grid`

- [sphx_glr_gallery_api_skewt.py](#)
- [sphx_glr_gallery_pyplots_pyplot_text.py](#)
- [sphx_glr_gallery_pyplots_pyplot_scales.py](#)
- [sphx_glr_gallery_subplots_axes_and_figures_invert_axes.py](#)
- [sphx_glr_gallery_subplots_axes_and_figures_geo_demo.py](#)
- [sphx_glr_gallery_lines_bars_and_markers_nan_test.py](#)
- [sphx_glr_gallery_text_labels_and_annotations_multiline.py](#)
- [sphx_glr_gallery_text_labels_and_annotations_stix_fonts_demo.py](#)
- [sphx_glr_gallery_showcase_bachelors_degrees_by_gender.py](#)
- [sphx_glr_gallery_misc_customize_rc.py](#)
- [sphx_glr_gallery_misc_findobj_demo.py](#)
- [sphx_glr_gallery_misc_custom_projection.py](#)
- [sphx_glr_gallery_scales_symlog_demo.py](#)
- [sphx_glr_gallery_scales_custom_scale.py](#)
- [Pyplot tutorial](#)

75.1.59 `matplotlib.pyplot.hexbin`

```
matplotlib.pyplot.hexbin(x, y, C=None, gridsize=100, bins=None, xscale='linear',
                        yscale='linear', extent=None, cmap=None, norm=None,
                        vmin=None, vmax=None, alpha=None, linewidths=None, edgecol-
                        ors='face', reduce_C_function=<function mean>, mincnt=None,
                        marginals=False, hold=None, data=None, **kwargs)
```

Make a hexagonal binning plot.

Make a hexagonal binning plot of x versus y , where x , y are 1-D sequences of the same length, N . If C is `None` (the default), this is a histogram of the number of occurrences of the observations at $(x[i], y[i])$.

If C is specified, it specifies values at the coordinate $(x[i], y[i])$. These values are accumulated for each hexagonal bin and then reduced according to `reduce_C_function`, which defaults to numpy's mean function (`np.mean`). (If C is specified, it must also be a 1-D sequence of the same length as x and y .)

Parameters

x , y [array or masked array]

C [array or masked array, optional, default is `None`]

gridsize [int or (int, int), optional, default is 100] The number of hexagons in the x-direction, default is 100. The corresponding number of hexagons in the y-direction is chosen such that the hexagons are approximately regular. Alternatively, **gridsize** can be a tuple with two elements specifying the number of hexagons in the x-direction and the y-direction.

bins [{`'log'`} or int or sequence, optional, default is None] If None, no binning is applied; the color of each hexagon directly corresponds to its count value.

If `'log'`, use a logarithmic scale for the color map. Internally, $\log_{10}(i + 1)$ is used to determine the hexagon color.

If an integer, divide the counts in the specified number of bins, and color the hexagons accordingly.

If a sequence of values, the values of the lower bound of the bins to be used.

xscale [{`'linear'`, `'log'`}, optional, default is `'linear'`] Use a linear or log10 scale on the horizontal axis.

yscale [{`'linear'`, `'log'`}, optional, default is `'linear'`] Use a linear or log10 scale on the vertical axis.

mincnt [int > 0, optional, default is None] If not None, only display cells with more than **mincnt** number of points in the cell

marginals [bool, optional, default is False] if **marginals** is True, plot the marginal density as colormapped rectangles along the bottom of the x-axis and left of the y-axis

extent [scalar, optional, default is None] The limits of the bins. The default assigns the limits based on **gridsize**, **x**, **y**, **xscale** and **yscale**.

If **xscale** or **yscale** is set to `'log'`, the limits are expected to be the exponent for a power of 10. E.g. for x-limits of 1 and 50 in `'linear'` scale and y-limits of 10 and 1000 in `'log'` scale, enter (1, 50, 1, 3).

Order of scalars is (left, right, bottom, top).

Returns

object a PolyCollection instance; use `get_array()` on this PolyCollection to get the counts in each hexagon.

If **marginals** is True, horizontal bar and vertical bar (both PolyCollections) will be attached to the return collection as attributes **hbar** and **vbar**.

Other Parameters

cmap [object, optional, default is None] a `matplotlib.colors.Colormap` instance. If None, defaults to `rc image.cmap`.

norm [object, optional, default is None] `matplotlib.colors.Normalize` instance is used to scale luminance data to 0,1.

vmin, **vmax** [scalar, optional, default is None] **vmin** and **vmax** are used in conjunction with **norm** to normalize luminance data. If None, the min and

max of the color array C are used. Note if you pass a norm instance your settings for vmin and vmax will be ignored.

alpha [scalar between 0 and 1, optional, default is None] the alpha value for the patches

linewidths [scalar, optional, default is None] If None, defaults to 1.0.

edgecolors [{ 'face', 'none', None } or color, optional] If 'face' (the default), draws the edges in the same color as the fill color.

If 'none', no edge is drawn; this can sometimes lead to unsightly unpainted pixels between the hexagons.

If None, draws outlines in the default color.

If a matplotlib color arg, draws outlines in the specified color.

Notes

The standard descriptions of all the Collection parameters:

	Property	Description
agg_filter		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a
alpha		float or None
animated		bool
antialiased or antialiaseds		Boolean or sequence of booleans
array		ndarray
capstyle		unknown
clim		a length 2 sequence of floats; may be overridden in methods that have vmin and vm
clip_box		a Bbox instance
clip_on		bool
clip_path		[(Path, Transform) Patch None]
cmap		a colormap or registered colormap name
color		matplotlib color arg or sequence of rgba tuples
contains		a callable function
edgecolor or edgecolors		matplotlib color spec or sequence of specs
facecolor or facecolors		matplotlib color spec or sequence of specs
figure		a Figure instance
gid		an id string
hatch		['/' '\' '.' '-' '+' 'x' 'o' 'O' '?' '*']
joinstyle		unknown
label		object
linestyle or dashes or linestyles		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.'
linewidth or linewidths or lw		float or sequence of floats
norm		Normalize
offset_position		['screen' 'data']
offsets		float or sequence of floats

Continued on next page

Table 14 – continued from previous page

	Property	Description
path_effects		AbstractPathEffect
picker		[None bool float callable]
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
transform		Transform
url		a url string
urls		List[str] or None
visible		bool
zorder		float

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All arguments with the following names: 'x', 'y'.
-

75.1.60 matplotlib.pyplot.hist

matplotlib.pyplot.hist(x, bins=None, range=None, density=None, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, normed=None, hold=None, data=None, **kwargs)

Plot a histogram.

Compute and draw the histogram of x. The return value is a tuple (n, bins, patches) or ([n0, n1, ...], bins, [patches0, patches1,...]) if the input contains multiple data.

Multiple data can be provided via x as a list of datasets of potentially different length ([x0, x1, ...]), or as a 2-D ndarray in which each column is a dataset. Note that the ndarray form is transposed relative to the list form.

Masked arrays are not supported at present.

Parameters

x [(n,) array or sequence of (n,) arrays] Input values, this takes either a single array or a sequence of arrays which are not required to be of the same length

bins [integer or sequence or 'auto', optional] If an integer is given, bins + 1 bin edges are calculated and returned, consistent with [numpy.histogram\(\)](#).

If bins is a sequence, gives bin edges, including left edge of first bin and right edge of last bin. In this case, bins is returned unmodified.

All but the last (righthand-most) bin is half-open. In other words, if bins is:

`[1, 2, 3, 4]`

then the first bin is [1, 2) (including 1, but excluding 2) and the second [2, 3). The last bin, however, is [3, 4], which includes 4.

Unequally spaced bins are supported if bins is a sequence.

If Numpy 1.11 is installed, may also be 'auto'.

Default is taken from the rcParam hist.bins.

range [tuple or None, optional] The lower and upper range of the bins. Lower and upper outliers are ignored. If not provided, range is (x.min(), x.max()). Range has no effect if bins is a sequence.

If bins is a sequence or range is specified, autoscaling is based on the specified bin range instead of the range of x.

Default is None

density [boolean, optional] If True, the first element of the return tuple will be the counts normalized to form a probability density, i.e., the area (or integral) under the histogram will sum to 1. This is achieved by dividing the count by the number of observations times the bin width and not dividing by the total number of observations. If stacked is also True, the sum of the histograms is normalized to 1.

Default is None for both normed and density. If either is set, then that value will be used. If neither are set, then the args will be treated as False.

If both density and normed are set an error is raised.

weights [(n,) array_like or None, optional] An array of weights, of the same shape as x. Each value in x only contributes its associated weight towards the bin count (instead of 1). If normed or density is True, the weights are normalized, so that the integral of the density over the range remains 1.

Default is None

cumulative [boolean, optional] If True, then a histogram is computed where each bin gives the counts in that bin plus all bins for smaller values. The last bin gives the total number of datapoints. If normed or density is also True then the histogram is normalized such that the last bin equals 1. If cumulative evaluates to less than 0 (e.g., -1), the direction of accumulation is reversed. In this case, if normed and/or density is also True, then the histogram is normalized such that the first bin equals 1.

Default is False

bottom [array_like, scalar, or None] Location of the bottom baseline of each bin. If a scalar, the base line for each bin is shifted by the same amount. If an array, each bin is shifted independently and the length of bottom must match the number of bins. If None, defaults to 0.

Default is None

histtype [{'bar', 'barstacked', 'step', 'stepfilled'}, optional] The type of histogram to draw.

- 'bar' is a traditional bar-type histogram. If multiple data are given the bars are arranged side by side.
- 'barstacked' is a bar-type histogram where multiple data are stacked on top of each other.
- 'step' generates a lineplot that is by default unfilled.
- 'stepfilled' generates a lineplot that is by default filled.

Default is 'bar'

align [{'left', 'mid', 'right'}, optional] Controls how the histogram is plotted.

- 'left': bars are centered on the left bin edges.
- 'mid': bars are centered between the bin edges.
- 'right': bars are centered on the right bin edges.

Default is 'mid'

orientation [{'horizontal', 'vertical'}, optional] If 'horizontal', barh will be used for bar-type histograms and the bottom kwarg will be the left edges.

rwidth [scalar or None, optional] The relative width of the bars as a fraction of the bin width. If None, automatically compute the width.

Ignored if histtype is 'step' or 'stepfilled'.

Default is None

log [boolean, optional] If True, the histogram axis will be set to a log scale. If log is True and x is a 1D array, empty bins will be filtered out and only the non-empty (n, bins, patches) will be returned.

Default is False

color [color or array_like of colors or None, optional] Color spec or sequence of color specs, one per dataset. Default (None) uses the standard line color sequence.

Default is None

label [string or None, optional] String, or sequence of strings to match multiple datasets. Bar charts yield multiple patches per dataset, but only the first gets the label, so that the legend command will work as expected.

default is None

`stacked` [boolean, optional] If True, multiple data are stacked on top of each other. If False multiple data are arranged side by side if `histtype` is 'bar' or on top of each other if `histtype` is 'step'.

Default is False

`normed` [bool, optional] Deprecated; use the `density` keyword argument instead.

Returns

`n` [array or list of arrays] The values of the histogram bins. See `normed` or `density` and `weights` for a description of the possible semantics. If input `x` is an array, then this is an array of length `nbins`. If input is a sequence of arrays `[data1, data2,...]`, then this is a list of arrays with the values of the histograms for each of the arrays in the same order.

`bins` [array] The edges of the bins. Length `nbins + 1` (`nbins` left edges and right edge of last bin). Always a single array even when multiple data sets are passed in.

`patches` [list or list of lists] Silent list of individual patches used to create the histogram or list of such list if multiple input datasets.

Other Parameters

`**kwargs` [Patch properties]

See also:

`hist2d` 2D histograms

Notes

Note: In addition to the above described arguments, this function can take a `data` keyword argument. If such a `data` argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'weights', 'x'.
-

Examples using `matplotlib.pyplot.hist`

- `sphx_glr_gallery_pyplots_pyplot_text.py`
- `sphx_glr_gallery_subplots_axes_and_figures_axes_demo.py`
- `sphx_glr_gallery_user_interfaces_svg_histogram_sgskip.py`
- [Image tutorial](#)
- [Pyplot tutorial](#)

75.1.61 matplotlib.pyplot.hist2d

matplotlib.pyplot.hist2d(x, y, bins=10, range=None, normed=False, weights=None, cmin=None, cmax=None, hold=None, data=None, **kwargs)

Make a 2D histogram plot.

Parameters

x, y: array_like, shape (n,) Input values

bins: [None | int | [int, int] | array_like | [array, array]] The bin specification:

- If int, the number of bins for the two dimensions (nx=ny=bins).
- If [int, int], the number of bins in each dimension (nx, ny = bins).
- If array_like, the bin edges for the two dimensions (x_edges=y_edges=bins).
- If [array, array], the bin edges in each dimension (x_edges, y_edges = bins).

The default value is 10.

range [array_like shape(2, 2), optional, default: None] The leftmost and rightmost edges of the bins along each dimension (if not specified explicitly in the bins parameters): [[xmin, xmax], [ymin, ymax]]. All values outside of this range will be considered outliers and not tallied in the histogram.

normed [boolean, optional, default: False] Normalize histogram.

weights [array_like, shape (n,), optional, default: None] An array of values w_i weighing each sample (x_i, y_i).

cmin [scalar, optional, default: None] All bins that has count less than cmin will not be displayed and these count values in the return value count histogram will also be set to nan upon return

cmax [scalar, optional, default: None] All bins that has count more than cmax will not be displayed (set to none before passing to imshow) and these count values in the return value count histogram will also be set to nan upon return

Returns

h [2D array] The bi-dimensional histogram of samples x and y. Values in x are histogrammed along the first dimension and values in y are histogrammed along the second dimension.

xedges [1D array] The bin edges along the x axis.

yedges [1D array] The bin edges along the y axis.

image [AxesImage]

Other Parameters

`cmap` [Colormap or str, optional] A `colors.Colormap` instance. If not set, use rc settings.

`norm` [Normalize, optional] A `colors.Normalize` instance is used to scale luminance data to [0, 1]. If not set, defaults to `colors.Normalize()`.

`vmin/vmax` [None or scalar, optional] Arguments passed to the `Normalize` instance.

`alpha` [0 <= scalar <= 1 or None, optional] The alpha blending value.

See also:

`hist` 1D histogram plotting

Notes

- Currently `hist2d` calculates it's own axis limits, and any limits previously set are ignored.
- Rendering the histogram with a logarithmic color scale is accomplished by passing a `colors.LogNorm` instance to the `norm` keyword argument. Likewise, power-law normalization (similar in effect to gamma correction) can be accomplished with `colors.PowerNorm`.

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: `'weights'`, `'x'`, `'y'`.
-

75.1.62 matplotlib.pyplot.hlines

`matplotlib.pyplot.hlines(y, xmin, xmax, colors='k', linestyle='solid', label="", hold=None, data=None, **kwargs)`

Plot horizontal lines at each `y` from `xmin` to `xmax`.

Parameters

`y` [scalar or sequence of scalar] y-indexes where to plot the lines.

`xmin, xmax` [scalar or 1D array_like] Respective beginning and end of each line. If scalars are provided, all lines will have same length.

`colors` [array_like of colors, optional, default: `'k'`]

`linestyle` [`'solid'` | `'dashed'` | `'dashdot'` | `'dotted'`], optional]

`label` [string, optional, default: `""`]

Returns

lines [LineCollection]

Other Parameters

****kwargs** [LineCollection properties.]

See also:

vlines vertical lines

axhline horizontal line across the axes

In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`: * All arguments with the following names: 'colors', 'xmax', 'xmin', 'y'.

75.1.63 matplotlib.pyplot.hold

`matplotlib.pyplot.hold(b=None)`

Deprecated since version 2.0: `pyplot.hold` is deprecated. Future behavior will be consistent with the long-time default: plot commands add elements without first clearing the Axes and/or Figure.

Set the hold state. If `b` is `None` (default), toggle the hold state, else set the hold state to boolean value `b`:

```
hold()      # toggle hold
hold(True)  # hold is on
hold(False) # hold is off
```

When `hold` is `True`, subsequent plot commands will add elements to the current axes. When `hold` is `False`, the current axes and figure will be cleared on the next plot command.

75.1.64 matplotlib.pyplot.hot

`matplotlib.pyplot.hot()`

Set the colormap to "hot".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.65 matplotlib.pyplot.hsv

`matplotlib.pyplot.hsv()`

Set the colormap to "hsv".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.66 matplotlib.pyplot.imread

`matplotlib.pyplot.imread(*args, **kwargs)`

Read an image from a file into an array.

`fname` may be a string path, a valid URL, or a Python file-like object. If using a file object, it must be opened in binary mode.

If `format` is provided, will try to read file of that type, otherwise the format is deduced from the filename. If nothing can be deduced, PNG is tried.

Return value is a `numpy.array`. For grayscale images, the return array is `MxN`. For RGB images, the return value is `MxNx3`. For RGBA images the return value is `MxNx4`.

matplotlib can only read PNGs natively, but if `PIL` is installed, it will use it to load the image and return an array (if possible) which can be used with `imshow()`. Note, URL strings may not be compatible with PIL. Check the PIL documentation for more information.

Examples using matplotlib.pyplot.imread

- `sphinx_glr_gallery_images_contours_and_fields_image_clip_path.py`
- `sphinx_glr_gallery_images_contours_and_fields_image_demo.py`
- `sphinx_glr_gallery_text_labels_and_annotations_demo_annotation_box.py`

75.1.67 matplotlib.pyplot.imsave

`matplotlib.pyplot.imsave(*args, **kwargs)`

Save an array as in image file.

The output formats available depend on the backend being used.

Parameters

`fname` [str or file-like] Path string to a filename, or a Python file-like object.
If `format` is `None` and `fname` is a string, the output format is deduced from the extension of the filename.

`arr` [array-like] An `MxN` (luminance), `MxNx3` (RGB) or `MxNx4` (RGBA) array.

`vmin, vmax`: [None | scalar] `vmin` and `vmax` set the color scaling for the image by fixing the values that map to the colormap color limits. If either `vmin` or `vmax` is `None`, that limit is determined from the `arr` min/max value.

`cmap` [matplotlib.colors.Colormap, optional] For example, `cm.viridis`. If `None`, defaults to the `image.cmap` rcParam.

`format` [str] One of the file extensions supported by the active backend. Most backends support `png`, `pdf`, `ps`, `eps` and `svg`.

`origin` `[['upper' | 'lower']]` Indicates whether the (0, 0) index of the array is in the upper left or lower left corner of the axes. Defaults to the `image.origin` rcParam.

`dpi` `[int]` The DPI to store in the metadata of the file. This does not affect the resolution of the output image.

75.1.68 matplotlib.pyplot.imshow

```
matplotlib.pyplot.imshow(X, cmap=None, norm=None, aspect=None, interpolation=None,
                        alpha=None, vmin=None, vmax=None, origin=None, extent=None,
                        shape=None, filternorm=1, filterrad=4.0, imlim=None,
                        resample=None, url=None, hold=None, data=None,
                        **kwargs)
```

Display an image on the axes.

Parameters

`X` `[array_like, shape (n, m) or (n, m, 3) or (n, m, 4)]` Display the image in `X` to current axes. `X` may be an array or a PIL image. If `X` is an array, it can have the following shapes and types:

- `MxN` – values to be mapped (float or int)
- `MxNx3` – RGB (float or uint8)
- `MxNx4` – RGBA (float or uint8)

`MxN` arrays are mapped to colors based on the `norm` (mapping scalar to scalar) and the `cmap` (mapping the normed scalar to a color).

Elements of RGB and RGBA arrays represent pixels of an `MxN` image. All values should be in the range `[0 .. 1]` for floats or `[0 .. 255]` for integers. Out-of-range values will be clipped to these bounds.

`cmap` `[Colormap, optional, default: None]` If `None`, default to rc image.cmap value. `cmap` is ignored if `X` is 3-D, directly specifying RGB(A) values.

`aspect` `[['auto' | 'equal' | scalar], optional, default: None]` If `'auto'`, changes the image aspect ratio to match that of the axes.

If `'equal'`, and `extent` is `None`, changes the axes aspect ratio to match that of the image. If `extent` is not `None`, the axes aspect ratio is changed to match that of the extent.

If `None`, default to rc image.aspect value.

`interpolation` `[string, optional, default: None]` Acceptable values are `'none'`, `'nearest'`, `'bilinear'`, `'bicubic'`, `'spline16'`, `'spline36'`, `'hanning'`, `'hamming'`, `'hermite'`, `'kaiser'`, `'quadric'`, `'catrom'`, `'gaussian'`, `'bessel'`, `'mitchell'`, `'sinc'`, `'lanczos'`

If `interpolation` is `None`, default to rc image.interpolation. See also the `filternorm` and `filterrad` parameters. If `interpolation` is `'none'`, then no in-

interpolation is performed on the Agg, ps and pdf backends. Other backends will fall back to 'nearest'.

norm [Normalize, optional, default: None] A Normalize instance is used to scale a 2-D float X input to the (0, 1) range for input to the cmap. If norm is None, use the default func:normalize. If norm is an instance of NoNorm, X must be an array of integers that index directly into the lookup table of the cmap.

vmin, vmax [scalar, optional, default: None] vmin and vmax are used in conjunction with norm to normalize luminance data. Note if you pass a norm instance, your settings for vmin and vmax will be ignored.

alpha [scalar, optional, default: None] The alpha blending value, between 0 (transparent) and 1 (opaque). The alpha argument is ignored for RGBA input data.

origin [['upper' | 'lower'], optional, default: None] Place the [0,0] index of the array in the upper left or lower left corner of the axes. If None, default to rc image.origin.

extent [scalars (left, right, bottom, top), optional, default: None] The location, in data-coordinates, of the lower-left and upper-right corners. If `None`, the image is positioned such that the pixel centers fall on zero-based (row, column) indices.

shape [scalars (columns, rows), optional, default: None] For raw buffer images

filternorm [scalar, optional, default: 1] A parameter for the antigrain image resize filter. From the antigrain documentation, if filternorm = 1, the filter normalizes integer values and corrects the rounding errors. It doesn't do anything with the source floating point values, it corrects only integers according to the rule of 1.0 which means that any sum of pixel weights must be equal to 1.0. So, the filter function must produce a graph of the proper shape.

filterrad [scalar, optional, default: 4.0] The filter radius for filters that have a radius parameter, i.e. when interpolation is one of: 'sinc', 'lanczos' or 'blackman'

Returns

image [AxesImage]

Other Parameters

****kwargs** [Artist properties.]

See also:

matshow Plot a matrix or an array as an image.

Notes

Unless extent is used, pixel centers will be located at integer coordinates. In other words: the origin will coincide with the center of pixel (0, 0).

Two typical representations are used for RGB images with an alpha channel:

- Straight (unassociated) alpha: R, G, and B channels represent the color of the pixel, disregarding its opacity.
- Premultiplied (associated) alpha: R, G, and B channels represent the color of the pixel, adjusted for its opacity by multiplication.

imshow expects RGB images adopting the straight (unassociated) alpha representation.

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All positional and all keyword arguments.
-

Examples using matplotlib.pyplot.imshow

- sphx_glr_gallery_subplots_axes_and_figures_subplots_adjust.py
- sphx_glr_gallery_color_colorbar_basics.py
- sphx_glr_gallery_images_contours_and_fields_image_clip_path.py
- sphx_glr_gallery_images_contours_and_fields_barcode_demo.py
- sphx_glr_gallery_images_contours_and_fields_interpolation_methods.py
- sphx_glr_gallery_images_contours_and_fields_layer_images.py
- sphx_glr_gallery_images_contours_and_fields_affine_image.py
- sphx_glr_gallery_images_contours_and_fields_shading_example.py
- sphx_glr_gallery_images_contours_and_fields_multi_image.py
- sphx_glr_gallery_images_contours_and_fields_image_demo.py
- sphx_glr_gallery_images_contours_and_fields_image_masked.py
- sphx_glr_gallery_images_contours_and_fields_pcolor_demo.py
- sphx_glr_gallery_images_contours_and_fields_image_transparency_blend.py
- sphx_glr_gallery_images_contours_and_fields_contour_image.py
- sphx_glr_gallery_images_contours_and_fields_custom_cmap.py
- sphx_glr_gallery_images_contours_and_fields_image_annotated_heatmap.py
- sphx_glr_gallery_shapes_and_collections_dolphin.py

- [sphx_glr_gallery_showcase_mandelbrot.py](#)
- [Animated image using a precomputed list of images](#)
- [sphx_glr_gallery_misc_hyperlinks_sgskip.py](#)
- [Image tutorial](#)
- [Tight Layout guide](#)

75.1.69 `matplotlib.pyplot.inferno`

`matplotlib.pyplot.inferno()`

Set the colormap to "inferno".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.70 `matplotlib.pyplot.install_repl_displayhook`

`matplotlib.pyplot.install_repl_displayhook()`

Install a repl display hook so that any stale figure are automatically redrawn when control is returned to the repl.

This works with IPython terminals and kernels, as well as vanilla python shells.

75.1.71 `matplotlib.pyplot.ioff`

`matplotlib.pyplot.ioff()`

Turn interactive mode off.

Examples using `matplotlib.pyplot.ioff`

- [Usage Guide](#)

75.1.72 `matplotlib.pyplot.ion`

`matplotlib.pyplot.ion()`

Turn interactive mode on.

75.1.73 `matplotlib.pyplot.ishold`

`matplotlib.pyplot.ishold()`

Deprecated since version 2.0: `pyplot.hold` is deprecated. Future behavior will be consistent with the long-time default: plot commands add elements without first clearing the Axes and/or Figure.

Return the hold status of the current axes.

75.1.74 matplotlib.pyplot.isinteractive

`matplotlib.pyplot.isinteractive()`

Return status of interactive mode.

75.1.75 matplotlib.pyplot.jet

`matplotlib.pyplot.jet()`

Set the colormap to "jet".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.76 matplotlib.pyplot.legend

`matplotlib.pyplot.legend(*args, **kwargs)`

Places a legend on the axes.

Call signatures:

```
legend()
legend(labels)
legend(handles, labels)
```

The call signatures correspond to three different ways how to use this method.

1. Automatic detection of elements to be shown in the legend

The elements to be added to the legend are automatically determined, when you do not pass in any extra arguments.

In this case, the labels are taken from the artist. You can specify them either at artist creation or by calling the `set_label()` method on the artist:

```
line, = ax.plot([1, 2, 3], label='Inline label')
ax.legend()
```

or:

```
line.set_label('Label via method')
line, = ax.plot([1, 2, 3])
ax.legend()
```

Specific lines can be excluded from the automatic legend element selection by defining a label starting with an underscore. This is default for all artists, so calling `Axes.legend` without any arguments and without setting the labels manually will result in no legend being drawn.

2. Labeling existing plot elements

To make a legend for lines which already exist on the axes (via `plot` for instance), simply call this function with an iterable of strings, one for each legend item. For example:

```
ax.plot([1, 2, 3])
ax.legend(['A simple line'])
```

Note: This way of using is discouraged, because the relation between plot elements and labels is only implicit by their order and can easily be mixed up.

3. Explicitly defining the elements in the legend

For full control of which artists have a legend entry, it is possible to pass an iterable of legend artists followed by an iterable of legend labels respectively:

```
legend((line1, line2, line3), ('label1', 'label2', 'label3'))
```

Parameters

`handles` [sequence of Artist, optional] A list of Artists (lines, patches) to be added to the legend. Use this together with labels, if you need full control on what is shown in the legend and the automatic mechanism described above is not sufficient.

The length of handles and labels should be the same in this case. If they are not, they are truncated to the smaller length.

`labels` [sequence of strings, optional] A list of labels to show next to the artists. Use this together with handles, if you need full control on what is shown in the legend and the automatic mechanism described above is not sufficient.

Returns

:class:‘matplotlib.legend.Legend’ instance

Other Parameters

`loc` [int or string or pair of floats, default: ‘upper right’] The location of the legend. Possible codes are:

Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

Alternatively can be a 2-tuple giving x, y of the lower-left corner of the legend in axes coordinates (in which case `bbox_to_anchor` will be ignored).

`bbox_to_anchor` [BboxBase or pair of floats] Specify any arbitrary location for the legend in `bbox_transform` coordinates (default Axes coordinates).

For example, to put the legend's upper right hand corner in the center of the axes the following keywords can be used:

```
loc='upper right', bbox_to_anchor=(0.5, 0.5)
```

`ncol` [integer] The number of columns that the legend has. Default is 1.

`prop` [None or `matplotlib.font_manager.FontProperties` or dict] The font properties of the legend. If None (default), the current `matplotlib.rcParams` will be used.

`fontsize` [int or float or {'xx-small', 'x-small', 'small', 'medium', 'large', 'xx-large', 'xx-large'}] Controls the font size of the legend. If the value is numeric the size will be the absolute font size in points. String values are relative to the current default font size. This argument is only used if `prop` is not specified.

`numpoints` [None or int] The number of marker points in the legend when creating a legend entry for a `Line2D` (line). Default is None, which will take the value from `rcParams["legend.numpoints"]`.

`scatterpoints` [None or int] The number of marker points in the legend when creating a legend entry for a `PathCollection` (scatter plot). Default is None, which will take the value from `rcParams["legend.scatterpoints"]`.

`scatteryoffsets` [iterable of floats] The vertical offset (relative to the font size) for the markers created for a scatter plot legend entry. 0.0 is at the base the legend text, and 1.0 is at the top. To draw all markers at the same height, set to [0.5]. Default is [0.375, 0.5, 0.3125].

`markerscale` [None or int or float] The relative size of legend markers compared with the originally drawn ones. Default is None, which will take the value

`from rcParams["legend.markerscale"]`.

markerfirst [bool] If True, legend marker is placed to the left of the legend label. If False, legend marker is placed to the right of the legend label. Default is True.

frameon [None or bool] Control whether the legend should be drawn on a patch (frame). Default is None, which will take the value from `rcParams["legend.frameon"]`.

fancybox [None or bool] Control whether round edges should be enabled around the FancyBboxPatch which makes up the legend's background. Default is None, which will take the value from `rcParams["legend.fancybox"]`.

shadow [None or bool] Control whether to draw a shadow behind the legend. Default is None, which will take the value from `rcParams["legend.shadow"]`.

framealpha [None or float] Control the alpha transparency of the legend's background. Default is None, which will take the value from `rcParams["legend.framealpha"]`. If shadow is activated and framealpha is None, the default value is ignored.

facecolor [None or "inherit" or a color spec] Control the legend's background color. Default is None, which will take the value from `rcParams["legend.facecolor"]`. If "inherit", it will take `rcParams["axes.facecolor"]`.

edgecolor [None or "inherit" or a color spec] Control the legend's background patch edge color. Default is None, which will take the value from `rcParams["legend.edgecolor"]`. If "inherit", it will take `rcParams["axes.edgecolor"]`.

mode [{"expand", None}] If mode is set to "expand" the legend will be horizontally expanded to fill the axes area (or `bbox_to_anchor` if defines the legend's size).

bbox_transform [None or `matplotlib.transforms.Transform`] The transform for the bounding box (`bbox_to_anchor`). For a value of None (default) the Axes' `transAxes` transform will be used.

title [str or None] The legend's title. Default is no title (None).

borderpad [float or None] The fractional whitespace inside the legend border. Measured in font-size units. Default is None, which will take the value from `rcParams["legend.borderpad"]`.

labelspacing [float or None] The vertical space between the legend entries. Measured in font-size units. Default is None, which will take the value from `rcParams["legend.labelspacing"]`.

handlelength [float or None] The length of the legend handles. Measured in font-size units. Default is None, which will take the value from `rcParams["legend.handlelength"]`.

handletextpad [float or None] The pad between the legend handle and text. Measured in font-size units. Default is None, which will take the value from

`rcParams["legend.handletextpad"]`.

`borderaxespad` [float or None] The pad between the axes and legend border. Measured in font-size units. Default is None, which will take the value from `rcParams["legend.borderaxespad"]`.

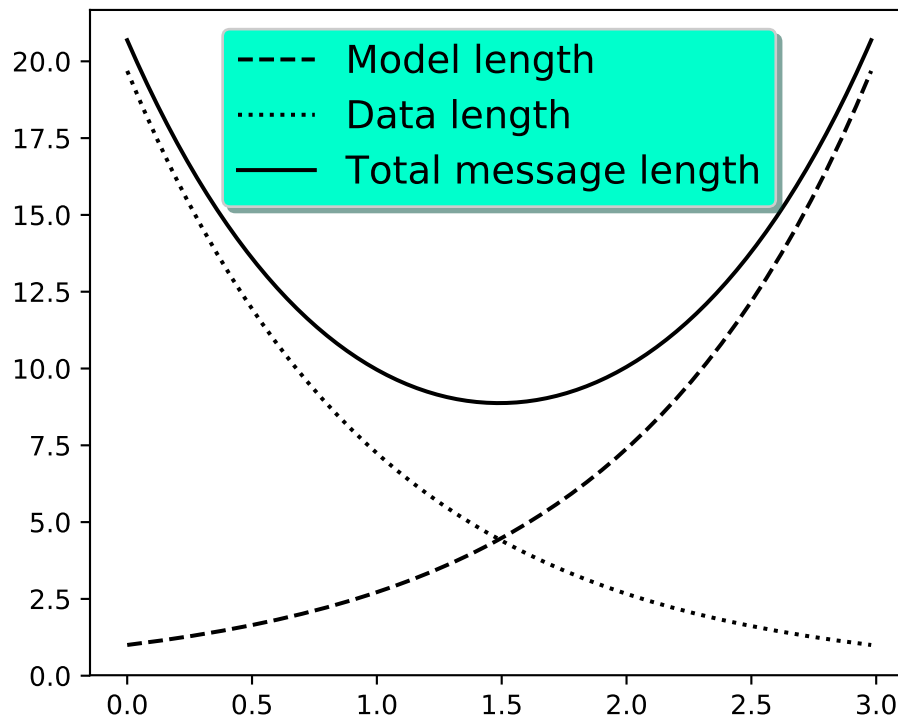
`columnspacing` [float or None] The spacing between columns. Measured in font-size units. Default is None, which will take the value from `rcParams["legend.columnspacing"]`.

`handler_map` [dict or None] The custom dictionary mapping instances or types to a legend handler. This `handler_map` updates the default handler map found at `matplotlib.legend.Legend.get_legend_handler_map()`.

Notes

Not all kinds of artist are supported by the legend command. See [Legend guide](#) for details.

Examples



Examples using `matplotlib.pyplot.legend`

- [sphx_glr_gallery_api_sankey_basics.py](#)
- [sphx_glr_gallery_pyplots_whats_new_98_4_legend.py](#)
- [sphx_glr_gallery_lines_bars_and_markers_scatter_symbol.py](#)
- [sphx_glr_gallery_lines_bars_and_markers_step_demo.py](#)
- [sphx_glr_gallery_lines_bars_and_markers_masked_demo.py](#)
- [sphx_glr_gallery_lines_bars_and_markers_bar_stacked.py](#)
- [sphx_glr_gallery_images_contours_and_fields_contourf_hatching.py](#)
- [sphx_glr_gallery_images_contours_and_fields_contourf_log.py](#)
- [sphx_glr_gallery_text_labels_and_annotations_usetex_demo.py](#)
- [sphx_glr_gallery_pie_and_polar_charts_pie_and_donut_labels.py](#)
- [sphx_glr_gallery_axes_grid1_parasite_simple.py](#)
- [sphx_glr_gallery_misc_findobj_demo.py](#)
- [sphx_glr_gallery_misc_zorder_demo.py](#)
- [sphx_glr_gallery_user_interfaces_pylab_with_gtk_sgskip.py](#)
- [sphx_glr_gallery_user_interfaces_svg_histogram_sgskip.py](#)
- [sphx_glr_gallery_userdemo_simple_legend01.py](#)
- [sphx_glr_gallery_userdemo_pgf_preamble_sgskip.py](#)
- [Usage Guide](#)
- [Legend guide](#)

75.1.77 `matplotlib.pyplot.locator_params`

`matplotlib.pyplot.locator_params(axis='both', tight=None, **kwargs)`

Control behavior of tick locators.

Parameters

`axis` `[['both' | 'x' | 'y'], optional]` The axis on which to operate.

`tight` `[bool or None, optional]` Parameter passed to `autoscale_view()`. Default is `None`, for no change.

Other Parameters

`**kw` : Remaining keyword arguments are passed to directly to the `set_params()` method.

Typically one might want to reduce the maximum number

of ticks and use tight bounds when plotting small subplots, for example:: `ax.locator_params(tight=True, nbins=4)`

Because the locator is involved in autoscaling, `:meth:'autoscale_view'` is called automatically after the parameters are changed.

This presently works only for the `:class:'~matplotlib.ticker.MaxNLocator'` used by default on linear axes, but it may be generalized.

75.1.78 matplotlib.pyplot.loglog

`matplotlib.pyplot.loglog(*args, **kwargs)`

Make a plot with log scaling on both the x and y axis.

Call signatures:

```
loglog([x], y, [fmt], data=None, **kwargs)
loglog([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

This is just a thin wrapper around `plot` which additionally changes both the x-axis and the y-axis to log scaling. All of the concepts and parameters of `plot` can be used here as well.

The additional parameters `basex/y`, `subsx/y` and `nonposx/y` control the x/y-axis properties. They are just forwarded to `Axes.set_xscale` and `Axes.set_yscale`.

Parameters

`basex, basey` [scalar, optional, default 10] Base of the x/y logarithm.

`subsx, subsy` [sequence, optional] The location of the minor x/y ticks. If None, reasonable locations are automatically chosen depending on the number of decades in the plot. See `Axes.set_xscale` / `Axes.set_yscale` for details.

`nonposx, nonposy` [{'mask', 'clip'}, optional, default 'mask'] Non-positive values in x or y can be masked as invalid, or clipped to a very small positive number.

Returns

`lines` A list of `Line2D` objects representing the plotted data.

Other Parameters

`**kwargs` All parameters supported by `plot`.

75.1.79 matplotlib.pyplot.magma

matplotlib.pyplot.magma()

Set the colormap to "magma".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.80 matplotlib.pyplot.magnitude_spectrum

matplotlib.pyplot.magnitude_spectrum(x, Fs=None, Fc=None, window=None,
pad_to=None, sides=None, scale=None,
hold=None, data=None, **kwargs)

Plot the magnitude spectrum.

Call signature:

```
magnitude_spectrum(x, Fs=2, Fc=0, window=mlab.window_hanning,  
pad_to=None, sides='default', **kwargs)
```

Compute the magnitude spectrum of x. Data is padded to a length of pad_to and the windowing function window is applied to the signal.

Parameters

- x [1-D array or sequence] Array or sequence containing the data
- Fs [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, freqs, in cycles per time unit. The default value is 2.
- window [callable or ndarray] A function or a vector of length NFFT. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.
- sides [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.
- pad_to [integer] The number of points to which the data segment is padded when performing the FFT. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the n parameter in the call to `fft()`. The default is None, which sets pad_to equal to the length of the input signal (i.e. no padding).

`scale` `[['default' | 'linear' | 'dB']]` The scaling of the values in the spec. 'linear' is no scaling. 'dB' returns the values in dB scale, i.e., the dB amplitude ($20 * \log_{10}$). 'default' is 'linear'.

`Fc` `[integer]` The center frequency of `x` (defaults to 0), which offsets the `x` extents of the plot to reflect the frequency range used when a signal is acquired and then filtered and downsampled to baseband.

Returns

`spectrum` `[1-D array]` The values for the magnitude spectrum before scaling (real valued)

`freqs` `[1-D array]` The frequencies corresponding to the elements in `spectrum`

`line` `[a Line2D instance]` The line created by this function

Other Parameters

****kwargs** : Keyword arguments control the Line2D properties:

	Property	Description
<code>agg_filter</code>		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)
<code>alpha</code>		float (0.0 transparent through 1.0 opaque)
<code>animated</code>		bool
<code>antialiased</code> or <code>aa</code>		bool
<code>clip_box</code>		a Bbox instance
<code>clip_on</code>		bool
<code>clip_path</code>		<code>[(Path, Transform) Patch None]</code>
<code>color</code> or <code>c</code>		any matplotlib color
<code>contains</code>		a callable function
<code>dash_capstyle</code>		<code>['butt' 'round' 'projecting']</code>
<code>dash_joinstyle</code>		<code>['miter' 'round' 'bevel']</code>
<code>dashes</code>		sequence of on/off ink in points
<code>drawstyle</code>		<code>['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']</code>
<code>figure</code>		a Figure instance
<code>fillstyle</code>		<code>['full' 'left' 'right' 'bottom' 'top' 'none']</code>
<code>gid</code>		an id string
<code>label</code>		object
<code>linestyle</code> or <code>ls</code>		<code>['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']</code>
<code>linewidth</code> or <code>lw</code>		float value in points
<code>marker</code>		A valid marker style
<code>markeredgecolor</code> or <code>mec</code>		any matplotlib color
<code>markeredgewidth</code> or <code>mew</code>		float value in points
<code>markerfacecolor</code> or <code>mfc</code>		any matplotlib color
<code>markerfacecoloralt</code> or <code>mfcalt</code>		any matplotlib color
<code>markersize</code> or <code>ms</code>		float
<code>markevery</code>		<code>[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]</code>
<code>path_effects</code>		AbstractPathEffect

Continued on next page

Table 15 – continued from previous page

	Property	Description
picker	float	distance in points or callable pick function fn(artist, event)
pickradius	float	distance in points
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
solid_capstyle	['butt' 'round' 'projecting']	
solid_joinstyle	['miter' 'round' 'bevel']	
transform	a matplotlib.transforms.Transform instance	
url	a url string	
visible	bool	
xdata	1D array	
ydata	1D array	
zorder	float	

See also:

- psd() psd() plots the power spectral density.
- angle_spectrum() angle_spectrum() plots the angles of the corresponding frequencies.
- phase_spectrum() phase_spectrum() plots the phase (unwrapped angle) of the corresponding frequencies.
- specgram() specgram() can plot the magnitude spectrum of segments within the signal in a colormap.

Notes

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All arguments with the following names: 'x'.

75.1.81 matplotlib.pyplot.margins

matplotlib.pyplot.margins(*args, **kw)
Set or retrieve autoscaling margins.

signatures:

margins()

returns xmargin, ymargin

```

margins(margin)

margins(xmargin, ymargin)

margins(x=xmargin, y=ymargin)

margins(..., tight=False)

```

All three forms above set the `xmargin` and `ymargin` parameters. All keyword parameters are optional. A single argument specifies both `xmargin` and `ymargin`. The padding added to the end of each interval is `margin` times the data interval. The `margin` must be a float in the range `[0, 1]`.

The `tight` parameter is passed to `autoscale_view()`, which is executed after a margin is changed; the default here is `True`, on the assumption that when margins are specified, no additional padding to match tick marks is usually desired. Setting `tight` to `None` will preserve the previous setting.

Specifying any margin changes only the autoscaling; for example, if `xmargin` is not `None`, then `xmargin` times the X data interval will be added to each end of that interval before it is used in autoscaling.

Examples using `matplotlib.pyplot.margins`

- `sphx_glr_gallery_ticks_and_spines_ticklabels_rotation.py`

75.1.82 matplotlib.pyplot.matshow

`matplotlib.pyplot.matshow(A, fignum=None, **kwargs)`

Display an array as a matrix in a new figure window.

The origin is set at the upper left hand corner and rows (first dimension of the array) are displayed horizontally. The aspect ratio of the figure window is that of the array, unless this would make an excessively short or narrow figure.

Tick labels for the xaxis are placed on top.

Parameters

`A` [array-like(M, N)] The matrix to be displayed.

`fignum` [None or int or False] If `None`, create a new figure window with automatic numbering.

If `fignum` is an integer, draw into the figure with the given number (create it if it does not exist).

If 0 or `False`, use the current axes if it exists instead of creating a new figure.

Note: Because of how `Axes.imshow` tries to set the figure aspect ratio to be the one of the array, strange things may happen if you reuse an existing figure.

Returns

image [AxesImage]

Other Parameters

**kwargs [imshow arguments]

Examples using `matplotlib.pyplot.imshow`

- `sphx_glr_gallery_images_contours_and_fields_matshow.py`

75.1.83 `matplotlib.pyplot.minorticks_off`

`matplotlib.pyplot.minorticks_off()`

Remove minor ticks from the current plot.

75.1.84 `matplotlib.pyplot.minorticks_on`

`matplotlib.pyplot.minorticks_on()`

Display minor ticks on the current plot.

Displaying minor ticks reduces performance; turn them off using `minorticks_off()` if drawing speed is a problem.

75.1.85 `matplotlib.pyplot.nipy_spectral`

`matplotlib.pyplot.nipy_spectral()`

Set the colormap to "nipy_spectral".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.86 `matplotlib.pyplot.over`

`matplotlib.pyplot.over(func, *args, **kwargs)`

Deprecated since version 2.0: `pyplot.hold` is deprecated. Future behavior will be consistent with the long-time default: plot commands add elements without first clearing the Axes and/or Figure.

Call a function with `hold(True)`.

Calls:

`func(*args, **kwargs)`

with `hold(True)` and then restores the hold state.

75.1.87 `matplotlib.pyplot.pause`

`matplotlib.pyplot.pause(interval)`

Pause for interval seconds.

If there is an active figure, it will be updated and displayed before the pause, and the GUI event loop (if any) will run during the pause.

This can be used for crude animation. For more complex animation, see `matplotlib.animation`.

Notes

This function is experimental; its behavior may be changed or extended in a future release.

Examples using `matplotlib.pyplot.pause`

- `sphx_glr_gallery_animation_animation_demo.py`
- `sphx_glr_gallery_mplot3d_rotate_axes3d.py`
- `sphx_glr_gallery_mplot3d_wire3d_animation.py`

75.1.88 `matplotlib.pyplot.pcolor`

`matplotlib.pyplot.pcolor(*args, **kwargs)`

Create a pseudocolor plot with a non-regular rectangular grid.

Call signature:

`pcolor([X, Y,] C, **kwargs)`

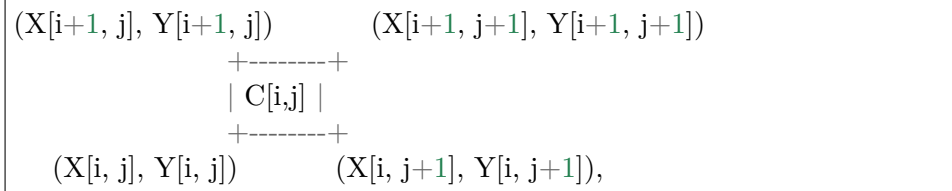
X and Y can be used to specify the corners of the quadrilaterals.

Hint: `pcolor()` can be very slow for large arrays. In most cases you should use the the similar but much faster `pcolormesh` instead. See there for a discussion of the differences.

Parameters

C [array_like] A scalar 2-D array. The values will be color-mapped.

X, Y [array_like, optional] The coordinates of the quadrilateral corners. The quadrilateral for `C[i,j]` has corners at:



Note that the column index corresponds to the x-coordinate, and the row index corresponds to y. For details, see the Notes section below.

The dimensions of X and Y should be one greater than those of C. Alternatively, X, Y and C may have equal dimensions, in which case the last row and column of C will be ignored.

If X and/or Y are 1-D arrays or column vectors they will be expanded as needed into the appropriate 2-D arrays, making a rectangular grid.

cmap [str or Colormap, optional] A Colormap instance or registered colormap name. The colormap maps the C values to colors. Defaults to `rcParams["image.cmap"]`.

norm [Normalize, optional] The Normalize instance scales the data values to the canonical colormap range [0, 1] for mapping to colors. By default, the data range is mapped to the colorbar range using linear scaling.

vmin, vmax [scalar, optional, default: None] The colorbar range. If None, suitable min/max values are automatically chosen by the Normalize instance (defaults to the respective min/max values of C in case of the default linear scaling).

edgecolors [{ 'none', None, 'face', color, color sequence }, optional] The color of the edges. Defaults to 'none'. Possible values:

- 'none' or "": No edge.
- None: `rcParams["patch.edgecolor"]` will be used. Note that currently `rcParams["patch.force_edgecolor"]` has to be True for this to work.
- 'face': Use the adjacent face color.
- An mpl color or sequence of colors will set the edge color.

The singular form `edgecolor` works as an alias.

alpha [scalar, optional, default: None] The alpha blending value of the face color, between 0 (transparent) and 1 (opaque). Note: The edgecolor is currently not affected by this.

snap [bool, optional, default: False] Whether to snap the mesh to pixel boundaries.

Returns

collection [matplotlib.collections.Collection]

Other Parameters

`antialiaseds` [bool, optional, default: False] The default `antialiaseds` is False if the default `edgecolors="none"` is used. This eliminates artificial lines at patch boundaries, and works regardless of the value of `alpha`. If `edgecolors` is not "none", then the default `antialiaseds` is taken from `rcParams["patch.antialiased"]`, which defaults to True. Stroking the edges may be preferred if `alpha` is 1, but will cause artifacts otherwise.

****kwargs** : Additionally, the following arguments are allowed. They are passed along to the `PolyCollection` constructor:

	Property	Description
<code>agg_filter</code>		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) float array
<code>alpha</code>		float or None
<code>animated</code>		bool
<code>antialiased</code> or <code>antialiaseds</code>		Boolean or sequence of booleans
<code>array</code>		ndarray
<code>capstyle</code>		unknown
<code>clim</code>		a length 2 sequence of floats; may be overridden in methods that have <code>vmin</code> and <code>vm</code>
<code>clip_box</code>		a Bbox instance
<code>clip_on</code>		bool
<code>clip_path</code>		[(Path, Transform) Patch None]
<code>cmap</code>		a colormap or registered colormap name
<code>color</code>		matplotlib color arg or sequence of rgba tuples
<code>contains</code>		a callable function
<code>edgecolor</code> or <code>edgecolors</code>		matplotlib color spec or sequence of specs
<code>facecolor</code> or <code>facecolors</code>		matplotlib color spec or sequence of specs
<code>figure</code>		a Figure instance
<code>gid</code>		an id string
<code>hatch</code>		['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']
<code>joinstyle</code>		unknown
<code>label</code>		object
<code>linestyle</code> or <code>dashes</code> or <code>linestyles</code>		['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' '']
<code>linewidth</code> or <code>linewidths</code> or <code>lw</code>		float or sequence of floats
<code>norm</code>		Normalize
<code>offset_position</code>		['screen' 'data']
<code>offsets</code>		float or sequence of floats
<code>path_effects</code>		AbstractPathEffect
<code>picker</code>		[None bool float callable]
<code>pickradius</code>		float distance in points
<code>rasterized</code>		bool or None
<code>sketch_params</code>		(scale: float, length: float, randomness: float)
<code>snap</code>		bool or None
<code>transform</code>		Transform
<code>url</code>		a url string
<code>urls</code>		List[str] or None
<code>visible</code>		bool

Continued on next page

Table 16 – continued from previous page

	Property	Description
zorder	float	

See also:

`pcolormesh` for an explanation of the differences between `pcolor` and `pcolormesh`.

`imshow` If `X` and `Y` are each equidistant, `imshow` can be a faster alternative.

Notes

Masked arrays

`X`, `Y` and `C` may be masked arrays. If either `C[i, j]`, or one of the vertices surrounding `C[i, j]` (`X` or `Y` at `[i, j]`, `[i+1, j]`, `[i, j+1]`, `[i+1, j+1]`) is masked, nothing is plotted.

Grid orientation

The grid orientation follows the standard matrix convention: An array `C` with shape `(nrows, ncolumns)` is plotted with the column number as `X` and the row number as `Y`.

Handling of `pcolor()` end-cases

`pcolor()` displays all columns of `C` if `X` and `Y` are not specified, or if `X` and `Y` have one more column than `C`. If `X` and `Y` have the same number of columns as `C` then the last column of `C` is dropped. Similarly for the rows.

Note: This behavior is different from MATLAB's `pcolor()`, which always discards the last row and column of `C`.

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All positional and all keyword arguments.
-

Examples using `matplotlib.pyplot.pcolor`

- `sphx_glr_gallery_images_contours_and_fields_pcolor_demo.py`

75.1.89 `matplotlib.pyplot.pcolormesh`

`matplotlib.pyplot.pcolormesh(*args, **kwargs)`

Create a pseudocolor plot with a non-regular rectangular grid.

Call signature:

```
pcolor([X, Y,] C, **kwargs)
```

X and Y can be used to specify the corners of the quadrilaterals.

Note: `pcolormesh()` is similar to `pcolor()`. It's much faster and preferred in most cases. For a detailed discussion on the differences see [Differences between `pcolor\(\)` and `pcolormesh\(\)`](#).

Parameters

C [array_like] A scalar 2-D array. The values will be color-mapped.

X, Y [array_like, optional] The coordinates of the quadrilateral corners. The quadrilateral for C[i,j] has corners at:

```
(X[i+1, j], Y[i+1, j])      (X[i+1, j+1], Y[i+1, j+1])
      +-----+
      | C[i,j] |
      +-----+
(X[i, j], Y[i, j])      (X[i, j+1], Y[i, j+1]),
```

Note that the column index corresponds to the x-coordinate, and the row index corresponds to y. For details, see the [Notes](#) section below.

The dimensions of X and Y should be one greater than those of C. Alternatively, X, Y and C may have equal dimensions, in which case the last row and column of C will be ignored.

If X and/or Y are 1-D arrays or column vectors they will be expanded as needed into the appropriate 2-D arrays, making a rectangular grid.

cmap [str or Colormap, optional] A Colormap instance or registered colormap name. The colormap maps the C values to colors. Defaults to `rcParams["image.cmap"]`.

norm [Normalize, optional] The Normalize instance scales the data values to the canonical colormap range [0, 1] for mapping to colors. By default, the data range is mapped to the colorbar range using linear scaling.

vmin, vmax [scalar, optional, default: None] The colorbar range. If None, suitable min/max values are automatically chosen by the Normalize instance (defaults to the respective min/max values of C in case of the default linear scaling).

edgecolors [{ 'none', None, 'face', color, color sequence }, optional] The color of the edges. Defaults to 'none'. Possible values:

- 'none' or '': No edge.
- None: `rcParams["patch.edgecolor"]` will be used. Note that currently `rcParams["patch.force_edgecolor"]` has to be True for this to work.

- 'face': Use the adjacent face color.
- An mpl color or sequence of colors will set the edge color.

The singular form edgecolor works as an alias.

alpha [scalar, optional, default: None] The alpha blending value, between 0 (transparent) and 1 (opaque).

shading [{ 'flat', 'gouraud' }, optional] The fill style, Possible values:

- 'flat': A solid color is used for each quad. The color of the quad (i, j), (i+1, j), (i, j+1), (i+1, j+1) is given by C[i,j].
- 'gouraud': Each quad will be Gouraud shaded: The color of the corners (i', j') are given by C[i',j']. The color values of the area in between is interpolated from the corner values. When Gouraud shading is used, edgecolors is ignored.

snap [bool, optional, default: False] Whether to snap the mesh to pixel boundaries.

Returns

mesh [matplotlib.collections.QuadMesh]

Other Parameters

****kwargs** Additionally, the following arguments are allowed. They are passed along to the QuadMesh constructor:

	<table><tr><th>Property</th><th>Description</th></tr></table>	Property	Description
Property	Description		
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) float array		
alpha	float or None		
animated	bool		
antialiased or antialiaseds	Boolean or sequence of booleans		
array	ndarray		
capstyle	unknown		
clim	a length 2 sequence of floats; may be overridden in methods that have vmin and vmax		
clip_box	a Bbox instance		
clip_on	bool		
clip_path	[(Path, Transform) Patch None]		
cmap	a colormap or registered colormap name		
color	matplotlib color arg or sequence of rgba tuples		
contains	a callable function		
edgecolor or edgecolors	matplotlib color spec or sequence of specs		
facecolor or facecolors	matplotlib color spec or sequence of specs		
figure	a Figure instance		
gid	an id string		
hatch	['/' '\' '.' '-' '+' 'x' 'o' 'O' '?' '*']		
joinstyle	unknown		

Continued on next page

Table 17 – continued from previous page

	Property	Description
label	object	
linestyle or dashes or linestyles	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ...]	
linewidth or linewidths or lw	float or sequence of floats	
norm	Normalize	
offset_position	['screen' 'data']	
offsets	float or sequence of floats	
path_effects	AbstractPathEffect	
picker	[None bool float callable]	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
transform	Transform	
url	a url string	
urls	List[str] or None	
visible	bool	
zorder	float	

See also:

`pcolor` An alternative implementation with slightly different features. For a detailed discussion on the differences see `Differences between pcolor() and pcolormesh()`.

`imshow` If `X` and `Y` are each equidistant, `imshow` can be a faster alternative.

Notes

Masked arrays

`C` may be a masked array. If `C[i, j]` is masked, the corresponding quadrilateral will be transparent. Masking of `X` and `Y` is not supported. Use `pcolor` if you need this functionality.

Grid orientation

The grid orientation follows the standard matrix convention: An array `C` with shape `(nrows, ncolumns)` is plotted with the column number as `X` and the row number as `Y`.

Differences between `pcolor()` and `pcolormesh()`

Both methods are used to create a pseudocolor plot of a 2-D array using quadrilaterals.

The main difference lies in the created object and internal data handling: While `pcolor` returns a `PolyCollection`, `pcolormesh` returns a `QuadMesh`. The latter is more specialized for the given purpose and thus is faster. It should almost always be preferred.

There is also a slight difference in the handling of masked arrays. Both `pcolor` and `pcolormesh` support masked arrays for `C`. However, only `pcolor` supports masked arrays for `X` and `Y`. The reason lies in the internal handling of the masked values. `pcolor` leaves out the respective

polygons from the PolyCollection. `pcolormesh` sets the facecolor of the masked elements to transparent. You can see the difference when using edgecolors. While all edges are drawn irrespective of masking in a QuadMesh, the edge between two adjacent masked quadrilaterals in `pcolor` is not drawn as the corresponding polygons do not exist in the PolyCollection.

Another difference is the support of Gouraud shading in `pcolormesh`, which is not available with `pcolor`.

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All positional and all keyword arguments.
-

Examples using `matplotlib.pyplot.pcolormesh`

- `sphx_glr_gallery_images_contours_and_fields_quadmesh_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_pcolormesh_levels.py`
- `sphx_glr_gallery_images_contours_and_fields_pcolor_demo.py`

75.1.90 `matplotlib.pyplot.phase_spectrum`

`matplotlib.pyplot.phase_spectrum(x, Fs=None, Fc=None, window=None, pad_to=None, sides=None, hold=None, data=None, **kwargs)`

Plot the phase spectrum.

Call signature:

```
phase_spectrum(x, Fs=2, Fc=0, window=mlab.window_hanning,
               pad_to=None, sides='default', **kwargs)
```

Compute the phase spectrum (unwrapped angle spectrum) of `x`. Data is padded to a length of `pad_to` and the windowing function `window` is applied to the signal.

Parameters

- `x` [1-D array or sequence] Array or sequence containing the data
- `Fs` [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.
- `window` [callable or ndarray] A function or a vector of length `NFFT`. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.

`sides` `[['default' | 'onesided' | 'twosided']]` Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. `'onesided'` forces the return of a one-sided spectrum, while `'twosided'` forces two-sided.

`pad_to` `[integer]` The number of points to which the data segment is padded when performing the FFT. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` equal to the length of the input signal (i.e. no padding).

`Fc` `[integer]` The center frequency of `x` (defaults to 0), which offsets the `x` extents of the plot to reflect the frequency range used when a signal is acquired and then filtered and downsampled to baseband.

Returns

`spectrum` `[1-D array]` The values for the phase spectrum in radians (real valued)

`freqs` `[1-D array]` The frequencies corresponding to the elements in `spectrum`

`line` `[a Line2D instance]` The line created by this function

Other Parameters

`**kwargs` : Keyword arguments control the `Line2D` properties:

	Property	Description
<code>agg_filter</code>		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)
<code>alpha</code>		float (0.0 transparent through 1.0 opaque)
<code>animated</code>		bool
<code>antialiased</code> or <code>aa</code>		bool
<code>clip_box</code>		a <code>Bbox</code> instance
<code>clip_on</code>		bool
<code>clip_path</code>		<code>[(Path, Transform) Patch None]</code>
<code>color</code> or <code>c</code>		any matplotlib color
<code>contains</code>		a callable function
<code>dash_capstyle</code>		<code>['butt' 'round' 'projecting']</code>
<code>dash_joinstyle</code>		<code>['miter' 'round' 'bevel']</code>
<code>dashes</code>		sequence of on/off ink in points
<code>drawstyle</code>		<code>['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']</code>
<code>figure</code>		a <code>Figure</code> instance
<code>fillstyle</code>		<code>['full' 'left' 'right' 'bottom' 'top' 'none']</code>
<code>gid</code>		an id string
<code>label</code>		object
<code>linestyle</code> or <code>ls</code>		<code>['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']</code>
<code>linewidth</code> or <code>lw</code>		float value in points
<code>marker</code>		A valid marker style

Continued on next page

Table 18 – continued from previous page

	Property	Description
markeredgecolor or mec		any matplotlib color
markeredgewidth or mew		float value in points
markerfacecolor or mfc		any matplotlib color
markerfacecoloralt or mfcalt		any matplotlib color
markersize or ms		float
markevery		[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]
path_effects		AbstractPathEffect
picker		float distance in points or callable pick function fn(artist, event)
pickradius		float distance in points
rasterized		bool or None
sketch_params		(scale: float, length: float, randomness: float)
snap		bool or None
solid_capstyle		['butt' 'round' 'projecting']
solid_joinstyle		['miter' 'round' 'bevel']
transform		a matplotlib.transforms.Transform instance
url		a url string
visible		bool
xdata		1D array
ydata		1D array
zorder		float

See also:

`magnitude_spectrum()` `magnitude_spectrum()` plots the magnitudes of the corresponding frequencies.

`angle_spectrum()` `angle_spectrum()` plots the wrapped version of this function.

`specgram()` `specgram()` can plot the phase spectrum of segments within the signal in a colormap.

Notes

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'x'.

75.1.91 matplotlib.pyplot.pie

```
matplotlib.pyplot.pie(x, explode=None, labels=None, colors=None, autopct=None, pctdistance=0.6, shadow=False, labeldistance=1.1, startangle=None, radius=None, counterclock=True, wedgeprops=None, textprops=None, center=(0, 0), frame=False, rotatelabels=False, hold=None, data=None)
```

Plot a pie chart.

Make a pie chart of array `x`. The fractional area of each wedge is given by `x/sum(x)`. If `sum(x) < 1`, then the values of `x` give the fractional area directly and the array will not be normalized. The resulting pie will have an empty wedge of size `1 - sum(x)`.

The wedges are plotted counterclockwise, by default starting from the x-axis.

Parameters

- `x` [array-like] The wedge sizes.
- `explode` [array-like, optional, default: None] If not None, is a `len(x)` array which specifies the fraction of the radius with which to offset each wedge.
- `labels` [list, optional, default: None] A sequence of strings providing the labels for each wedge
- `colors` [array-like, optional, default: None] A sequence of matplotlib color args through which the pie chart will cycle. If None, will use the colors in the currently active cycle.
- `autopct` [None (default), string, or function, optional] If not None, is a string or function used to label the wedges with their numeric value. The label will be placed inside the wedge. If it is a format string, the label will be `fmt%pct`. If it is a function, it will be called.
- `pctdistance` [float, optional, default: 0.6] The ratio between the center of each pie slice and the start of the text generated by `autopct`. Ignored if `autopct` is None.
- `shadow` [bool, optional, default: False] Draw a shadow beneath the pie.
- `labeldistance` [float, optional, default: 1.1] The radial distance at which the pie labels are drawn
- `startangle` [float, optional, default: None] If not None, rotates the start of the pie chart by angle degrees counterclockwise from the x-axis.
- `radius` [float, optional, default: None] The radius of the pie, if radius is None it will be set to 1.
- `counterclock` [bool, optional, default: True] Specify fractions direction, clockwise or counterclockwise.
- `wedgeprops` [dict, optional, default: None] Dict of arguments passed to the wedge objects making the pie. For example, you can pass in `wedgeprops = {'linewidth': 3}` to set the width of the wedge border lines equal to 3. For

more details, look at the doc/arguments of the wedge object. By default `clip_on=False`.

`textprops` [dict, optional, default: None] Dict of arguments to pass to the text objects.

`center` [list of float, optional, default: (0, 0)] Center position of the chart. Takes value (0, 0) or is a sequence of 2 scalars.

`frame` [bool, optional, default: False] Plot axes frame with the chart if true.

`rotatelabels` [bool, optional, default: False] Rotate each label to the angle of the corresponding slice if true.

Returns

`patches` [list] A sequence of `matplotlib.patches.Wedge` instances

`texts` [list] A list of the label `matplotlib.text.Text` instances.

`autotexts` [list] A list of `Text` instances for the numeric labels. This will only be returned if the parameter `autopct` is not None.

Notes

The pie chart will probably look best if the figure and axes are square, or the Axes aspect is equal.

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'colors', 'explode', 'labels', 'x'.
-

Examples using `matplotlib.pyplot.pie`

- `sphinx_glr_gallery_pie_and_polar_charts_pie_features.py`
- `sphinx_glr_gallery_pie_and_polar_charts_pie_demo2.py`
- `sphinx_glr_gallery_pie_and_polar_charts_nested_pie.py`
- `sphinx_glr_gallery_pie_and_polar_charts_pie_and_donut_labels.py`

75.1.92 `matplotlib.pyplot.pink`

`matplotlib.pyplot.pink()`

Set the colormap to "pink".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.93 matplotlib.pyplot.plasma

matplotlib.pyplot.plasma()

Set the colormap to "plasma".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.94 matplotlib.pyplot.plot

matplotlib.pyplot.plot(*args, **kwargs)

Plot y versus x as lines and/or markers.

Call signatures:

```
plot([x], y, [fmt], data=None, **kwargs)
plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

The coordinates of the points or line nodes are given by x, y.

The optional parameter `fmt` is a convenient way for defining basic formatting like color, marker and linestyle. It's a shortcut string notation described in the Notes section below.

```
>>> plot(x, y)      # plot x and y using default line style and color
>>> plot(x, y, 'bo') # plot x and y using blue circle markers
>>> plot(y)         # plot y using x as index array 0..N-1
>>> plot(y, 'r+')    # ditto, but with red plusses
```

You can use `Line2D` properties as keyword arguments for more control on the appearance. Line properties and `fmt` can be mixed. The following two calls yield identical results:

```
>>> plot(x, y, 'go--', linewidth=2, markersize=12)
>>> plot(x, y, color='green', marker='o', linestyle='dashed',
        linewidth=2, markersize=12)
```

When conflicting with `fmt`, keyword arguments take precedence.

Plotting labelled data

There's a convenient way for plotting objects with labelled data (i.e. data that can be accessed by index `obj['y']`). Instead of giving the data in x and y, you can provide the object in the `data` parameter and just give the labels for x and y:

```
>>> plot('xlabel', 'ylabel', data=obj)
```

All indexable objects are supported. This could e.g. be a `dict`, a `pandas.DataFrame` or a structured numpy array.

Plotting multiple sets of data

There are various ways to plot multiple sets of data.

- The most straight forward way is just to call plot multiple times. Example:

```
>>> plot(x1, y1, 'bo')
>>> plot(x2, y2, 'go')
```

- Alternatively, if your data is already a 2d array, you can pass it directly to x, y. A separate data set will be drawn for every column.

Example: an array a where the first column represents the x values and the other columns are the y columns:

```
>>> plot(a[0], a[1:])
```

- The third way is to specify multiple sets of [x], y, [fmt] groups:

```
>>> plot(x1, y1, 'g^', x2, y2, 'g-')
```

In this case, any additional keyword argument applies to all datasets. Also this syntax cannot be combined with the data parameter.

By default, each line is assigned a different style specified by a 'style cycle'. The fmt and line property parameters are only necessary if you want explicit deviations from these defaults. Alternatively, you can also change the style cycle using the 'axes.prop_cycle' rcParam.

Parameters

x, y [array-like or scalar] The horizontal / vertical coordinates of the data points. x values are optional. If not given, they default to [0, ..., N-1].

Commonly, these parameters are arrays of length N. However, scalars are supported as well (equivalent to an array with constant value).

The parameters can also be 2-dimensional. Then, the columns represent separate data sets.

fmt [str, optional] A format string, e.g. 'ro' for red circles. See the Notes section for a full description of the format strings.

Format strings are just an abbreviation for quickly setting basic line properties. All of these and more can also be controlled by keyword arguments.

data [indexable object, optional] An object with labelled data. If given, provide the label names to plot in x and y.

Note: Technically there's a slight ambiguity in calls where the second label is a valid fmt. `plot('n', 'o', data=obj)` could be `plt(x, y)` or `plt(y, fmt)`. In such cases, the former interpretation is chosen, but a warning is issued. You may suppress the warning by adding an empty format string `plot('n', 'o', '', data=obj)`.

Returns

lines A list of Line2D objects representing the plotted data.

Other Parameters

scalex, scaley [bool, optional, default: True] These parameters determined if the view limits are adapted to the data limits. The values are passed on to autoscale_view.

**kwargs [Line2D properties, optional] kwargs are used to specify properties like a line label (for auto legends), linewidth, antialiasing, marker face color. Example:

```
>>> plot([1,2,3], [1,2,3], 'go-', label='line 1', linewidth=2)
>>> plot([1,2,3], [1,4,9], 'rs', label='line 2')
```

If you make multiple lines with one plot command, the kwargs apply to all those lines.

Here is a list of available Line2D properties:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)
alpha	float (0.0 transparent through 1.0 opaque)
animated	bool
antialiased or aa	bool
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
color or c	any matplotlib color
contains	a callable function
dash_capstyle	['butt' 'round' 'projecting']
dash_joinstyle	['miter' 'round' 'bevel']
dashes	sequence of on/off ink in points
drawstyle	['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']
figure	a Figure instance
fillstyle	['full' 'left' 'right' 'bottom' 'top' 'none']
gid	an id string
label	object
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' ''
linewidth or lw	float value in points
marker	A valid marker style
markeredgecolor or mec	any matplotlib color
markeredgewidth or mew	float value in points
markerfacecolor or mfc	any matplotlib color
markerfacecoloralt or mfcalt	any matplotlib color
markersize or ms	float
markevery	[None int length-2 tuple of int slice list/array of int float length-2 tuple of
path_effects	AbstractPathEffect

Continued on next page

Table 19 – continued from previous page

	Property	Description
picker	float	distance in points or callable pick function <code>fn(artist, event)</code>
pickradius	float	distance in points
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
solid_capstyle	['butt' 'round' 'projecting']	
solid_joinstyle	['miter' 'round' 'bevel']	
transform	a <code>matplotlib.transforms.Transform</code> instance	
url	a url string	
visible	bool	
xdata	1D array	
ydata	1D array	
zorder	float	

See also:

`scatter` XY scatter plot with markers of varying size and/or color (sometimes also called bubble chart).

Notes

Format Strings

A format string consists of a part for color, marker and line:

```
fmt = '[color][marker][line]'
```

Each of them is optional. If not provided, the value from the style cycle is used. Exception: If line is given, but no marker, the data will be a line without markers.

Colors

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

If the color is the only part of the format string, you can additionally use any `matplotlib.colors` spec, e.g. full names ('green') or hex strings ('#008000').

Markers

character	description
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
' '	vline marker
'_'	hline marker

Line Styles

character	description
'_'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
'::'	dotted line style

Example format strings:

```
'b'   # blue markers with default shape
'ro'  # red circles
'g-'  # green solid line
'--'  # dashed line with default color
'k^:' # black triangle_up markers connected by a dotted line
```

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All arguments with the following names: 'x', 'y'.
-

Examples using `matplotlib.pyplot.plot`

- `sphinx_glr_gallery_pyplots_pyplot_simple.py`
- `sphinx_glr_gallery_pyplots_pyplot_formatstr.py`
- `sphinx_glr_gallery_pyplots_pyplot_three.py`
- `sphinx_glr_gallery_pyplots_whats_new_98_4_legend.py`
- `sphinx_glr_gallery_pyplots_pyplot_two_subplots.py`
- `sphinx_glr_gallery_pyplots_pyplot_mathtext.py`
- `sphinx_glr_gallery_pyplots_pyplot_scales.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_invert_axes.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_multiple_figs_demo.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_subplot.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_shared_axis_demo.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_figure_title.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_axhspan_demo.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_axes_demo.py`
- `sphinx_glr_gallery_color_color_by_yvalue.py`
- `sphinx_glr_gallery_lines_bars_and_markers_arctest.py`
- `sphinx_glr_gallery_lines_bars_and_markers_simple_plot.py`
- `sphinx_glr_gallery_lines_bars_and_markers_masked_demo.py`
- `sphinx_glr_gallery_lines_bars_and_markers_nan_test.py`
- `sphinx_glr_gallery_lines_bars_and_markers_scatter_masked.py`
- `sphinx_glr_gallery_lines_bars_and_markers_psd_demo.py`
- `sphinx_glr_gallery_images_contours_and_fields_triinterp_demo.py`
- `sphinx_glr_gallery_shapes_and_collections_marker_path.py`
- `sphinx_glr_gallery_shapes_and_collections_dolphin.py`
- `sphinx_glr_gallery_text_labels_and_annotations_titles_demo.py`
- `sphinx_glr_gallery_text_labels_and_annotations_text_fontdict.py`
- `sphinx_glr_gallery_text_labels_and_annotations_text_rotation_relative_to_line.py`
- `sphinx_glr_gallery_text_labels_and_annotations_multiline.py`

- sphx_glr_gallery_text_labels_and_annotations_stix_fonts_demo.py
- sphx_glr_gallery_text_labels_and_annotations_usetex_demo.py
- sphx_glr_gallery_style_sheets_plot_solarizedlight2.py
- sphx_glr_gallery_showcase_integral.py
- sphx_glr_gallery_showcase_xkcd.py
- sphx_glr_gallery_showcase_bachelors_degrees_by_gender.py
- Frame grabbing
- sphx_glr_gallery_event_handling_ginput_demo_sgskip.py
- sphx_glr_gallery_misc_print_stdout_sgskip.py
- sphx_glr_gallery_misc_coords_report.py
- sphx_glr_gallery_misc_customize_rc.py
- sphx_glr_gallery_misc_set_and_get.py
- sphx_glr_gallery_misc_agg_buffer.py
- sphx_glr_gallery_misc_findobj_demo.py
- sphx_glr_gallery_misc_zorder_demo.py
- sphx_glr_gallery_misc_transoffset.py
- sphx_glr_gallery_misc_multipage_pdf.py
- sphx_glr_gallery_misc_custom_projection.py
- sphx_glr_gallery_scales_symlog_demo.py
- sphx_glr_gallery_scales_custom_scale.py
- sphx_glr_gallery_specialty_plots_anscombe.py
- sphx_glr_gallery_ticks_and_spines_ticklabels_rotation.py
- sphx_glr_gallery_user_interfaces_pylab_with_gtk_sgskip.py
- sphx_glr_gallery_user_interfaces_toolmanager_sgskip.py
- sphx_glr_gallery_userdemo_simple_legend01.py
- sphx_glr_gallery_userdemo_pgf_texsystem_sgskip.py
- sphx_glr_gallery_userdemo_pgf_fonts_sgskip.py
- sphx_glr_gallery_userdemo_pgf_preamble_sgskip.py
- sphx_glr_gallery_widgets_textbox.py
- sphx_glr_gallery_widgets_buttons.py
- sphx_glr_gallery_widgets_slider_demo.py
- sphx_glr_gallery_widgets_rectangle_selector.py

- [Customizing Matplotlib with style sheets and rcParams](#)
- [Usage Guide](#)
- [Pyplot tutorial](#)
- [Legend guide](#)
- [Path effects guide](#)

75.1.95 `matplotlib.pyplot.plot_date`

`matplotlib.pyplot.plot_date(x, y, fmt='o', tz=None, xdate=True, ydate=False, hold=None, data=None, **kwargs)`
Plot data that contains dates.

Similar to `plot`, this plots `y` vs. `x` as lines or markers. However, the axis labels are formatted as dates depending on `xdate` and `ydate`.

Parameters

- `x, y` [array-like] The coordinates of the data points. If `xdate` or `ydate` is `True`, the respective values `x` or `y` are interpreted as Matplotlib dates.
- `fmt` [str, optional] The plot format string. For details, see the corresponding parameter in `plot`.
- `tz` [[None | timezone string | tzinfo instance]] The time zone to use in labeling dates. If `None`, defaults to `rcParam` `timezone`.
- `xdate` [bool, optional, default: `True`] If `True`, the x-axis will be interpreted as Matplotlib dates.
- `ydate` [bool, optional, default: `False`] If `True`, the y-axis will be interpreted as Matplotlib dates.

Returns

`lines` A list of `Line2D` objects representing the plotted data.

Other Parameters

`**kwargs` Keyword arguments control the `Line2D` properties:

	<table><tr><th>Property</th><th>Description</th></tr></table>	Property	Description
Property	Description		
<code>agg_filter</code>	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)		
<code>alpha</code>	float (0.0 transparent through 1.0 opaque)		
<code>animated</code>	bool		
<code>antialiased</code> or <code>aa</code>	bool		
<code>clip_box</code>	a Bbox instance		
<code>clip_on</code>	bool		
<code>clip_path</code>	[(Path, Transform) Patch None]		
<code>color</code> or <code>c</code>	any matplotlib color		

Continued on next page

Table 20 – continued from previous page

	Property	Description
contains	a callable function	
dash_capstyle	['butt' 'round' 'projecting']	
dash_joinstyle	['miter' 'round' 'bevel']	
dashes	sequence of on/off ink in points	
drawstyle	['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']	
figure	a Figure instance	
fillstyle	['full' 'left' 'right' 'bottom' 'top' 'none']	
gid	an id string	
label	object	
linestyle or ls	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']	
linewidth or lw	float value in points	
marker	A valid marker style	
markeredgecolor or mec	any matplotlib color	
markeredgewidth or mew	float value in points	
markerfacecolor or mfc	any matplotlib color	
markerfacecoloralt or mfcalt	any matplotlib color	
markersize or ms	float	
markevery	[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]	
path_effects	AbstractPathEffect	
picker	float distance in points or callable pick function fn(artist, event)	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
solid_capstyle	['butt' 'round' 'projecting']	
solid_joinstyle	['miter' 'round' 'bevel']	
transform	a matplotlib.transforms.Transform instance	
url	a url string	
visible	bool	
xdata	1D array	
ydata	1D array	
zorder	float	

See also:

matplotlib.dates Helper functions on dates.

matplotlib.dates.date2num Convert dates to num.

matplotlib.dates.num2date Convert num to dates.

matplotlib.dates.drange Create an equally spaced sequence of dates.

Notes

If you are using custom date tickers and formatters, it may be necessary to set the formatters/locators after the call to `plot_date`. `plot_date` will set the default tick locator to `AutoDateLocator` (if the tick locator is not already set to a `DateLocator` instance) and the default tick formatter to `AutoDateFormatter` (if the tick formatter is not already set to a `DateFormatter` instance).

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: `'x'`, `'y'`.
-

Examples using `matplotlib.pyplot.plot_date`

- `sphx_glr_gallery_ticks_and_spines_date_demo_rrule.py`

75.1.96 `matplotlib.pyplot.plotfile`

`matplotlib.pyplot.plotfile(fname, cols=(0,), plotfuncs=None, comments='#', skiprows=0, checkrows=5, delimiter=', ', names=None, subplots=True, newfig=True, **kwargs)`

Plot the data in a file.

`cols` is a sequence of column identifiers to plot. An identifier is either an int or a string. If it is an int, it indicates the column number. If it is a string, it indicates the column header. `matplotlib` will make column headers lower case, replace spaces with underscores, and remove all illegal characters; so `'Adj Close*'` will have name `'adj_close'`.

- If `len(cols) == 1`, only that column will be plotted on the y axis.
- If `len(cols) > 1`, the first element will be an identifier for data for the x axis and the remaining elements will be the column indexes for multiple subplots if `subplots` is `True` (the default), or for lines in a single subplot if `subplots` is `False`.

`plotfuncs`, if not `None`, is a dictionary mapping identifier to an Axes plotting function as a string. Default is `'plot'`, other choices are `'semilogy'`, `'fill'`, `'bar'`, etc. You must use the same type of identifier in the `cols` vector as you use in the `plotfuncs` dictionary, e.g., integer column numbers in both or column names in both. If `subplots` is `False`, then including any function such as `'semilogy'` that changes the axis scaling will set the scaling for all columns.

`comments`, `skiprows`, `checkrows`, `delimiter`, and `names` are all passed on to `matplotlib.pylab.csv2rec()` to load the data into a record array.

If `newfig` is `True`, the plot always will be made in a new figure; if `False`, it will be made in the current figure if one exists, else in a new figure.

`kwargs` are passed on to plotting functions.

Example usage:

```
# plot the 2nd and 4th column against the 1st in two subplots
plotfile(fname, (0,1,3))

# plot using column names; specify an alternate plot type for volume
plotfile(fname, ('date', 'volume', 'adj_close'),
         plotfuncs={'volume': 'semilogy'})
```

Note: plotfile is intended as a convenience for quickly plotting data from flat files; it is not intended as an alternative interface to general plotting with pyplot or matplotlib.

Examples using matplotlib.pyplot.plotfile

- sphx_glr_gallery_misc_plotfile_demo.py

75.1.97 matplotlib.pyplot.polar

matplotlib.pyplot.polar(*args, **kwargs)

Make a polar plot.

call signature:

```
polar(theta, r, **kwargs)
```

Multiple theta, r arguments are supported, with format strings, as in plot().

Examples using matplotlib.pyplot.polar

- sphx_glr_gallery_misc_transoffset.py

75.1.98 matplotlib.pyplot.prism

matplotlib.pyplot.prism()

Set the colormap to "prism".

This changes the default colormap as well as the colormap of the current image if there is one. See help(colormaps) for more information.

75.1.99 matplotlib.pyplot.psd

matplotlib.pyplot.psd(x, NFFT=None, Fs=None, Fc=None, detrend=None, window=None, noverlap=None, pad_to=None, sides=None, scale_by_freq=None, return_line=None, hold=None, data=None, **kwargs)

Plot the power spectral density.

Call signature:

```
psd(x, NFFT=256, Fs=2, Fc=0, detrend=mlab.detrend_none,  
    window=mlab.window_hanning, noverlap=0, pad_to=None,  
    sides='default', scale_by_freq=None, return_line=None, **kwargs)
```

The power spectral density P_{xx} by Welch's average periodogram method. The vector x is divided into NFFT length segments. Each segment is detrended by function `detrend` and windowed by function `window`. `noverlap` gives the length of the overlap between segments. The $|\text{fft}(i)|^2$ of each segment i are averaged to compute P_{xx} , with a scaling to correct for power loss due to windowing.

If $\text{len}(x) < \text{NFFT}$, it will be zero padded to NFFT.

Parameters

- `x` [1-D array or sequence] Array or sequence containing the data
- `Fs` [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.
- `window` [callable or ndarray] A function or a vector of length NFFT. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.
- `sides` [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.
- `pad_to` [integer] The number of points to which the data segment is padded when performing the FFT. This can be different from NFFT, which specifies the number of data points used. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` equal to NFFT
- `NFFT` [integer] The number of data points used in each block for the FFT. A power 2 is most efficient. The default value is 256. This should NOT be used to get zero padding, or the scaling of the result will be incorrect. Use `pad_to` for this instead.
- `detrend` [{ 'default', 'constant', 'mean', 'linear', 'none' } or callable] The function applied to each segment before `fft`-ing, designed to remove the mean or linear trend. Unlike in MATLAB, where the `detrend` parameter is a vector, in matplotlib is it a function. The `pylab` module defines `detrend_none()`, `detrend_mean()`, and `detrend_linear()`, but you can use a custom function as well. You can also use a string to choose one of the functions. 'default',

'constant', and 'mean' call `detrend_mean()`. 'linear' calls `detrend_linear()`. 'none' calls `detrend_none()`.

`scale_by_freq` [boolean, optional] Specifies whether the resulting density values should be scaled by the scaling frequency, which gives density in units of Hz^{-1} . This allows for integration over the returned frequency values. The default is True for MATLAB compatibility.

`noverlap` [integer] The number of points of overlap between segments. The default value is 0 (no overlap).

`Fc` [integer] The center frequency of `x` (defaults to 0), which offsets the `x` extents of the plot to reflect the frequency range used when a signal is acquired and then filtered and downsampled to baseband.

`return_line` [bool] Whether to include the line object plotted in the returned values. Default is False.

Returns

`Pxx` [1-D array] The values for the power spectrum $P_{\{xx\}}$ before scaling (real valued)

`freqs` [1-D array] The frequencies corresponding to the elements in `Pxx`

`line` [a Line2D instance] The line created by this function. Only returned if `return_line` is True.

Other Parameters

****kwargs** : Keyword arguments control the Line2D properties:

	Property Description
<code>agg_filter</code>	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3)
<code>alpha</code>	float (0.0 transparent through 1.0 opaque)
<code>animated</code>	bool
<code>antialiased</code> or <code>aa</code>	bool
<code>clip_box</code>	a Bbox instance
<code>clip_on</code>	bool
<code>clip_path</code>	[(Path, Transform) Patch None]
<code>color</code> or <code>c</code>	any matplotlib color
<code>contains</code>	a callable function
<code>dash_capstyle</code>	['butt' 'round' 'projecting']
<code>dash_joinstyle</code>	['miter' 'round' 'bevel']
<code>dashes</code>	sequence of on/off ink in points
<code>drawstyle</code>	['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']
<code>figure</code>	a Figure instance
<code>fillstyle</code>	['full' 'left' 'right' 'bottom' 'top' 'none']
<code>gid</code>	an id string
<code>label</code>	object
<code>linestyle</code> or <code>ls</code>	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '']

Continued on next page

Table 21 – continued from previous page

	Property	Description
linewidth or lw	float value in points	
marker	A valid marker style	
markeredgecolor or mec	any matplotlib color	
markeredgewidth or mew	float value in points	
markerfacecolor or mfc	any matplotlib color	
markerfacecoloralt or mfcalt	any matplotlib color	
markersize or ms	float	
markevery	[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]	
path_effects	AbstractPathEffect	
picker	float distance in points or callable pick function fn(artist, event)	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
solid_capstyle	['butt' 'round' 'projecting']	
solid_joinstyle	['miter' 'round' 'bevel']	
transform	a matplotlib.transforms.Transform instance	
url	a url string	
visible	bool	
xdata	1D array	
ydata	1D array	
zorder	float	

See also:

`specgram()` `specgram()` differs in the default overlap; in not returning the mean of the segment periodograms; in returning the times of the segments; and in plotting a colormap instead of a line.

`magnitude_spectrum()` `magnitude_spectrum()` plots the magnitude spectrum.

`csd()` `csd()` plots the spectral density between two signals.

Notes

For plotting, the power is plotted as $10 \log_{10}(P_{xx})$ for decibels, though P_{xx} itself is returned.

References

Bendat & Piersol – Random Data: Analysis and Measurement Procedures, John Wiley & Sons (1986)

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by

`data[<arg>]:`

- All arguments with the following names: 'x'.

Examples using `matplotlib.pyplot.psd`

- `sphx_glr_gallery_lines_bars_and_markers_psd_demo.py`

75.1.100 `matplotlib.pyplot.quiver`

`matplotlib.pyplot.quiver(*args, **kw)`

Plot a 2-D field of arrows.

Call signatures:

```
quiver(U, V, **kw)
quiver(U, V, C, **kw)
quiver(X, Y, U, V, **kw)
quiver(X, Y, U, V, C, **kw)
```

`U` and `V` are the arrow data, `X` and `Y` set the location of the arrows, and `C` sets the color of the arrows. These arguments may be 1-D or 2-D arrays or sequences.

If `X` and `Y` are absent, they will be generated as a uniform grid. If `U` and `V` are 2-D arrays and `X` and `Y` are 1-D, and if `len(X)` and `len(Y)` match the column and row dimensions of `U`, then `X` and `Y` will be expanded with `numpy.meshgrid()`.

The default settings auto-scales the length of the arrows to a reasonable size. To change this behavior see the `scale` and `scale_units` kwargs.

The defaults give a slightly swept-back arrow; to make the head a triangle, make `headaxislength` the same as `headlength`. To make the arrow more pointed, reduce `headwidth` or increase `headlength` and `headaxislength`. To make the head smaller relative to the shaft, scale down all the head parameters. You will probably do best to leave `minshaft` alone.

`linewidths` and `edgecolors` can be used to customize the arrow outlines.

Parameters

- `X` [1D or 2D array, sequence, optional] The x coordinates of the arrow locations
- `Y` [1D or 2D array, sequence, optional] The y coordinates of the arrow locations
- `U` [1D or 2D array or masked array, sequence] The x components of the arrow vectors
- `V` [1D or 2D array or masked array, sequence] The y components of the arrow vectors
- `C` [1D or 2D array, sequence, optional] The arrow colors

`units` [['width' | 'height' | 'dots' | 'inches' | 'x' | 'y' | 'xy']] The arrow dimensions (except for length) are measured in multiples of this unit.

'width' or 'height': the width or height of the axis

'dots' or 'inches': pixels or inches, based on the figure dpi

'x', 'y', or 'xy': respectively X, Y, or $\sqrt{X^2 + Y^2}$ in data units

The arrows scale differently depending on the units. For 'x' or 'y', the arrows get larger as one zooms in; for other units, the arrow size is independent of the zoom state. For 'width' or 'height', the arrow size increases with the width and height of the axes, respectively, when the window is resized; for 'dots' or 'inches', resizing does not change the arrows.

`angles` [['uv' | 'xy'], array, optional] Method for determining the angle of the arrows. Default is 'uv'.

'uv': the arrow axis aspect ratio is 1 so that if $U=V$ the orientation of the arrow on the plot is 45 degrees counter-clockwise from the horizontal axis (positive to the right).

'xy': arrows point from (x,y) to (x+u, y+v). Use this for plotting a gradient field, for example.

Alternatively, arbitrary angles may be specified as an array of values in degrees, counter-clockwise from the horizontal axis.

Note: inverting a data axis will correspondingly invert the arrows only with `angles='xy'`.

`scale` [None, float, optional] Number of data units per arrow length unit, e.g., m/s per plot width; a smaller scale parameter makes the arrow longer. Default is None.

If None, a simple autoscaling algorithm is used, based on the average vector length and the number of vectors. The arrow length unit is given by the `scale_units` parameter

`scale_units` [['width' | 'height' | 'dots' | 'inches' | 'x' | 'y' | 'xy'], None, optional] If the scale kwarg is None, the arrow length unit. Default is None.

e.g. `scale_units` is 'inches', scale is 2.0, and $(u,v) = (1,0)$, then the vector will be 0.5 inches long.

If `scale_units` is 'width'/'height', then the vector will be half the width/height of the axes.

If `scale_units` is 'x' then the vector will be 0.5 x-axis units. To plot vectors in the x-y plane, with u and v having the same units as x and y, use `angles='xy', scale_units='xy', scale=1`.

`width` [scalar, optional] Shaft width in arrow units; default depends on choice of units, above, and number of vectors; a typical starting value is about 0.005 times the width of the plot.

`headwidth` [scalar, optional] Head width as multiple of shaft width, default is 3

`headlength` [scalar, optional] Head length as multiple of shaft width, default is 5

`headaxislength` [scalar, optional] Head length at shaft intersection, default is 4.5

`minshaft` [scalar, optional] Length below which arrow scales, in units of head length. Do not set this to less than 1, or small arrows will look terrible! Default is 1

`minlength` [scalar, optional] Minimum length as a multiple of shaft width; if an arrow length is less than this, plot a dot (hexagon) of this diameter instead. Default is 1.

`pivot` [['tail' | 'mid' | 'middle' | 'tip'], optional] The part of the arrow that is at the grid point; the arrow rotates about this point, hence the name pivot.

`color` [[color | color sequence], optional] This is a synonym for the `PolyCollection` `facecolor` kwarg. If `C` has been set, `color` has no effect.

See also:

`quiverkey` Add a key to a quiver plot

Notes

Additional `PolyCollection` keyword arguments:

	Property	Description
<code>agg_filter</code>		a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) float array
<code>alpha</code>		float or None
<code>animated</code>		bool
<code>antialiased</code> or <code>antialiaseds</code>		Boolean or sequence of booleans
<code>array</code>		ndarray
<code>capstyle</code>		unknown
<code>clim</code>		a length 2 sequence of floats; may be overridden in methods that have <code>vmin</code> and <code>vm</code>
<code>clip_box</code>		a <code>Bbox</code> instance
<code>clip_on</code>		bool
<code>clip_path</code>		[(<code>Path</code> , <code>Transform</code>) <code>Patch</code> <code>None</code>]
<code>cmap</code>		a colormap or registered colormap name
<code>color</code>		matplotlib color arg or sequence of rgba tuples
<code>contains</code>		a callable function
<code>edgecolor</code> or <code>edgecolors</code>		matplotlib color spec or sequence of specs
<code>facecolor</code> or <code>facecolors</code>		matplotlib color spec or sequence of specs
<code>figure</code>		a <code>Figure</code> instance
<code>gid</code>		an id string

Continued on next page

Table 22 – continued from previous page

	Property	Description
hatch	['/' '\' ' ' '-' '+' 'x' 'o' 'O' '?' '*']	
joinstyle	unknown	
label	object	
linestyle or dashes or linestyles	['solid' 'dashed', 'dashdot', 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ...]	
linewidth or linewidths or lw	float or sequence of floats	
norm	Normalize	
offset_position	['screen' 'data']	
offsets	float or sequence of floats	
path_effects	AbstractPathEffect	
picker	[None bool float callable]	
pickradius	float distance in points	
rasterized	bool or None	
sketch_params	(scale: float, length: float, randomness: float)	
snap	bool or None	
transform	Transform	
url	a url string	
urls	List[str] or None	
visible	bool	
zorder	float	

Examples using `matplotlib.pyplot.quiver`

- `sphx_glr_gallery_images_contours_and_fields_quiver_simple_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_quiver_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_trigradient_demo.py`

75.1.101 `matplotlib.pyplot.quiverkey`

`matplotlib.pyplot.quiverkey(*args, **kw)`

Add a key to a quiver plot.

Call signature:

```
quiverkey(Q, X, Y, U, label, **kw)
```

Arguments:

Q: The Quiver instance returned by a call to `quiver`.

X, Y: The location of the key; additional explanation follows.

U: The length of the key

label: A string with the length and units of the key

Keyword arguments:

angle = 0 The angle of the key arrow. Measured in degrees anti-clockwise from the x-axis.

coordinates = ['axes' | 'figure' | 'data' | 'inches'] Coordinate system and units for X, Y: 'axes' and 'figure' are normalized coordinate systems with 0,0 in the lower left and 1,1 in the upper right; 'data' are the axes data coordinates (used for the locations of the vectors in the quiver plot itself); 'inches' is position in the figure in inches, with 0,0 at the lower left corner.

color: overrides face and edge colors from Q.

labelpos = ['N' | 'S' | 'E' | 'W'] Position the label above, below, to the right, to the left of the arrow, respectively.

labelsep: Distance in inches between the arrow and the label. Default is 0.1

labelcolor: defaults to default Text color.

fontproperties: A dictionary with keyword arguments accepted by the FontProperties initializer: family, style, variant, size, weight

Any additional keyword arguments are used to override vector properties taken from Q.

The positioning of the key depends on X, Y, coordinates, and labelpos. If labelpos is 'N' or 'S', X, Y give the position of the middle of the key arrow. If labelpos is 'E', X, Y positions the head, and if labelpos is 'W', X, Y positions the tail; in either of these two cases, X, Y is somewhere in the middle of the arrow+label key object.

Examples using `matplotlib.pyplot.quiverkey`

- `sphx_glr_gallery_images_contours_and_fields_quiver_simple_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_quiver_demo.py`

75.1.102 `matplotlib.pyplot.rc`

`matplotlib.pyplot.rc(*args, **kwargs)`

Set the current rc params. Group is the grouping for the rc, e.g., for `lines.linewidth` the group is `lines`, for `axes.facecolor`, the group is `axes`, and so on. Group may also be a list or tuple of group names, e.g., `(xtick, ytick)`. kwargs is a dictionary attribute name/value pairs, e.g.,:

```
rc('lines', linewidth=2, color='r')
```

sets the current rc params and is equivalent to:

```
rcParams['lines.linewidth'] = 2
rcParams['lines.color'] = 'r'
```

The following aliases are available to save typing for interactive users:

Alias	Property
'lw'	'linewidth'
'ls'	'linestyle'
'c'	'color'
'fc'	'facecolor'
'ec'	'edgecolor'
'mew'	'markeredgewidth'
'aa'	'antialiased'

Thus you could abbreviate the above rc command as:

```
rc('lines', lw=2, c='r')
```

Note you can use python's kwargs dictionary facility to store dictionaries of default parameters. e.g., you can customize the font rc as follows:

```
font = {'family' : 'monospace',
        'weight' : 'bold',
        'size'   : 'larger'}

rc('font', **font) # pass in the font dict as kwargs
```

This enables you to easily switch between several configurations. Use `matplotlib.style.use('default')` or `rcdefaults()` to restore the default rc params after changes.

Examples using `matplotlib.pyplot.rc`

- `sphx_glr_gallery_color_color_cycler.py`
- `sphx_glr_gallery_text_labels_and_annotations_usetex_demo.py`
- `sphx_glr_gallery_pie_and_polar_charts_polar_legend.py`
- `sphx_glr_gallery_misc_customize_rc.py`
- `sphx_glr_gallery_misc_multipage_pdf.py`
- [Styling with cycler](#)

75.1.103 `matplotlib.pyplot.rc_context`

`matplotlib.pyplot.rc_context(rc=None, fname=None)`
Return a context manager for managing rc settings.

This allows one to do:

```
with mpl.rc_context(fname='screen.rc'):
    plt.plot(x, a)
    with mpl.rc_context(fname='print.rc'):
        plt.plot(x, b)
    plt.plot(x, c)
```

The 'a' vs 'x' and 'c' vs 'x' plots would have settings from 'screen.rc', while the 'b' vs 'x' plot would have settings from 'print.rc'.

A dictionary can also be passed to the context manager:

```
with mpl.rc_context(rc={'text.usetex': True}, fname='screen.rc'):
    plt.plot(x, a)
```

The 'rc' dictionary takes precedence over the settings loaded from 'fname'. Passing a dictionary only is also valid. For example a common usage is:

```
with mpl.rc_context(rc={'interactive': False}):
    fig, ax = plt.subplots()
    ax.plot(range(3), range(3))
    fig.savefig('A.png', format='png')
    plt.close(fig)
```

Examples using `matplotlib.pyplot.rc_context`

- `sphinx_glr_gallery_ticks_and_spines_auto_ticks.py`

75.1.104 `matplotlib.pyplot.rcdefaults`

`matplotlib.pyplot.rcdefaults()`

Restore the rc params from Matplotlib's internal defaults.

See also:

`rc_file_defaults` Restore the rc params from the rc file originally loaded by Matplotlib.

`matplotlib.style.use` Use a specific style file. Call `style.use('default')` to restore the default style.

Examples using `matplotlib.pyplot.rcdefaults`

- `sphinx_glr_gallery_lines_bars_and_markers_barh.py`
- `sphinx_glr_gallery_misc_customize_rc.py`

75.1.105 matplotlib.pyplot.rgrids

`matplotlib.pyplot.rgrids(*args, **kwargs)`

Get or set the radial gridlines on a polar plot.

call signatures:

```
lines, labels = rgrids()
lines, labels = rgrids(radial, labels=None, angle=22.5, **kwargs)
```

When called with no arguments, `rgrid()` simply returns the tuple (lines, labels), where lines is an array of radial gridlines (Line2D instances) and labels is an array of tick labels (Text instances). When called with arguments, the labels will appear at the specified radial distances and angles.

labels, if not None, is a len(radial) list of strings of the labels to use at each angle.

If labels is None, the rformatter will be used

Examples:

```
# set the locations of the radial gridlines and labels
lines, labels = rgrids( (0.25, 0.5, 1.0) )

# set the locations and labels of the radial gridlines and labels
lines, labels = rgrids( (0.25, 0.5, 1.0), ('Tom', 'Dick', 'Harry' )
```

75.1.106 matplotlib.pyplot.savefig

`matplotlib.pyplot.savefig(*args, **kwargs)`

Save the current figure.

Call signature:

```
savefig(fname, dpi=None, facecolor='w', edgecolor='w',
        orientation='portrait', papertype=None, format=None,
        transparent=False, bbox_inches=None, pad_inches=0.1,
        frameon=None)
```

The output formats available depend on the backend being used.

Parameters

`fname` [str or file-like object] A string containing a path to a filename, or a Python file-like object, or possibly some backend-dependent object such as PdfPages.

If format is None and fname is a string, the output format is deduced from the extension of the filename. If the filename has no extension, the value of the rc parameter `savefig.format` is used.

If `fname` is not a string, remember to specify format to ensure that the correct backend is used.

Other Parameters

`dpi` `[[None | scalar > 0 | 'figure']]` The resolution in dots per inch. If `None` it will default to the value `savefig.dpi` in the `matplotlibrc` file. If `'figure'` it will set the `dpi` to be the value of the figure.

`facecolor` `[color spec or None, optional]` the facecolor of the figure; if `None`, defaults to `savefig.facecolor`

`edgecolor` `[color spec or None, optional]` the edgecolor of the figure; if `None`, defaults to `savefig.edgecolor`

`orientation` `[{'landscape', 'portrait'}]` not supported on all backends; currently only on postscript output

`papertype` `[str]` One of `'letter'`, `'legal'`, `'executive'`, `'ledger'`, `'a0'` through `'a10'`, `'b0'` through `'b10'`. Only supported for postscript output.

`format` `[str]` One of the file extensions supported by the active backend. Most backends support `png`, `pdf`, `ps`, `eps` and `svg`.

`transparent` `[bool]` If `True`, the axes patches will all be transparent; the figure patch will also be transparent unless `facecolor` and/or `edgecolor` are specified via `kwargs`. This is useful, for example, for displaying a plot on top of a colored background on a web page. The transparency of these patches will be restored to their original values upon exit of this function.

`frameon` `[bool]` If `True`, the figure patch will be colored, if `False`, the figure background will be transparent. If not provided, the `rcParam` `'savefig.frameon'` will be used.

`bbox_inches` `[str or Bbox, optional]` Bbox in inches. Only the given portion of the figure is saved. If `'tight'`, try to figure out the tight bbox of the figure. If `None`, use `savefig.bbox`

`pad_inches` `[scalar, optional]` Amount of padding around the figure when `bbox_inches` is `'tight'`. If `None`, use `savefig.pad_inches`

`bbox_extra_artists` `[list of Artist, optional]` A list of extra artists that will be considered when the tight bbox is calculated.

Examples using `matplotlib.pyplot.savefig`

- `sphx_glr_gallery_text_labels_and_annotations_usetex_fonteffects.py`
- `sphx_glr_gallery_text_labels_and_annotations_stix_fonts_demo.py`
- `sphx_glr_gallery_misc_print_stdout_sgskip.py`
- `sphx_glr_gallery_misc_tight_bbox_test.py`
- `sphx_glr_gallery_misc_rasterization_demo.py`

- sphx_glr_gallery_misc_svg_filter_line.py
- sphx_glr_gallery_misc_svg_filter_pie.py
- sphx_glr_gallery_user_interfaces_svg_tooltip_sgskip.py
- sphx_glr_gallery_user_interfaces_svg_histogram_sgskip.py
- sphx_glr_gallery_userdemo_pgf_texsystem_sgskip.py
- sphx_glr_gallery_userdemo_pgf_fonts_sgskip.py
- sphx_glr_gallery_userdemo_pgf_preamble_sgskip.py

75.1.107 matplotlib.pyplot.sca

matplotlib.pyplot.sca(ax)

Set the current Axes instance to ax.

The current Figure is updated to the parent of ax.

75.1.108 matplotlib.pyplot.scatter

matplotlib.pyplot.scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, verts=None, edgecolors=None, hold=None, data=None, **kwargs)

A scatter plot of y vs x with varying marker size and/or color.

Parameters

- x, y [array_like, shape (n,)] The data positions.
- s [scalar or array_like, shape (n,), optional] The marker size in points**2. Default is rcParams['lines.markersize'] ** 2.
- c [color, sequence, or sequence of color, optional, default: 'b'] The marker color. Possible values:
- A single color format string.
 - A sequence of color specifications of length n.
 - A sequence of n numbers to be mapped to colors using cmap and norm.
 - A 2-D array in which the rows are RGB or RGBA.

Note that c should not be a single numeric RGB or RGBA sequence because that is indistinguishable from an array of values to be colormapped. If you want to specify the same RGB or RGBA value for all points, use a 2-D array with a single row.

- marker [MarkerStyle, optional, default: 'o'] The marker style. marker can be either an instance of the class or the text shorthand for a particular marker. See markers for more information marker styles.

`cmap` [Colormap, optional, default: None] A Colormap instance or registered colormap name. `cmap` is only used if `c` is an array of floats. If None, defaults to `rc image.cmap`.

`norm` [Normalize, optional, default: None] A Normalize instance is used to scale luminance data to 0, 1. `norm` is only used if `c` is an array of floats. If None, use the default `colors.Normalize`.

`vmin`, `vmax` [scalar, optional, default: None] `vmin` and `vmax` are used in conjunction with `norm` to normalize luminance data. If None, the respective min and max of the color array is used. `vmin` and `vmax` are ignored if you pass a `norm` instance.

`alpha` [scalar, optional, default: None] The alpha blending value, between 0 (transparent) and 1 (opaque).

`linewidths` [scalar or array_like, optional, default: None] The linewidth of the marker edges. Note: The default `edgecolors` is 'face'. You may want to change this as well. If None, defaults to `rcParams lines.linewidth`.

`verts` [sequence of (x, y), optional] If `marker` is None, these vertices will be used to construct the marker. The center of the marker is located at (0, 0) in normalized units. The overall marker is rescaled by `s`.

`edgecolors` [color or sequence of color, optional, default: 'face'] The edge color of the marker. Possible values:

- 'face': The edge color will always be the same as the face color.
- 'none': No patch boundary will be drawn.
- A matplotlib color.

For non-filled markers, the `edgecolors` kwarg is ignored and forced to 'face' internally.

Returns

`paths` [PathCollection]

Other Parameters

`**kwargs` [Collection properties]

See also:

`plot` To plot scatter plots when markers are identical in size and color.

Notes

- The `plot` function will be faster for scatterplots where markers don't vary in size or color.
- Any or all of `x`, `y`, `s`, and `c` may be masked arrays, in which case all masks will be combined and only unmasked points will be plotted.

- Fundamentally, scatter works with 1-D arrays; x, y, s, and c may be input as 2-D arrays, but within scatter they will be flattened. The exception is c, which will be flattened only if its size matches the size of x and y.

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by data[<arg>]:

- All arguments with the following names: 'c', 'color', 'edgecolors', 'facecolor', 'facecolors', 'linewidths', 's', 'x', 'y'.
-

Examples using `matplotlib.pyplot.scatter`

- [sphx_glr_gallery_lines_bars_and_markers_scatter_symbol.py](#)
- [sphx_glr_gallery_lines_bars_and_markers_scatter_masked.py](#)
- [sphx_glr_gallery_lines_bars_and_markers_scatter_star_poly.py](#)
- [sphx_glr_gallery_shapes_and_collections_scatter.py](#)
- [sphx_glr_gallery_pie_and_polar_charts_polar_scatter.py](#)
- [sphx_glr_gallery_misc_hyperlinks_sgskip.py](#)
- [sphx_glr_gallery_misc_zorder_demo.py](#)
- [Pyplot tutorial](#)

75.1.109 `matplotlib.pyplot.sci`

`matplotlib.pyplot.sci(im)`

Set the current image. This image will be the target of colormap commands like `jet()`, `hot()` or `clim()`. The current image is an attribute of the current axes.

Examples using `matplotlib.pyplot.sci`

- [sphx_glr_gallery_shapes_and_collections_line_collection.py](#)

75.1.110 `matplotlib.pyplot.semilogx`

`matplotlib.pyplot.semilogx(*args, **kwargs)`

Make a plot with log scaling on the x axis.

Call signatures:

```
semilogx([x], y, [fmt], data=None, **kwargs)
semilogx([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

This is just a thin wrapper around `plot` which additionally changes the x-axis to log scaling. All of the concepts and parameters of `plot` can be used here as well.

The additional parameters `basex`, `subsx` and `nonposx` control the x-axis properties. They are just forwarded to `Axes.set_xscale`.

Parameters

`basex` [scalar, optional, default 10] Base of the x logarithm.

`subsx` [array_like, optional] The location of the minor xticks. If `None`, reasonable locations are automatically chosen depending on the number of decades in the plot. See `Axes.set_xscale` for details.

`nonposx` [{`'mask'`, `'clip'`}, optional, default `'mask'`] Non-positive values in x can be masked as invalid, or clipped to a very small positive number.

Returns

`lines` A list of `Line2D` objects representing the plotted data.

Other Parameters

`**kwargs` All parameters supported by `plot`.

75.1.111 matplotlib.pyplot.semilogy

`matplotlib.pyplot.semilogy(*args, **kwargs)`

Make a plot with log scaling on the y axis.

Call signatures:

```
semilogy([x], y, [fmt], data=None, **kwargs)
semilogy([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

This is just a thin wrapper around `plot` which additionally changes the y-axis to log scaling. All of the concepts and parameters of `plot` can be used here as well.

The additional parameters `basey`, `subsy` and `nonposy` control the y-axis properties. They are just forwarded to `Axes.set_yscale`.

Parameters

`basey` [scalar, optional, default 10] Base of the y logarithm.

`subsy` [array_like, optional] The location of the minor yticks. If `None`, reasonable locations are automatically chosen depending on the number of decades in the plot. See `Axes.set_yscale` for details.

`nonposy` [{`'mask'`, `'clip'`}, optional, default `'mask'`] Non-positive values in y can be masked as invalid, or clipped to a very small positive number.

Returns

lines A list of Line2D objects representing the plotted data.

Other Parameters

**kwargs All parameters supported by plot.

75.1.112 matplotlib.pyplot.set_cmap

matplotlib.pyplot.set_cmap(cmap)

Set the default colormap. Applies to the current image if any. See `help(colormaps)` for more information.

cmap must be a Colormap instance, or the name of a registered colormap.

See `matplotlib.cm.register_cmap()` and `matplotlib.cm.get_cmap()`.

75.1.113 matplotlib.pyplot.setp

matplotlib.pyplot.setp(*args, **kwargs)

Set a property on an artist object.

matplotlib supports the use of `setp()` ("set property") and `getp()` to set and get object properties, as well as to do introspection on the object. For example, to set the linestyle of a line to be dashed, you can do:

```
>>> line, = plot([1,2,3])
>>> setp(line, linestyle='--')
```

If you want to know the valid types of arguments, you can provide the name of the property you want to set without a value:

```
>>> setp(line, 'linestyle')
linestyle: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' ]
```

If you want to see all the properties that can be set, and their possible values, you can do:

```
>>> setp(line)
... long output listing omitted
```

You may specify another output file to `setp` if `sys.stdout` is not acceptable for some reason using the file keyword-only argument:

```
>>> with fopen('output.log') as f:
>>>     setp(line, file=f)
```

`setp()` operates on a single instance or a iterable of instances. If you are in query mode introspecting the possible values, only the first instance in the sequence is used. When

actually setting values, all the instances will be set. e.g., suppose you have a list of two lines, the following will make both lines thicker and red:

```
>>> x = arange(0,1.0,0.01)
>>> y1 = sin(2*pi*x)
>>> y2 = sin(4*pi*x)
>>> lines = plot(x, y1, x, y2)
>>> setp(lines, linewidth=2, color='r')
```

setp() works with the MATLAB style string/value pairs or with python kwargs. For example, the following are equivalent:

```
>>> setp(lines, 'linewidth', 2, 'color', 'r') # MATLAB style
>>> setp(lines, linewidth=2, color='r')      # python style
```

Examples using matplotlib.pyplot.setp

- sphx_glr_gallery_subplots_axes_and_figures_shared_axis_demo.py
- sphx_glr_gallery_statistics_boxplot_vs_violin.py
- sphx_glr_gallery_statistics_boxplot_demo.py
- sphx_glr_gallery_lines_bars_and_markers_arctest.py
- sphx_glr_gallery_lines_bars_and_markers_stem_plot.py
- sphx_glr_gallery_lines_bars_and_markers_masked_demo.py
- sphx_glr_gallery_images_contours_and_fields_contour_demo.py
- sphx_glr_gallery_images_contours_and_fields_image_annotated_heatmap.py
- sphx_glr_gallery_text_labels_and_annotations_usetex_fonteffects.py
- sphx_glr_gallery_pie_and_polar_charts_pie_and_donut_labels.py
- sphx_glr_gallery_axes_grid1_inset_locator_demo2.py
- sphx_glr_gallery_axes_grid1_demo_axes_divider.py
- sphx_glr_gallery_event_handling_ginput_manual_clabel_sgskip.py
- sphx_glr_gallery_misc_set_and_get.py
- sphx_glr_gallery_misc_patheffect_demo.py
- sphx_glr_gallery_specialty_plots_anscombe.py
- sphx_glr_gallery_specialty_plots_topographic_hillshading.py
- sphx_glr_gallery_units_evans_test.py
- [The Lifecycle of a Plot](#)

75.1.114 matplotlib.pyplot.show

`matplotlib.pyplot.show(*args, **kw)`

Display a figure. When running in ipython with its pylab mode, display all figures and return to the ipython prompt.

In non-interactive mode, display all figures and block until the figures have been closed; in interactive mode it has no effect unless figures were created prior to a change from non-interactive to interactive mode (not recommended). In that case it displays the figures but does not block.

A single experimental keyword argument, `block`, may be set to `True` or `False` to override the blocking behavior described above.

Examples using matplotlib.pyplot.show

- `sphx_glr_gallery_api_font_family_rc_sgskip.py`
- `sphx_glr_gallery_api_unicode_minus.py`
- `sphx_glr_gallery_api_font_file.py`
- `sphx_glr_gallery_api_watermark_text.py`
- `sphx_glr_gallery_api_quad_bezier.py`
- `sphx_glr_gallery_api_legend.py`
- `sphx_glr_gallery_api_watermark_image.py`
- `sphx_glr_gallery_api_image_zcoord.py`
- `sphx_glr_gallery_api_mathtext_asarray.py`
- `sphx_glr_gallery_api_span_regions.py`
- `sphx_glr_gallery_api_two_scales.py`
- `sphx_glr_gallery_api_power_norm.py`
- `sphx_glr_gallery_api_joinstyle.py`
- `sphx_glr_gallery_api_histogram_path.py`
- `sphx_glr_gallery_api_scatter_piecharts.py`
- `sphx_glr_gallery_api_patch_collection.py`
- `sphx_glr_gallery_api_sankey_links.py`
- `sphx_glr_gallery_api_line_with_text.py`
- `sphx_glr_gallery_api_sankey_basics.py`
- `sphx_glr_gallery_api_logos2.py`
- `sphx_glr_gallery_api_sankey_rankine.py`

- sphx_glr_gallery_api_filled_step.py
- sphx_glr_gallery_api_radar_chart.py
- sphx_glr_gallery_api_skewt.py
- sphx_glr_gallery_pyplots_pyplot_simple.py
- sphx_glr_gallery_pyplots_pyplot_formatstr.py
- sphx_glr_gallery_pyplots_pyplot_three.py
- sphx_glr_gallery_pyplots_fig_x.py
- sphx_glr_gallery_pyplots_whats_new_98_4_legend.py
- sphx_glr_gallery_pyplots_pyplot_two_subplots.py
- sphx_glr_gallery_pyplots_annotation_basic.py
- sphx_glr_gallery_pyplots_whats_new_99_mplot3d.py
- sphx_glr_gallery_pyplots_whats_new_98_4_fill_between.py
- sphx_glr_gallery_pyplots_pyplot_mathtext.py
- sphx_glr_gallery_pyplots_dollar_ticks.py
- sphx_glr_gallery_pyplots_pyplot_text.py
- sphx_glr_gallery_pyplots_fig_axes_labels_simple.py
- sphx_glr_gallery_pyplots_fig_axes_customize_simple.py
- sphx_glr_gallery_pyplots_annotation_polar.py
- sphx_glr_gallery_pyplots_align_ylabel.py
- sphx_glr_gallery_pyplots_whats_new_99_axes_grid.py
- sphx_glr_gallery_pyplots_text_commands.py
- sphx_glr_gallery_pyplots_whats_new_1_subplot3d.py
- sphx_glr_gallery_pyplots_auto_subplots_adjust.py
- sphx_glr_gallery_pyplots_annotate_transform.py
- sphx_glr_gallery_pyplots_whats_new_99_spines.py
- sphx_glr_gallery_pyplots_compound_path_demo.py
- sphx_glr_gallery_pyplots_pyplot_scales.py
- sphx_glr_gallery_pyplots_boxplot_demo_pyplot.py
- sphx_glr_gallery_pyplots_whats_new_98_4_fancy.py
- sphx_glr_gallery_pyplots_text_layout.py
- sphx_glr_gallery_subplots_axes_and_figures_axes_props.py
- sphx_glr_gallery_subplots_axes_and_figures_invert_axes.py

- sphx_glr_gallery_subplots_axes_and_figures_subplot_toolbar.py
- sphx_glr_gallery_subplots_axes_and_figures_multiple_figs_demo.py
- sphx_glr_gallery_subplots_axes_and_figures_subplot.py
- sphx_glr_gallery_subplots_axes_and_figures_subplots_adjust.py
- sphx_glr_gallery_subplots_axes_and_figures_geo_demo.py
- sphx_glr_gallery_subplots_axes_and_figures_subplot_demo.py
- sphx_glr_gallery_subplots_axes_and_figures_custom_figure_class.py
- sphx_glr_gallery_subplots_axes_and_figures_shared_axis_demo.py
- sphx_glr_gallery_subplots_axes_and_figures_axes_margins.py
- sphx_glr_gallery_subplots_axes_and_figures_ganged_plots.py
- sphx_glr_gallery_subplots_axes_and_figures_align_labels_demo.py
- sphx_glr_gallery_subplots_axes_and_figures_figure_title.py
- sphx_glr_gallery_subplots_axes_and_figures_axhspan_demo.py
- sphx_glr_gallery_subplots_axes_and_figures_fahrenheit_celsius_scales.py
- sphx_glr_gallery_subplots_axes_and_figures_axis_equal_demo.py
- sphx_glr_gallery_subplots_axes_and_figures_axes_demo.py
- sphx_glr_gallery_subplots_axes_and_figures_subplots_demo.py
- sphx_glr_gallery_subplots_axes_and_figures_broken_axis.py
- sphx_glr_gallery_subplots_axes_and_figures_demo_tight_layout.py
- sphx_glr_gallery_subplots_axes_and_figures_axes_zoom_effect.py
- sphx_glr_gallery_color_color_by_yvalue.py
- sphx_glr_gallery_color_color_demo.py
- sphx_glr_gallery_color_colorbar_basics.py
- sphx_glr_gallery_color_color_cycle_default.py
- sphx_glr_gallery_color_color_cycler.py
- sphx_glr_gallery_color_named_colors.py
- sphx_glr_gallery_color_colormap_reference.py
- sphx_glr_gallery_statistics_errorbar.py
- sphx_glr_gallery_statistics_histogram_histtypes.py
- sphx_glr_gallery_statistics_errorbar_features.py
- sphx_glr_gallery_statistics_hexbin_demo.py
- sphx_glr_gallery_statistics_histogram_multihist.py

- sphx_glr_gallery_statistics_histogram_features.py
- sphx_glr_gallery_statistics_boxplot_vs_violin.py
- sphx_glr_gallery_statistics_histogram_cumulative.py
- sphx_glr_gallery_statistics_boxplot_color.py
- sphx_glr_gallery_statistics_multiple_histograms_side_by_side.py
- sphx_glr_gallery_statistics_errorbars_and_boxes.py
- sphx_glr_gallery_statistics_errorbar_limits.py
- sphx_glr_gallery_statistics_violinplot.py
- sphx_glr_gallery_statistics_hist.py
- sphx_glr_gallery_statistics_customized_violin.py
- sphx_glr_gallery_statistics_bxp.py
- sphx_glr_gallery_statistics_boxplot.py
- sphx_glr_gallery_statistics_barchart_demo.py
- sphx_glr_gallery_statistics_boxplot_demo.py
- sphx_glr_gallery_lines_bars_and_markers_interp_demo.py
- sphx_glr_gallery_lines_bars_and_markers_arctest.py
- sphx_glr_gallery_lines_bars_and_markers_stem_plot.py
- sphx_glr_gallery_lines_bars_and_markers_scatter_with_legend.py
- sphx_glr_gallery_lines_bars_and_markers_scatter_custom_symbol.py
- sphx_glr_gallery_lines_bars_and_markers_scatter_symbol.py
- sphx_glr_gallery_lines_bars_and_markers_step_demo.py
- sphx_glr_gallery_lines_bars_and_markers_xcorr_acorr_demo.py
- sphx_glr_gallery_lines_bars_and_markers_simple_plot.py
- sphx_glr_gallery_lines_bars_and_markers_line_demo_dash_control.py
- sphx_glr_gallery_lines_bars_and_markers_masked_demo.py
- sphx_glr_gallery_lines_bars_and_markers_fill.py
- sphx_glr_gallery_lines_bars_and_markers_nan_test.py
- sphx_glr_gallery_lines_bars_and_markers_errorbar_subsample.py
- sphx_glr_gallery_lines_bars_and_markers_scatter_masked.py
- sphx_glr_gallery_lines_bars_and_markers_barh.py
- sphx_glr_gallery_lines_bars_and_markers_bar_stacked.py
- sphx_glr_gallery_lines_bars_and_markers_broken_barh.py

- sphx_glr_gallery_lines_bars_and_markers_categorical_variables.py
- sphx_glr_gallery_lines_bars_and_markers_cohere.py
- sphx_glr_gallery_lines_bars_and_markers_vline_hline_demo.py
- sphx_glr_gallery_lines_bars_and_markers_scatter_star_poly.py
- sphx_glr_gallery_lines_bars_and_markers_line_styles_reference.py
- sphx_glr_gallery_lines_bars_and_markers_gradient_bar.py
- sphx_glr_gallery_lines_bars_and_markers_errorbar_limits_simple.py
- sphx_glr_gallery_lines_bars_and_markers_marker_fillstyle_reference.py
- sphx_glr_gallery_lines_bars_and_markers_csd_demo.py
- sphx_glr_gallery_lines_bars_and_markers_stackplot_demo.py
- sphx_glr_gallery_lines_bars_and_markers_scatter_demo2.py
- sphx_glr_gallery_lines_bars_and_markers_spectrum_demo.py
- sphx_glr_gallery_lines_bars_and_markers_scatter_hist.py
- sphx_glr_gallery_lines_bars_and_markers_eventcollection_demo.py
- sphx_glr_gallery_lines_bars_and_markers_multicolored_line.py
- sphx_glr_gallery_lines_bars_and_markers_fill_betweenx_demo.py
- sphx_glr_gallery_lines_bars_and_markers_barchart.py
- sphx_glr_gallery_lines_bars_and_markers_fill_between_demo.py
- sphx_glr_gallery_lines_bars_and_markers_linestyles.py
- sphx_glr_gallery_lines_bars_and_markers_eventplot_demo.py
- sphx_glr_gallery_lines_bars_and_markers_marker_reference.py
- sphx_glr_gallery_lines_bars_and_markers_markevery_demo.py
- sphx_glr_gallery_lines_bars_and_markers_psd_demo.py
- sphx_glr_gallery_images_contours_and_fields_matshow.py
- sphx_glr_gallery_images_contours_and_fields_spy_demos.py
- sphx_glr_gallery_images_contours_and_fields_figimage_demo.py
- sphx_glr_gallery_images_contours_and_fields_quiver_simple_demo.py
- sphx_glr_gallery_images_contours_and_fields_image_clip_path.py
- sphx_glr_gallery_images_contours_and_fields_barcode_demo.py
- sphx_glr_gallery_images_contours_and_fields_interpolation_methods.py
- sphx_glr_gallery_images_contours_and_fields_contour_corner_mask.py
- sphx_glr_gallery_images_contours_and_fields_specgram_demo.py

- sphx_glr_gallery_images_contours_and_fields_quadmesh_demo.py
- sphx_glr_gallery_images_contours_and_fields_layer_images.py
- sphx_glr_gallery_images_contours_and_fields_contourf_hatching.py
- sphx_glr_gallery_images_contours_and_fields_quiver_demo.py
- sphx_glr_gallery_images_contours_and_fields_affine_image.py
- sphx_glr_gallery_images_contours_and_fields_image_nonuniform.py
- sphx_glr_gallery_images_contours_and_fields_contourf_log.py
- sphx_glr_gallery_images_contours_and_fields_shading_example.py
- sphx_glr_gallery_images_contours_and_fields_pcolormesh_levels.py
- sphx_glr_gallery_images_contours_and_fields_multi_image.py
- sphx_glr_gallery_images_contours_and_fields_contour_label_demo.py
- sphx_glr_gallery_images_contours_and_fields_barb_demo.py
- sphx_glr_gallery_images_contours_and_fields_demo_bboximage.py
- sphx_glr_gallery_images_contours_and_fields_plot_streamplot.py
- sphx_glr_gallery_images_contours_and_fields_irregulardatagrid.py
- sphx_glr_gallery_images_contours_and_fields_triinterp_demo.py
- sphx_glr_gallery_images_contours_and_fields_image_demo.py
- sphx_glr_gallery_images_contours_and_fields_image_masked.py
- sphx_glr_gallery_images_contours_and_fields_pcolor_demo.py
- sphx_glr_gallery_images_contours_and_fields_contour_demo.py
- sphx_glr_gallery_images_contours_and_fields_tricontour_smooth_user.py
- sphx_glr_gallery_images_contours_and_fields_image_transparency_blend.py
- sphx_glr_gallery_images_contours_and_fields_contour_image.py
- sphx_glr_gallery_images_contours_and_fields_trigradient_demo.py
- sphx_glr_gallery_images_contours_and_fields_contourf_demo.py
- sphx_glr_gallery_images_contours_and_fields_triplot_demo.py
- sphx_glr_gallery_images_contours_and_fields_tricontour_demo.py
- sphx_glr_gallery_images_contours_and_fields_tripcolor_demo.py
- sphx_glr_gallery_images_contours_and_fields_custom_cmap.py
- sphx_glr_gallery_images_contours_and_fields_tricontour_smooth_delaunay.py
- sphx_glr_gallery_images_contours_and_fields_image_annotated_heatmap.py
- sphx_glr_gallery_shapes_and_collections_scatter.py

- sphx_glr_gallery_shapes_and_collections_marker_path.py
- sphx_glr_gallery_shapes_and_collections_ellipse_collection.py
- sphx_glr_gallery_shapes_and_collections_compound_path.py
- sphx_glr_gallery_shapes_and_collections_path_patch.py
- sphx_glr_gallery_shapes_and_collections_ellipse_demo.py
- sphx_glr_gallery_shapes_and_collections_hatch_demo.py
- sphx_glr_gallery_shapes_and_collections_donut.py
- sphx_glr_gallery_shapes_and_collections_line_collection.py
- sphx_glr_gallery_shapes_and_collections_artist_reference.py
- sphx_glr_gallery_shapes_and_collections_collections.py
- sphx_glr_gallery_shapes_and_collections_dolphin.py
- sphx_glr_gallery_shapes_and_collections_fancybox_demo.py
- sphx_glr_gallery_text_labels_and_annotations_arrow_simple_demo.py
- sphx_glr_gallery_text_labels_and_annotations_titles_demo.py
- sphx_glr_gallery_text_labels_and_annotations_dfrac_demo.py
- sphx_glr_gallery_text_labels_and_annotations_figlegend_demo.py
- sphx_glr_gallery_text_labels_and_annotations_mathtext_demo.py
- sphx_glr_gallery_text_labels_and_annotations_fancytextbox_demo.py
- sphx_glr_gallery_text_labels_and_annotations_text_fontdict.py
- sphx_glr_gallery_text_labels_and_annotations_tex_demo.py
- sphx_glr_gallery_text_labels_and_annotations_autowrap.py
- sphx_glr_gallery_text_labels_and_annotations_accented_text.py
- sphx_glr_gallery_text_labels_and_annotations_text_rotation_relative_to_line.py
- sphx_glr_gallery_text_labels_and_annotations_dashpointlabel.py
- sphx_glr_gallery_text_labels_and_annotations_text_rotation.py
- sphx_glr_gallery_text_labels_and_annotations_multiline.py
- sphx_glr_gallery_text_labels_and_annotations_date.py
- sphx_glr_gallery_text_labels_and_annotations_engineering_formatter.py
- sphx_glr_gallery_text_labels_and_annotations_custom_legends.py
- sphx_glr_gallery_text_labels_and_annotations_custom_date_formatter.py
- sphx_glr_gallery_text_labels_and_annotations_fancyarrow_demo.py
- sphx_glr_gallery_text_labels_and_annotations_demo_text_rotation_mode.py

- sphx_glr_gallery_text_labels_and_annotations_rainbow_text.py
- sphx_glr_gallery_text_labels_and_annotations_stix_fonts_demo.py
- sphx_glr_gallery_text_labels_and_annotations_font_table_ttf_sgskip.py
- sphx_glr_gallery_text_labels_and_annotations_usetex_baseline_test.py
- sphx_glr_gallery_text_labels_and_annotations_fonts_demo_kw.py
- sphx_glr_gallery_text_labels_and_annotations_text_alignment.py
- sphx_glr_gallery_text_labels_and_annotations_usetex_demo.py
- sphx_glr_gallery_text_labels_and_annotations_fonts_demo.py
- sphx_glr_gallery_text_labels_and_annotations_demo_annotation_box.py
- sphx_glr_gallery_text_labels_and_annotations_mathtext_examples.py
- sphx_glr_gallery_text_labels_and_annotations_demo_text_path.py
- sphx_glr_gallery_text_labels_and_annotations_legend_demo.py
- sphx_glr_gallery_text_labels_and_annotations_arrow_demo.py
- sphx_glr_gallery_text_labels_and_annotations_annotation_demo.py
- sphx_glr_gallery_pie_and_polar_charts_pie_features.py
- sphx_glr_gallery_pie_and_polar_charts_polar_bar.py
- sphx_glr_gallery_pie_and_polar_charts_polar_demo.py
- sphx_glr_gallery_pie_and_polar_charts_polar_legend.py
- sphx_glr_gallery_pie_and_polar_charts_polar_scatter.py
- sphx_glr_gallery_pie_and_polar_charts_pie_demo2.py
- sphx_glr_gallery_pie_and_polar_charts_nested_pie.py
- sphx_glr_gallery_pie_and_polar_charts_pie_and_donut_labels.py
- sphx_glr_gallery_style_sheets_dark_background.py
- sphx_glr_gallery_style_sheets_bmh.py
- sphx_glr_gallery_style_sheets_fivethirtyeight.py
- sphx_glr_gallery_style_sheets_grayscale.py
- sphx_glr_gallery_style_sheets_plot_solarizedlight2.py
- sphx_glr_gallery_style_sheets_ggplot.py
- sphx_glr_gallery_style_sheets_style_sheets_reference.py
- sphx_glr_gallery_showcase_integral.py
- sphx_glr_gallery_showcase_xkcd.py
- sphx_glr_gallery_showcase_mandelbrot.py

- sphx_glr_gallery_showcase_anatomy.py
- sphx_glr_gallery_showcase_firefox.py
- sphx_glr_gallery_showcase_bachelors_degrees_by_gender.py
- Animated line plot
- Decay
- Animated image using a precomputed list of images
- Animated histogram
- Oscilloscope
- The Bayes update
- MATPLOTLIB UNCHAINED
- Animated 3D random walk
- The double pendulum problem
- Rain simulation
- sphx_glr_gallery_axes_grid1_demo_imagegrid_aspect.py
- sphx_glr_gallery_axes_grid1_simple_axesgrid.py
- sphx_glr_gallery_axes_grid1_parasite_simple.py
- sphx_glr_gallery_axes_grid1_simple_axisline4.py
- sphx_glr_gallery_axes_grid1_demo_colorbar_with_axes_divider.py
- sphx_glr_gallery_axes_grid1_simple_rgb.py
- sphx_glr_gallery_axes_grid1_simple_axesgrid2.py
- sphx_glr_gallery_axes_grid1_simple_axes_divider1.py
- sphx_glr_gallery_axes_grid1_simple_axes_divider2.py
- sphx_glr_gallery_axes_grid1_simple_axes_divider3.py
- sphx_glr_gallery_axes_grid1_demo_colorbar_with_inset_locator.py
- sphx_glr_gallery_axes_grid1_demo_colorbar_of_inset_axes.py
- sphx_glr_gallery_axes_grid1_parasite_simple2.py
- sphx_glr_gallery_axes_grid1_demo_fixed_size_axes.py
- sphx_glr_gallery_axes_grid1_scatter_hist_locatable_axes.py
- sphx_glr_gallery_axes_grid1_demo_axes_hbox_divider.py
- sphx_glr_gallery_axes_grid1_make_room_for_ylabel_using_axesgrid.py
- sphx_glr_gallery_axes_grid1_demo_axes_rgb.py
- sphx_glr_gallery_axes_grid1_simple_anchored_artists.py

- sphx_glr_gallery_axes_grid1_inset_locator_demo2.py
- sphx_glr_gallery_axes_grid1_demo_edge_colorbar.py
- sphx_glr_gallery_axes_grid1_demo_axes_divider.py
- sphx_glr_gallery_axes_grid1_demo_axes_grid2.py
- sphx_glr_gallery_axes_grid1_demo_axes_grid.py
- sphx_glr_gallery_axes_grid1_inset_locator_demo.py
- sphx_glr_gallery_axisartist_simple_axisline3.py
- sphx_glr_gallery_axisartist_simple_axis_direction01.py
- sphx_glr_gallery_axisartist_axis_direction_demo_step01.py
- sphx_glr_gallery_axisartist_simple_axisartist1.py
- sphx_glr_gallery_axisartist_simple_axis_direction03.py
- sphx_glr_gallery_axisartist_simple_axisline2.py
- sphx_glr_gallery_axisartist_demo_axisline_style.py
- sphx_glr_gallery_axisartist_simple_axisline.py
- sphx_glr_gallery_axisartist_demo_ticklabel_alignment.py
- sphx_glr_gallery_axisartist_axis_direction_demo_step02.py
- sphx_glr_gallery_axisartist_demo_ticklabel_direction.py
- sphx_glr_gallery_axisartist_demo_parasite_axes2.py
- sphx_glr_gallery_axisartist_axis_direction_demo_step03.py
- sphx_glr_gallery_axisartist_demo_parasite_axes.py
- sphx_glr_gallery_axisartist_axis_direction_demo_step04.py
- sphx_glr_gallery_axisartist_demo_curvelinear_grid2.py
- sphx_glr_gallery_axisartist_demo_floating_axis.py
- sphx_glr_gallery_axisartist_demo_axis_direction.py
- sphx_glr_gallery_axisartist_simple_axis_pad.py
- sphx_glr_gallery_axisartist_demo_curvelinear_grid.py
- sphx_glr_gallery_axisartist_demo_floating_axes.py
- sphx_glr_gallery_event_handling_ginput_demo_sgskip.py
- sphx_glr_gallery_event_handling_close_event.py
- sphx_glr_gallery_event_handling_keypress_demo.py
- sphx_glr_gallery_event_handling_zoom_window.py
- sphx_glr_gallery_event_handling_timers.py

- sphx_glr_gallery_event_handling_pick_event_demo2.py
- sphx_glr_gallery_event_handling_coords_demo.py
- sphx_glr_gallery_event_handling_image_slices_viewer.py
- sphx_glr_gallery_event_handling_pong_sgskip.py
- sphx_glr_gallery_event_handling_legend_picking.py
- sphx_glr_gallery_event_handling_figure_axes_enter_leave.py
- sphx_glr_gallery_event_handling_looking_glass.py
- sphx_glr_gallery_event_handling_trifinder_event_demo.py
- sphx_glr_gallery_event_handling_ginput_manual_clabel_sgskip.py
- sphx_glr_gallery_event_handling_resample.py
- sphx_glr_gallery_event_handling_lasso_demo.py
- sphx_glr_gallery_event_handling_data_browser.py
- sphx_glr_gallery_event_handling_viewlims.py
- sphx_glr_gallery_event_handling_pick_event_demo.py
- sphx_glr_gallery_event_handling_path_editor.py
- sphx_glr_gallery_event_handling_poly_editor.py
- sphx_glr_gallery_frontpage_contour.py
- sphx_glr_gallery_misc_coords_report.py
- sphx_glr_gallery_misc_keyword_plotting.py
- sphx_glr_gallery_misc_agg_buffer_to_array.py
- sphx_glr_gallery_misc_pythonic_matplotlib.py
- sphx_glr_gallery_misc_fill_spiral.py
- sphx_glr_gallery_misc_tight_bbox_test.py
- sphx_glr_gallery_misc_customize_rc.py
- sphx_glr_gallery_misc_load_converter.py
- sphx_glr_gallery_misc_bbox_intersect.py
- sphx_glr_gallery_misc_set_and_get.py
- sphx_glr_gallery_misc_findobj_demo.py
- sphx_glr_gallery_misc_zorder_demo.py
- sphx_glr_gallery_misc_contour_manual.py
- sphx_glr_gallery_misc_transoffset.py
- sphx_glr_gallery_misc_plotfile_demo.py

- sphx_glr_gallery_misc_patheffect_demo.py
- sphx_glr_gallery_misc_table_demo.py
- sphx_glr_gallery_misc_cursor_demo_sgskip.py
- sphx_glr_gallery_misc_multiprocess_sgskip.py
- sphx_glr_gallery_misc_anchored_artists.py
- sphx_glr_gallery_misc_demo_ribbon_box.py
- sphx_glr_gallery_misc_demo_agg_filter.py
- sphx_glr_gallery_misc_custom_projection.py
- sphx_glr_gallery_mplot3d_contourf3d.py
- sphx_glr_gallery_mplot3d_wire3d.py
- sphx_glr_gallery_mplot3d_contour3d_2.py
- sphx_glr_gallery_mplot3d_contour3d.py
- sphx_glr_gallery_mplot3d_offset.py
- sphx_glr_gallery_mplot3d_lines3d.py
- sphx_glr_gallery_mplot3d_surface3d_2.py
- sphx_glr_gallery_mplot3d_wire3d_zero_stride.py
- sphx_glr_gallery_mplot3d_3d_bars.py
- sphx_glr_gallery_mplot3d_quiver3d.py
- sphx_glr_gallery_mplot3d_surface3d_radial.py
- sphx_glr_gallery_mplot3d_voxels.py
- sphx_glr_gallery_mplot3d_surface3d.py
- sphx_glr_gallery_mplot3d_text3d.py
- sphx_glr_gallery_mplot3d_contour3d_3.py
- sphx_glr_gallery_mplot3d_contourf3d_2.py
- sphx_glr_gallery_mplot3d_trisurf3d.py
- sphx_glr_gallery_mplot3d_scatter3d.py
- sphx_glr_gallery_mplot3d_mixed_subplots.py
- sphx_glr_gallery_mplot3d_surface3d_3.py
- sphx_glr_gallery_mplot3d_hist3d.py
- sphx_glr_gallery_mplot3d_bars3d.py
- sphx_glr_gallery_mplot3d_voxels_rgb.py
- sphx_glr_gallery_mplot3d_tricontour3d.py

- sphx_glr_gallery_mplot3d_voxels_torus.py
- sphx_glr_gallery_mplot3d_subplot3d.py
- sphx_glr_gallery_mplot3d_custom_shaded_3d_surface.py
- sphx_glr_gallery_mplot3d_tricontourf3d.py
- sphx_glr_gallery_mplot3d_voxels_numpy_logo.py
- sphx_glr_gallery_mplot3d_2dcollections3d.py
- sphx_glr_gallery_mplot3d_lorenz_attractor.py
- sphx_glr_gallery_mplot3d_polys3d.py
- sphx_glr_gallery_mplot3d_trisurf3d_2.py
- sphx_glr_gallery_mplot3d_pathpatch3d.py
- sphx_glr_gallery_recipes_share_axis_lims_views.py
- sphx_glr_gallery_recipes_transparent_legends.py
- sphx_glr_gallery_recipes_create_subplots.py
- sphx_glr_gallery_recipes_placing_text_boxes.py
- sphx_glr_gallery_recipes_common_date_problems.py
- sphx_glr_gallery_recipes_fill_between_alpha.py
- sphx_glr_gallery_scales_log_test.py
- sphx_glr_gallery_scales_log_bar.py
- sphx_glr_gallery_scales_aspect_loglog.py
- sphx_glr_gallery_scales_symlog_demo.py
- sphx_glr_gallery_scales_log_demo.py
- sphx_glr_gallery_scales_scales.py
- sphx_glr_gallery_scales_custom_scale.py
- sphx_glr_gallery_specialty_plots_mri_demo.py
- sphx_glr_gallery_specialty_plots_hinton_demo.py
- sphx_glr_gallery_specialty_plots_system_monitor.py
- sphx_glr_gallery_specialty_plots_mri_with_eeg.py
- sphx_glr_gallery_specialty_plots_advanced_hillshading.py
- sphx_glr_gallery_specialty_plots_anscombe.py
- sphx_glr_gallery_specialty_plots_topographic_hillshading.py
- sphx_glr_gallery_specialty_plots_leftventricle_bulleye.py
- sphx_glr_gallery_ticks_and_spines_tick_xlabel_top.py

- sphx_glr_gallery_ticks_and_spines_tick_label_right.py
- sphx_glr_gallery_ticks_and_spines_ticklabels_rotation.py
- sphx_glr_gallery_ticks_and_spines_custom_ticker1.py
- sphx_glr_gallery_ticks_and_spines_tick_labels_from_values.py
- sphx_glr_gallery_ticks_and_spines_auto_ticks.py
- sphx_glr_gallery_ticks_and_spines_spines_dropped.py
- sphx_glr_gallery_ticks_and_spines_spines_bounds.py
- sphx_glr_gallery_ticks_and_spines_date_demo_rrule.py
- sphx_glr_gallery_ticks_and_spines_colorbar_tick_labelling_demo.py
- sphx_glr_gallery_ticks_and_spines_spines.py
- sphx_glr_gallery_ticks_and_spines_major_minor_demo.py
- sphx_glr_gallery_ticks_and_spines_date_demo_convert.py
- sphx_glr_gallery_ticks_and_spines_centered_ticklabels.py
- sphx_glr_gallery_ticks_and_spines_date_index_formatter2.py
- sphx_glr_gallery_ticks_and_spines_multiple_yaxis_with_spines.py
- sphx_glr_gallery_ticks_and_spines_spine_placement_demo.py
- sphx_glr_gallery_ticks_and_spines_scalarformatter.py
- sphx_glr_gallery_ticks_and_spines_tick-locators.py
- sphx_glr_gallery_ticks_and_spines_tick-formatters.py
- sphx_glr_gallery_units_radian_demo.py
- sphx_glr_gallery_units_units_sample.py
- sphx_glr_gallery_units_units_scatter.py
- sphx_glr_gallery_units_bar_demo2.py
- sphx_glr_gallery_units_annotate_with_units.py
- sphx_glr_gallery_units_bar_unit_demo.py
- sphx_glr_gallery_units_artist_tests.py
- sphx_glr_gallery_units_ellipse_with_units.py
- sphx_glr_gallery_units_evans_test.py
- sphx_glr_gallery_user_interfaces_lineprops_dialog_gtk_sgskip.py
- sphx_glr_gallery_user_interfaces_pylab_with_gtk_sgskip.py
- sphx_glr_gallery_user_interfaces_toolmanager_sgskip.py
- sphx_glr_gallery_userdemo_anchored_box01.py

- sphx_glr_gallery_userdemo_annotate_simple01.py
- sphx_glr_gallery_userdemo_annotate_simple02.py
- sphx_glr_gallery_userdemo_anchored_box03.py
- sphx_glr_gallery_userdemo_simple_legend02.py
- sphx_glr_gallery_userdemo_anchored_box02.py
- sphx_glr_gallery_userdemo_annotate_simple03.py
- sphx_glr_gallery_userdemo_annotate_simple_coord01.py
- sphx_glr_gallery_userdemo_demo_gridspec05.py
- sphx_glr_gallery_userdemo_simple_legend01.py
- sphx_glr_gallery_userdemo_annotate_simple_coord02.py
- sphx_glr_gallery_userdemo_demo_gridspec01.py
- sphx_glr_gallery_userdemo_demo_gridspec02.py
- sphx_glr_gallery_userdemo_annotate_simple_coord03.py
- sphx_glr_gallery_userdemo_colormap_normalizations_power.py
- sphx_glr_gallery_userdemo_connect_simple01.py
- sphx_glr_gallery_userdemo_demo_gridspec03.py
- sphx_glr_gallery_userdemo_colormap_normalizations_lognorm.py
- sphx_glr_gallery_userdemo_annotate_text_arrow.py
- sphx_glr_gallery_userdemo_demo_gridspec04.py
- sphx_glr_gallery_userdemo_colormap_normalizations_symlognorm.py
- sphx_glr_gallery_userdemo_annotate_simple04.py
- sphx_glr_gallery_userdemo_anchored_box04.py
- sphx_glr_gallery_userdemo_colormap_normalizations_bounds.py
- sphx_glr_gallery_userdemo_demo_gridspec06.py
- sphx_glr_gallery_userdemo_custom_boxstyle01.py
- sphx_glr_gallery_userdemo_colormap_normalizations_custom.py
- sphx_glr_gallery_userdemo_connectionstyle_demo.py
- sphx_glr_gallery_userdemo_custom_boxstyle02.py
- sphx_glr_gallery_userdemo_annotate_explain.py
- sphx_glr_gallery_userdemo_simple_annotate01.py
- sphx_glr_gallery_userdemo_colormap_normalizations.py
- sphx_glr_gallery_widgets_multicursor.py

- [sphx_glr_gallery_widgets_cursor.py](#)
- [sphx_glr_gallery_widgets_textbox.py](#)
- [sphx_glr_gallery_widgets_check_buttons.py](#)
- [sphx_glr_gallery_widgets_buttons.py](#)
- [sphx_glr_gallery_widgets_span_selector.py](#)
- [sphx_glr_gallery_widgets_radio_buttons.py](#)
- [sphx_glr_gallery_widgets_slider_demo.py](#)
- [sphx_glr_gallery_widgets_rectangle_selector.py](#)
- [sphx_glr_gallery_widgets_polygon_selector_demo.py](#)
- [sphx_glr_gallery_widgets_lasso_selector_demo_sgskip.py](#)
- [sphx_glr_gallery_widgets_menu.py](#)
- [Sample plots in Matplotlib](#)
- [Customizing Matplotlib with style sheets and rcParams](#)
- [Usage Guide](#)
- [Pyplot tutorial](#)
- [The Lifecycle of a Plot](#)
- [Styling withycler](#)
- [Legend guide](#)
- [Artist tutorial](#)
- [Customizing Figure Layouts Using GridSpec and Other Functions](#)
- [Tight Layout guide](#)
- [Constrained Layout Guide](#)
- [Path effects guide](#)
- [Path Tutorial](#)
- [Transformations Tutorial](#)
- [Colormaps in Matplotlib](#)
- [Text properties and layout](#)
- [Text in Matplotlib Plots](#)

75.1.115 matplotlib.pyplot.specgram

```
matplotlib.pyplot.specgram(x, NFFT=None, Fs=None, Fc=None, detrend=None, win-
                             dow=None, noverlap=None, cmap=None, xextent=None,
                             pad_to=None, sides=None, scale_by_freq=None, mode=None,
                             scale=None, vmin=None, vmax=None, hold=None, data=None,
                             **kwargs)
```

Plot a spectrogram.

Call signature:

```
specgram(x, NFFT=256, Fs=2, Fc=0, detrend=mlab.detrend_none,
          window=mlab.window_hanning, noverlap=128,
          cmap=None, xextent=None, pad_to=None, sides='default',
          scale_by_freq=None, mode='default', scale='default',
          **kwargs)
```

Compute and plot a spectrogram of data in `x`. Data are split into NFFT length segments and the spectrum of each section is computed. The windowing function `window` is applied to each segment, and the amount of overlap of each segment is specified with `noverlap`. The spectrogram is plotted as a colormap (using `imshow`).

Parameters

- `x` [1-D array or sequence] Array or sequence containing the data.
- `Fs` [scalar] The sampling frequency (samples per time unit). It is used to calculate the Fourier frequencies, `freqs`, in cycles per time unit. The default value is 2.
- `window` [callable or ndarray] A function or a vector of length NFFT. To create window vectors see `window_hanning()`, `window_none()`, `numpy.blackman()`, `numpy.hamming()`, `numpy.bartlett()`, `scipy.signal()`, `scipy.signal.get_window()`, etc. The default is `window_hanning()`. If a function is passed as the argument, it must take a data segment as an argument and return the windowed version of the segment.
- `sides` [['default' | 'onesided' | 'twosided']] Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.
- `pad_to` [integer] The number of points to which the data segment is padded when performing the FFT. This can be different from NFFT, which specifies the number of data points used. While not increasing the actual resolution of the spectrum (the minimum distance between resolvable peaks), this can give more points in the plot, allowing for more detail. This corresponds to the `n` parameter in the call to `fft()`. The default is `None`, which sets `pad_to` equal to NFFT
- `NFFT` [integer] The number of data points used in each block for the FFT. A power 2 is most efficient. The default value is 256. This should NOT be

used to get zero padding, or the scaling of the result will be incorrect. Use `pad_` to for this instead.

`detrend` [{`'default'`, `'constant'`, `'mean'`, `'linear'`, `'none'`} or callable] The function applied to each segment before fft-ing, designed to remove the mean or linear trend. Unlike in MATLAB, where the `detrend` parameter is a vector, in matplotlib is it a function. The `pylab` module defines `detrend_none()`, `detrend_mean()`, and `detrend_linear()`, but you can use a custom function as well. You can also use a string to choose one of the functions. `'default'`, `'constant'`, and `'mean'` call `detrend_mean()`. `'linear'` calls `detrend_linear()`. `'none'` calls `detrend_none()`.

`scale_by_freq` [boolean, optional] Specifies whether the resulting density values should be scaled by the scaling frequency, which gives density in units of Hz^{-1} . This allows for integration over the returned frequency values. The default is `True` for MATLAB compatibility.

`mode` [[`'default'` | `'psd'` | `'magnitude'` | `'angle'` | `'phase'`]] What sort of spectrum to use. Default is `'psd'`, which takes the power spectral density. `'complex'` returns the complex-valued frequency spectrum. `'magnitude'` returns the magnitude spectrum. `'angle'` returns the phase spectrum without unwrapping. `'phase'` returns the phase spectrum with unwrapping.

`noverlap` [integer] The number of points of overlap between blocks. The default value is 128.

`scale` [[`'default'` | `'linear'` | `'dB'`]] The scaling of the values in the spec. `'linear'` is no scaling. `'dB'` returns the values in dB scale. When mode is `'psd'`, this is dB power ($10 * \log_{10}$). Otherwise this is dB amplitude ($20 * \log_{10}$). `'default'` is `'dB'` if mode is `'psd'` or `'magnitude'` and `'linear'` otherwise. This must be `'linear'` if mode is `'angle'` or `'phase'`.

`Fc` [integer] The center frequency of `x` (defaults to 0), which offsets the `x` extents of the plot to reflect the frequency range used when a signal is acquired and then filtered and downsampled to baseband.

`cmap` : A `matplotlib.colors.Colormap` instance; if `None`, use default determined by `rc`

`xextent` [[`None` | (`xmin`, `xmax`)]] The image extent along the x-axis. The default sets `xmin` to the left border of the first bin (spectrum column) and `xmax` to the right border of the last bin. Note that for `noverlap>0` the width of the bins is smaller than those of the segments.

****kwargs** : Additional kwargs are passed on to `imshow` which makes the spectrogram image

Returns

`spectrum` [2-D array] Columns are the periodograms of successive segments.

`freqs` [1-D array] The frequencies corresponding to the rows in `spectrum`.

`t` [1-D array] The times corresponding to midpoints of segments (i.e., the columns in spectrum).

`im` [instance of class AxesImage] The image created by `imshow` containing the spectrogram

See also:

`psd()` `psd()` differs in the default overlap; in returning the mean of the segment periodograms; in not returning times; and in generating a line plot instead of colormap.

`magnitude_spectrum()` A single spectrum, similar to having a single segment when mode is 'magnitude'. Plots a line instead of a colormap.

`angle_spectrum()` A single spectrum, similar to having a single segment when mode is 'angle'. Plots a line instead of a colormap.

`phase_spectrum()` A single spectrum, similar to having a single segment when mode is 'phase'. Plots a line instead of a colormap.

Notes

The parameters `detrend` and `scale_by_freq` do only apply when mode is set to 'psd'.

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'x'.
-

Examples using `matplotlib.pyplot.specgram`

- `sphx_glr_gallery_images_contours_and_fields_specgram_demo.py`

75.1.116 `matplotlib.pyplot.spectral`

`matplotlib.pyplot.spectral()`

Set the colormap to "spectral".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.117 `matplotlib.pyplot.spring`

`matplotlib.pyplot.spring()`

Set the colormap to "spring".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.118 `matplotlib.pyplot.spy`

`matplotlib.pyplot.spy(Z, precision=0, marker=None, markersize=None, aspect='equal', **kwargs)`

Plot the sparsity pattern on a 2-D array.

`spy(Z)` plots the sparsity pattern of the 2-D array `Z`.

Parameters

`Z` [sparse array (n, m)] The array to be plotted.

`precision` [float, optional, default: 0] If precision is 0, any non-zero value will be plotted; else, values of $|Z| > \textit{precision}$ will be plotted.

For `scipy.sparse.spmatrix` instances, there is a special case: if precision is 'present', any value present in the array will be plotted, even if it is identically zero.

`origin` [["upper", "lower"], optional, default: "upper"] Place the [0,0] index of the array in the upper left or lower left corner of the axes.

`aspect` [['auto' | 'equal' | scalar], optional, default: "equal"] If 'equal', and extent is None, changes the axes aspect ratio to match that of the image. If extent is not None, the axes aspect ratio is changed to match that of the extent.

If 'auto', changes the image aspect ratio to match that of the axes.

If None, default to `rc image.aspect` value.

Two plotting styles are available: image or marker. Both are available for full arrays, but only the marker style works for `:class:'scipy.sparse.spmatrix'` instances.

If `*marker*` and `*markersize*` are `*None*`, an image will be returned and any remaining kwargs are passed to

`:func:'~matplotlib.pyplot.imshow';` else, a

`:class:'~matplotlib.lines.Line2D'` object will be returned with the value of marker determining the marker type, and any remaining kwargs passed to the

`:meth:'~matplotlib.axes.Axes.plot'` method.

If `*marker*` and `*markersize*` are `*None*`, useful kwargs include:

`* *cmap*`

`**alpha*`

See also:

`imshow` for image options.

`plot` for plotting options

Examples using `matplotlib.pyplot.spy`

- `sphx_glr_gallery_images_contours_and_fields_spy_demos.py`

75.1.119 `matplotlib.pyplot.stackplot`

`matplotlib.pyplot.stackplot(x, *args, **kwargs)`

Draws a stacked area plot.

Parameters

`x` [1d array of dimension N]

`y` [2d array (dimension MxN), or sequence of 1d arrays (each dimension 1xN)]
The data is assumed to be unstacked. Each of the following calls is legal:

<code>stackplot(x, y)</code>	<code># where y is MxN</code>
<code>stackplot(x, y1, y2, y3, y4)</code>	<code># where y1, y2, y3, y4, are all 1xNm</code>

`baseline` `['zero' | 'sym' | 'wiggle' | 'weighted_wiggle']` Method used to calculate the baseline:

- 'zero': Constant zero baseline, i.e. a simple stacked plot.
- 'sym': Symmetric around zero and is sometimes called 'ThemeRiver'.
- 'wiggle': Minimizes the sum of the squared slopes.
- 'weighted_wiggle': Does the same but weights to account for size of each layer. It is also called 'Streamgraph'-layout. More details can be found at <http://leebyron.com/streamgraph/>.

`labels` [Length N sequence of strings] Labels to assign to each data series.

`colors` [Length N sequence of colors] A list or tuple of colors. These will be cycled through and used to colour the stacked areas.

`**kwargs` : All other keyword arguments are passed to `Axes.fill_between()`.

Returns

`list` [list of `PolyCollection`] A list of `PolyCollection` instances, one for each element in the stacked area plot.

75.1.120 matplotlib.pyplot.stem

matplotlib.pyplot.stem(*args, **kwargs)

Create a stem plot.

A stem plot plots vertical lines at each x location from the baseline to y, and places a marker there.

Call signature:

```
stem([x,] y, linefmt=None, markerfmt=None, basefmt=None)
```

The x-positions are optional. The formats may be provided either as positional or as keyword-arguments.

Parameters

x [array-like, optional] The x-positions of the stems. Default: (0, 1, ..., len(y) - 1).

y [array-like] The y-values of the stem heads.

linefmt [str, optional] A string defining the properties of the vertical lines. Usually, this will be a color or a color and a linestyle:

Character	Line Style
'-'	solid line
'--'	dashed line
'-.'	dash-dot line
':'	dotted line

Default: 'C0-', i.e. solid line with the first color of the color cycle.

Note: While it is technically possible to specify valid formats other than color or color and linestyle (e.g. 'rx' or '-.'), this is beyond the intention of the method and will most likely not result in a reasonable reasonable plot.

markerfmt [str, optional] A string defining the properties of the markers at the stem heads. Default: 'C0o', i.e. filled circles with the first color of the color cycle.

basefmt [str, optional] A format string defining the properties of the baseline.
Default: 'C3-' ('C2-' in classic mode).

bottom [float, optional, default: 0] The y-position of the baseline.

label [str, optional, default: None] The label to use for the stems in legends.

Returns

container [StemContainer] The container may be treated like a tuple (markerline, stemlines, baseline)

Other Parameters

****kwargs** No other parameters are supported. They are currently ignored silently for backward compatibility. This behavior is deprecated. Future versions will not accept any other parameters and will raise a `TypeError` instead.

Notes

See also:

The MATLAB function `stem` which inspired this method.

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All positional and all keyword arguments.
-

Examples using `matplotlib.pyplot.stem`

- `sphx_glr_gallery_lines_bars_and_markers_stem_plot.py`

75.1.121 `matplotlib.pyplot.step`

`matplotlib.pyplot.step(x, y, *args, **kwargs)`

Make a step plot.

Call signatures:

```
step(x, y, [fmt], *, data=None, where='pre', **kwargs)
step(x, y, [fmt], x2, y2, [fmt2], ..., *, where='pre', **kwargs)
```

This is just a thin wrapper around `plot` which changes some formatting options. Most of the concepts and parameters of `plot` can be used here as well.

Parameters

- x** `[array_like]` 1-D sequence of x positions. It is assumed, but not checked, that it is uniformly increasing.
- y** `[array_like]` 1-D sequence of y levels.
- fmt** `[str, optional]` A format string, e.g. `'g'` for a green line. See `plot` for a more detailed description.

Note: While full format strings are accepted, it is recommended to only specify the color. Line styles are currently ignored (use the keyword argument `linestyle` instead). Markers are accepted and plotted on the given positions, however, this is a rarely needed feature for step plots.

data [indexable object, optional] An object with labelled data. If given, provide the label names to plot in x and y.

where [{`'pre'`, `'post'`, `'mid'`}, optional, default `'pre'`] Define where the steps should be placed:

- `'pre'`: The y value is continued constantly to the left from every x position, i.e. the interval $(x[i-1], x[i])$ has the value $y[i]$.
- `'post'`: The y value is continued constantly to the right from every x position, i.e. the interval $[x[i], x[i+1])$ has the value $y[i]$.
- `'mid'`: Steps occur half-way between the x positions.

Returns

lines A list of Line2D objects representing the plotted data.

Other Parameters

****kwargs** Additional parameters are the same as those for plot.

Notes

Note: In addition to the above described arguments, this function can take a data key-word argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: `'x'`, `'y'`.
-

Examples using matplotlib.pyplot.step

- sphx_glr_gallery_lines_bars_and_markers_step_demo.py

75.1.122 matplotlib.pyplot.streamplot

`matplotlib.pyplot.streamplot(x, y, u, v, density=1, linewidth=None, color=None, cmap=None, norm=None, arrowsize=1, arrowstyle='>', minlength=0.1, transform=None, zorder=None, start_points=None, maxlength=4.0, integration_direction='both', hold=None, data=None)`

Draws streamlines of a vector flow.

x, y [1d arrays] an evenly spaced grid.

u, v [2d arrays] x and y-velocities. Number of rows should match length of y, and the number of columns should match x.

density [float or 2-tuple] Controls the closeness of streamlines. When `density = 1`, the domain is divided into a 30x30 grid—density linearly scales this grid. Each cell in the grid can have, at most, one traversing streamline. For different densities in each direction, use `[density_x, density_y]`.

linewidth [numeric or 2d array] vary linewidth when given a 2d array with the same shape as velocities.

color [matplotlib color code, or 2d array] Streamline color. When given an array with the same shape as velocities, color values are converted to colors using `cmap`.

cmap [Colormap] Colormap used to plot streamlines and arrows. Only necessary when using an array input for color.

norm [Normalize] Normalize object used to scale luminance data to 0, 1. If `None`, stretch `(min, max)` to `(0, 1)`. Only necessary when color is an array.

arrowsize [float] Factor scale arrow size.

arrowstyle [str] Arrow style specification. See `FancyArrowPatch`.

minlength [float] Minimum length of streamline in axes coordinates.

start_points: Nx2 array Coordinates of starting points for the streamlines. In data coordinates, the same as the `x` and `y` arrays.

zorder [int] any number

maxlength [float] Maximum length of streamline in axes coordinates.

integration_direction [['forward', 'backward', 'both']] Integrate the streamline in forward, backward or both directions.

Returns:

stream_container [StreamplotSet] Container object with attributes

- **lines**: `matplotlib.collections.LineCollection` of streamlines
- **arrows**: collection of `matplotlib.patches.FancyArrowPatch` objects representing arrows half-way along stream lines.

This container will probably change in the future to allow changes to the colormap, alpha, etc. for both lines and arrows, but these changes should be backward compatible.

Examples using `matplotlib.pyplot.streamplot`

- `sphx_glr_gallery_images_contours_and_fields_plot_streamplot.py`

75.1.123 `matplotlib.pyplot.subplot`

`matplotlib.pyplot.subplot(*args, **kwargs)`

Return a subplot axes at the given grid position.

Call signature:

`subplot(nrows, ncols, index, **kwargs)`

In the current figure, create and return an Axes, at position index of a (virtual) grid of nrows by ncols axes. Indexes go from 1 to nrows * ncols, incrementing in row-major order.

If nrows, ncols and index are all less than 10, they can also be given as a single, concatenated, three-digit number.

For example, subplot(2, 3, 3) and subplot(233) both create an matplotlib.axes.Axes at the top right corner of the current figure, occupying half of the figure height and a third of the figure width.

Note: Creating a subplot will delete any pre-existing subplot that overlaps with it beyond sharing a boundary:

```
import matplotlib.pyplot as plt
# plot a line, implicitly creating a subplot(111)
plt.plot([1,2,3])
# now create a subplot which represents the top plot of a grid
# with 2 rows and 1 column. Since this subplot will overlap the
# first, the plot (and its axes) previously created, will be removed
plt.subplot(211)
plt.plot(range(12))
plt.subplot(212, facecolor='y') # creates 2nd subplot with yellow background
```

If you do not want this behavior, use the add_subplot() method or the axes() function instead.

Keyword arguments:

facecolor: The background color of the subplot, which can be any valid color specifier. See matplotlib.colors for more information.

polar: A boolean flag indicating whether the subplot plot should be a polar projection. Defaults to False.

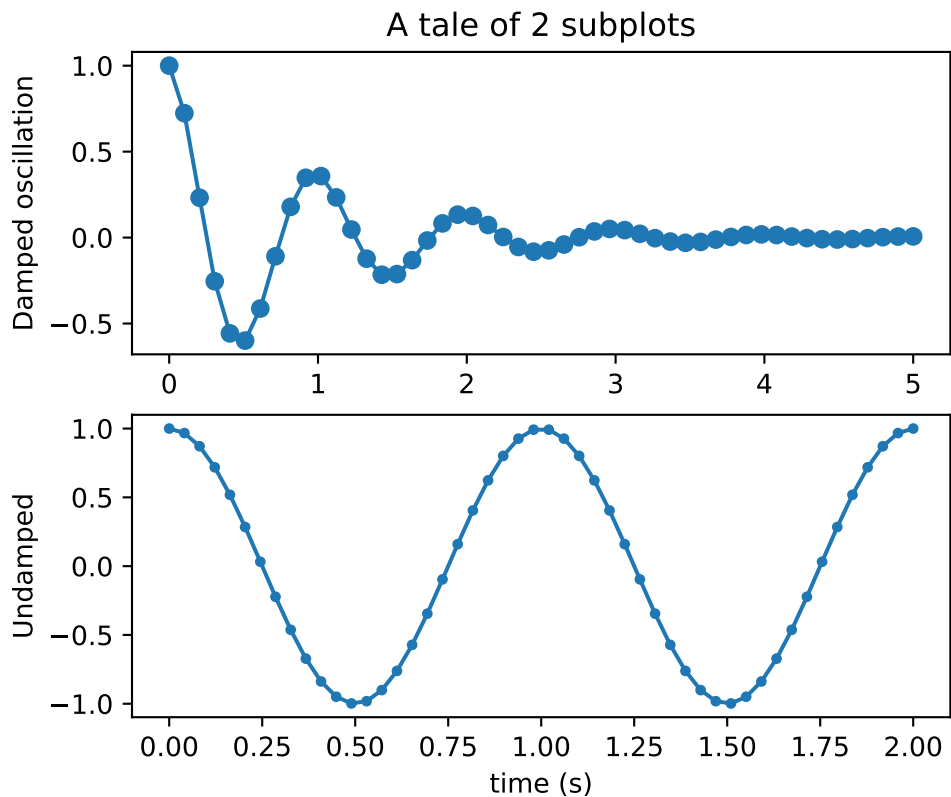
projection: A string giving the name of a custom projection to be used for the subplot. This projection must have been previously registered. See matplotlib.projections.

See also:

axes() For additional information on axes() and subplot() keyword arguments.

gallery/pie_and_polar_charts/polar_scatter.py For an example

Example:



Examples using `matplotlib.pyplot.subplot`

- `sphinx_glr_gallery_pyplots_whats_new_98_4_legend.py`
- `sphinx_glr_gallery_pyplots_pyplot_two_subplots.py`
- `sphinx_glr_gallery_pyplots_pyplot_scales.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_multiple_figs_demo.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_subplot.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_subplots_adjust.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_geo_demo.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_shared_axis_demo.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_axes_margins.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_figure_title.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_demo_tight_layout.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_axes_zoom_effect.py`
- `sphinx_glr_gallery_lines_bars_and_markers_nan_test.py`
- `sphinx_glr_gallery_lines_bars_and_markers_scatter_star_poly.py`

- [sphx_glr_gallery_lines_bars_and_markers_linestyles.py](#)
- [sphx_glr_gallery_lines_bars_and_markers_psd_demo.py](#)
- [sphx_glr_gallery_shapes_and_collections_ellipse_demo.py](#)
- [sphx_glr_gallery_shapes_and_collections_fancybox_demo.py](#)
- [sphx_glr_gallery_text_labels_and_annotations_multiline.py](#)
- [sphx_glr_gallery_text_labels_and_annotations_fonts_demo_kw.py](#)
- [sphx_glr_gallery_text_labels_and_annotations_fonts_demo.py](#)
- [sphx_glr_gallery_text_labels_and_annotations_demo_text_path.py](#)
- [sphx_glr_gallery_pie_and_polar_charts_polar_bar.py](#)
- [sphx_glr_gallery_pie_and_polar_charts_polar_demo.py](#)
- [sphx_glr_gallery_pie_and_polar_charts_pie_demo2.py](#)
- [MATPLOTLIB UNCHAINED](#)
- [sphx_glr_gallery_axes_grid1_simple_colorbar.py](#)
- [sphx_glr_gallery_event_handling_trifinder_event_demo.py](#)
- [sphx_glr_gallery_misc_customize_rc.py](#)
- [sphx_glr_gallery_misc_zorder_demo.py](#)
- [sphx_glr_gallery_misc_transoffset.py](#)
- [sphx_glr_gallery_misc_patheffect_demo.py](#)
- [sphx_glr_gallery_misc_demo_agg_filter.py](#)
- [sphx_glr_gallery_misc_custom_projection.py](#)
- [sphx_glr_gallery_recipes_share_axis_lims_views.py](#)
- [sphx_glr_gallery_scales_symlog_demo.py](#)
- [sphx_glr_gallery_specialty_plots_anscombe.py](#)
- [sphx_glr_gallery_ticks_and_spines_tick-locators.py](#)
- [sphx_glr_gallery_userdemo_demo_gridspec05.py](#)
- [sphx_glr_gallery_userdemo_simple_legend01.py](#)
- [sphx_glr_gallery_userdemo_demo_gridspec02.py](#)
- [sphx_glr_gallery_userdemo_demo_gridspec03.py](#)
- [Pyplot tutorial](#)
- [Legend guide](#)
- [Tight Layout guide](#)
- [Constrained Layout Guide](#)

75.1.124 matplotlib.pyplot.subplot2grid

`matplotlib.pyplot.subplot2grid(shape, loc, rowspan=1, colspan=1, fig=None, **kwargs)`

Create an axis at specific location inside a regular grid.

Parameters

`shape` [sequence of 2 ints] Shape of grid in which to place axis. First entry is number of rows, second entry is number of columns.

`loc` [sequence of 2 ints] Location to place axis within grid. First entry is row number, second entry is column number.

`rowspan` [int] Number of rows for the axis to span to the right.

`colspan` [int] Number of columns for the axis to span downwards.

`fig` [Figure, optional] Figure to place axis in. Defaults to current figure.

`**kwargs` Additional keyword arguments are handed to `add_subplot`.

Notes

The following call

```
subplot2grid(shape, loc, rowspan=1, colspan=1)
```

is identical to

```
gridspec=GridSpec(shape[0], shape[1])
subplotspec=gridspec.new_subplotspec(loc, rowspan, colspan)
subplot(subplotspec)
```

Examples using matplotlib.pyplot.subplot2grid

- [sphx_glr_gallery_subplots_axes_and_figures_demo_tight_layout.py](#)
- [sphx_glr_gallery_userdemo_demo_gridspec01.py](#)
- [Tight Layout guide](#)
- [Constrained Layout Guide](#)

75.1.125 matplotlib.pyplot.subplot_tool

`matplotlib.pyplot.subplot_tool(targetfig=None)`

Launch a subplot tool window for a figure.

A `matplotlib.widgets.SubplotTool` instance is returned.

Examples using `matplotlib.pyplot.subplot_tool`

- `sphinx_glr_gallery_subplots_axes_and_figures_subplot_toolbar.py`

75.1.126 `matplotlib.pyplot.subplots`

```
matplotlib.pyplot.subplots(nrows=1, ncols=1, sharex=False, sharey=False, squeeze=True,
                           subplot_kw=None, gridspec_kw=None, **fig_kw)
```

Create a figure and a set of subplots

This utility wrapper makes it convenient to create common layouts of subplots, including the enclosing figure object, in a single call.

Parameters

`nrows, ncols` [int, optional, default: 1] Number of rows/columns of the subplot grid.

`sharex, sharey` [bool or {'none', 'all', 'row', 'col'}, default: False] Controls sharing of properties among x (`sharex`) or y (`sharey`) axes:

- True or 'all': x- or y-axis will be shared among all subplots.
- False or 'none': each subplot x- or y-axis will be independent.
- 'row': each subplot row will share an x- or y-axis.
- 'col': each subplot column will share an x- or y-axis.

When subplots have a shared x-axis along a column, only the x tick labels of the bottom subplot are created. Similarly, when subplots have a shared y-axis along a row, only the y tick labels of the first column subplot are created. To later turn other subplots' ticklabels on, use `tick_params()`.

`squeeze` [bool, optional, default: True]

- If True, extra dimensions are squeezed out from the returned array of Axes:
 - if only one subplot is constructed (`nrows=ncols=1`), the resulting single Axes object is returned as a scalar.
 - for Nx1 or 1xM subplots, the returned object is a 1D numpy object array of Axes objects.
 - for NxM, subplots with $N > 1$ and $M > 1$ are returned as a 2D array.
- If False, no squeezing at all is done: the returned Axes object is always a 2D array containing Axes instances, even if it ends up being 1x1.

`subplot_kw` [dict, optional] Dict with keywords passed to the `add_subplot()` call used to create each subplot.

`gridspec_kw` [dict, optional] Dict with keywords passed to the `GridSpec` constructor used to create the grid the subplots are placed on.

****fig_kw** : All additional keyword arguments are passed to the `figure()` call.

Returns

fig [matplotlib.figure.Figure object]
ax [Axes object or array of Axes objects.] **ax** can be either a single matplotlib.axes.Axes object or an array of Axes objects if more than one subplot was created. The dimensions of the resulting array can be controlled with the `squeeze` keyword, see above.

See also:

`figure`, `subplot`

Examples

First create some toy data:

```
>>> x = np.linspace(0, 2*np.pi, 400)
>>> y = np.sin(x**2)
```

Creates just a figure and only one subplot

```
>>> fig, ax = plt.subplots()
>>> ax.plot(x, y)
>>> ax.set_title('Simple plot')
```

Creates two subplots and unpacks the output array immediately

```
>>> f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
>>> ax1.plot(x, y)
>>> ax1.set_title('Sharing Y axis')
>>> ax2.scatter(x, y)
```

Creates four polar axes, and accesses them through the returned array

```
>>> fig, axes = plt.subplots(2, 2, subplot_kw=dict(polar=True))
>>> axes[0, 0].plot(x, y)
>>> axes[1, 1].scatter(x, y)
```

Share a X axis with each column of subplots

```
>>> plt.subplots(2, 2, sharex='col')
```

Share a Y axis with each row of subplots

```
>>> plt.subplots(2, 2, sharey='row')
```

Share both X and Y axes with all subplots

```
>>> plt.subplots(2, 2, sharex='all', sharey='all')
```

Note that this is the same as

```
>>> plt.subplots(2, 2, sharex=True, sharey=True)
```

Examples using `matplotlib.pyplot.subplots`

- `sphinx_glr_gallery_api_font_family_rc_sgskip.py`
- `sphinx_glr_gallery_api_unicode_minus.py`
- `sphinx_glr_gallery_api_font_file.py`
- `sphinx_glr_gallery_api_watermark_text.py`
- `sphinx_glr_gallery_api_quad_bezier.py`
- `sphinx_glr_gallery_api_legend.py`
- `sphinx_glr_gallery_api_watermark_image.py`
- `sphinx_glr_gallery_api_image_zcoord.py`
- `sphinx_glr_gallery_api_span_regions.py`
- `sphinx_glr_gallery_api_two_scales.py`
- `sphinx_glr_gallery_api_power_norm.py`
- `sphinx_glr_gallery_api_joinstyle.py`
- `sphinx_glr_gallery_api_histogram_path.py`
- `sphinx_glr_gallery_api_scatter_piecharts.py`
- `sphinx_glr_gallery_api_patch_collection.py`
- `sphinx_glr_gallery_api_line_with_text.py`
- `sphinx_glr_gallery_api_filled_step.py`
- `sphinx_glr_gallery_api_radar_chart.py`
- `sphinx_glr_gallery_pyplots_annotation_basic.py`
- `sphinx_glr_gallery_pyplots_whats_new_98_4_fill_between.py`
- `sphinx_glr_gallery_pyplots_dollar_ticks.py`
- `sphinx_glr_gallery_pyplots_align_ylabels.py`
- `sphinx_glr_gallery_pyplots_auto_subplots_adjust.py`
- `sphinx_glr_gallery_pyplots_annotate_transform.py`
- `sphinx_glr_gallery_pyplots_boxplot_demo_pyplot.py`
- `sphinx_glr_gallery_subplots_axes_and_figures_axes_props.py`

- sphx_glr_gallery_subplots_axes_and_figures_subplot_toolbar.py
- sphx_glr_gallery_subplots_axes_and_figures_subplot_demo.py
- sphx_glr_gallery_subplots_axes_and_figures_ganged_plots.py
- sphx_glr_gallery_subplots_axes_and_figures_fahrenheit_celsius_scales.py
- sphx_glr_gallery_subplots_axes_and_figures_axis_equal_demo.py
- sphx_glr_gallery_subplots_axes_and_figures_subplots_demo.py
- sphx_glr_gallery_subplots_axes_and_figures_broken_axis.py
- sphx_glr_gallery_subplots_axes_and_figures_demo_tight_layout.py
- sphx_glr_gallery_color_color_by_yvalue.py
- sphx_glr_gallery_color_color_demo.py
- sphx_glr_gallery_color_colorbar_basics.py
- sphx_glr_gallery_color_color_cycle_default.py
- sphx_glr_gallery_color_color_cycler.py
- sphx_glr_gallery_color_named_colors.py
- sphx_glr_gallery_color_colormap_reference.py
- sphx_glr_gallery_statistics_errorbar.py
- sphx_glr_gallery_statistics_histogram_histtypes.py
- sphx_glr_gallery_statistics_errorbar_features.py
- sphx_glr_gallery_statistics_hexbin_demo.py
- sphx_glr_gallery_statistics_histogram_multihist.py
- sphx_glr_gallery_statistics_histogram_features.py
- sphx_glr_gallery_statistics_boxplot_vs_violin.py
- sphx_glr_gallery_statistics_histogram_cumulative.py
- sphx_glr_gallery_statistics_boxplot_color.py
- sphx_glr_gallery_statistics_multiple_histograms_side_by_side.py
- sphx_glr_gallery_statistics_errorbars_and_boxes.py
- sphx_glr_gallery_statistics_errorbar_limits.py
- sphx_glr_gallery_statistics_violinplot.py
- sphx_glr_gallery_statistics_hist.py
- sphx_glr_gallery_statistics_customized_violin.py
- sphx_glr_gallery_statistics_bxp.py
- sphx_glr_gallery_statistics_boxplot.py

- sphx_glr_gallery_statistics_barchart_demo.py
- sphx_glr_gallery_statistics_boxplot_demo.py
- sphx_glr_gallery_lines_bars_and_markers_interp_demo.py
- sphx_glr_gallery_lines_bars_and_markers_scatter_with_legend.py
- sphx_glr_gallery_lines_bars_and_markers_scatter_custom_symbol.py
- sphx_glr_gallery_lines_bars_and_markers_xcorr_acorr_demo.py
- sphx_glr_gallery_lines_bars_and_markers_simple_plot.py
- sphx_glr_gallery_lines_bars_and_markers_line_demo_dash_control.py
- sphx_glr_gallery_lines_bars_and_markers_fill.py
- sphx_glr_gallery_lines_bars_and_markers_errorbar_subsample.py
- sphx_glr_gallery_lines_bars_and_markers_barh.py
- sphx_glr_gallery_lines_bars_and_markers_broken_barh.py
- sphx_glr_gallery_lines_bars_and_markers_categorical_variables.py
- sphx_glr_gallery_lines_bars_and_markers_cohere.py
- sphx_glr_gallery_lines_bars_and_markers_vline_hline_demo.py
- sphx_glr_gallery_lines_bars_and_markers_line_styles_reference.py
- sphx_glr_gallery_lines_bars_and_markers_gradient_bar.py
- sphx_glr_gallery_lines_bars_and_markers_marker_fillstyle_reference.py
- sphx_glr_gallery_lines_bars_and_markers_csd_demo.py
- sphx_glr_gallery_lines_bars_and_markers_stackplot_demo.py
- sphx_glr_gallery_lines_bars_and_markers_scatter_demo2.py
- sphx_glr_gallery_lines_bars_and_markers_spectrum_demo.py
- sphx_glr_gallery_lines_bars_and_markers_multicolored_line.py
- sphx_glr_gallery_lines_bars_and_markers_fill_betweenx_demo.py
- sphx_glr_gallery_lines_bars_and_markers_barchart.py
- sphx_glr_gallery_lines_bars_and_markers_fill_between_demo.py
- sphx_glr_gallery_lines_bars_and_markers_eventplot_demo.py
- sphx_glr_gallery_lines_bars_and_markers_marker_reference.py
- sphx_glr_gallery_lines_bars_and_markers_psd_demo.py
- sphx_glr_gallery_images_contours_and_fields_spy_demos.py
- sphx_glr_gallery_images_contours_and_fields_quiver_simple_demo.py
- sphx_glr_gallery_images_contours_and_fields_image_clip_path.py

- sphx_glr_gallery_images_contours_and_fields_interpolation_methods.py
- sphx_glr_gallery_images_contours_and_fields_contour_corner_mask.py
- sphx_glr_gallery_images_contours_and_fields_specgram_demo.py
- sphx_glr_gallery_images_contours_and_fields_quadmesh_demo.py
- sphx_glr_gallery_images_contours_and_fields_contourf_hatching.py
- sphx_glr_gallery_images_contours_and_fields_quiver_demo.py
- sphx_glr_gallery_images_contours_and_fields_affine_image.py
- sphx_glr_gallery_images_contours_and_fields_image_nonuniform.py
- sphx_glr_gallery_images_contours_and_fields_contourf_log.py
- sphx_glr_gallery_images_contours_and_fields_shading_example.py
- sphx_glr_gallery_images_contours_and_fields_pcolormesh_levels.py
- sphx_glr_gallery_images_contours_and_fields_multi_image.py
- sphx_glr_gallery_images_contours_and_fields_contour_label_demo.py
- sphx_glr_gallery_images_contours_and_fields_barb_demo.py
- sphx_glr_gallery_images_contours_and_fields_demo_bboximage.py
- sphx_glr_gallery_images_contours_and_fields_irregulardatagrid.py
- sphx_glr_gallery_images_contours_and_fields_triinterp_demo.py
- sphx_glr_gallery_images_contours_and_fields_image_demo.py
- sphx_glr_gallery_images_contours_and_fields_image_masked.py
- sphx_glr_gallery_images_contours_and_fields_pcolor_demo.py
- sphx_glr_gallery_images_contours_and_fields_contour_demo.py
- sphx_glr_gallery_images_contours_and_fields_tricontour_smooth_user.py
- sphx_glr_gallery_images_contours_and_fields_image_transparency_blend.py
- sphx_glr_gallery_images_contours_and_fields_contour_image.py
- sphx_glr_gallery_images_contours_and_fields_trigradient_demo.py
- sphx_glr_gallery_images_contours_and_fields_contourf_demo.py
- sphx_glr_gallery_images_contours_and_fields_triplot_demo.py
- sphx_glr_gallery_images_contours_and_fields_tricontour_demo.py
- sphx_glr_gallery_images_contours_and_fields_tripcolor_demo.py
- sphx_glr_gallery_images_contours_and_fields_custom_cmap.py
- sphx_glr_gallery_images_contours_and_fields_tricontour_smooth_delaunay.py
- sphx_glr_gallery_images_contours_and_fields_image_annotated_heatmap.py

- sphx_glr_gallery_shapes_and_collections_ellipse_collection.py
- sphx_glr_gallery_shapes_and_collections_compound_path.py
- sphx_glr_gallery_shapes_and_collections_path_patch.py
- sphx_glr_gallery_shapes_and_collections_ellipse_demo.py
- sphx_glr_gallery_shapes_and_collections_hatch_demo.py
- sphx_glr_gallery_shapes_and_collections_donut.py
- sphx_glr_gallery_shapes_and_collections_line_collection.py
- sphx_glr_gallery_shapes_and_collections_artist_reference.py
- sphx_glr_gallery_shapes_and_collections_collections.py
- sphx_glr_gallery_shapes_and_collections_dolphin.py
- sphx_glr_gallery_text_labels_and_annotations_figlegend_demo.py
- sphx_glr_gallery_text_labels_and_annotations_mathtext_demo.py
- sphx_glr_gallery_text_labels_and_annotations_tex_demo.py
- sphx_glr_gallery_text_labels_and_annotations_accented_text.py
- sphx_glr_gallery_text_labels_and_annotations_dashpointlabel.py
- sphx_glr_gallery_text_labels_and_annotations_text_rotation.py
- sphx_glr_gallery_text_labels_and_annotations_date.py
- sphx_glr_gallery_text_labels_and_annotations_engineering_formatter.py
- sphx_glr_gallery_text_labels_and_annotations_custom_legends.py
- sphx_glr_gallery_text_labels_and_annotations_custom_date_formatter.py
- sphx_glr_gallery_text_labels_and_annotations_demo_annotation_box.py
- sphx_glr_gallery_text_labels_and_annotations_legend_demo.py
- sphx_glr_gallery_text_labels_and_annotations_annotation_demo.py
- sphx_glr_gallery_pie_and_polar_charts_pie_features.py
- sphx_glr_gallery_pie_and_polar_charts_nested_pie.py
- sphx_glr_gallery_pie_and_polar_charts_pie_and_donut_labels.py
- sphx_glr_gallery_style_sheets_dark_background.py
- sphx_glr_gallery_style_sheets_bmh.py
- sphx_glr_gallery_style_sheets_fivethirtyeight.py
- sphx_glr_gallery_style_sheets_grayscale.py
- sphx_glr_gallery_style_sheets_ggplot.py
- sphx_glr_gallery_style_sheets_style_sheets_reference.py

- sphx_glr_gallery_showcase_integral.py
- sphx_glr_gallery_showcase_bachelors_degrees_by_gender.py
- sphx_glr_gallery_animation_animation_demo.py
- Animated line plot
- Decay
- Animated histogram
- Oscilloscope
- The Bayes update
- sphx_glr_gallery_axes_grid1_demo_colorbar_with_axes_divider.py
- sphx_glr_gallery_axes_grid1_demo_colorbar_with_inset_locator.py
- sphx_glr_gallery_axes_grid1_demo_colorbar_of_inset_axes.py
- sphx_glr_gallery_axes_grid1_scatter_hist_locatable_axes.py
- sphx_glr_gallery_axes_grid1_demo_axes_hbox_divider.py
- sphx_glr_gallery_axes_grid1_demo_axes_rgb.py
- sphx_glr_gallery_axes_grid1_inset_locator_demo2.py
- sphx_glr_gallery_axes_grid1_inset_locator_demo.py
- sphx_glr_gallery_event_handling_keypress_demo.py
- sphx_glr_gallery_event_handling_zoom_window.py
- sphx_glr_gallery_event_handling_timers.py
- sphx_glr_gallery_event_handling_pick_event_demo2.py
- sphx_glr_gallery_event_handling_coords_demo.py
- sphx_glr_gallery_event_handling_image_slices_viewer.py
- sphx_glr_gallery_event_handling_pong_sgskip.py
- sphx_glr_gallery_event_handling_legend_picking.py
- sphx_glr_gallery_event_handling_figure_axes_enter_leave.py
- sphx_glr_gallery_event_handling_looking_glass.py
- sphx_glr_gallery_event_handling_resample.py
- sphx_glr_gallery_event_handling_data_browser.py
- sphx_glr_gallery_event_handling_viewlims.py
- sphx_glr_gallery_event_handling_pick_event_demo.py
- sphx_glr_gallery_event_handling_path_editor.py
- sphx_glr_gallery_event_handling_poly_editor.py

- sphx_glr_gallery_frontpage_histogram.py
- sphx_glr_gallery_frontpage_membrane.py
- sphx_glr_gallery_frontpage_contour.py
- sphx_glr_gallery_frontpage_3D.py
- sphx_glr_gallery_misc_coords_report.py
- sphx_glr_gallery_misc_keyword_plotting.py
- sphx_glr_gallery_misc_agg_buffer_to_array.py
- sphx_glr_gallery_misc_pythonic_matplotlib.py
- sphx_glr_gallery_misc_load_converter.py
- sphx_glr_gallery_misc_bbox_intersect.py
- sphx_glr_gallery_misc_findobj_demo.py
- sphx_glr_gallery_misc_rasterization_demo.py
- sphx_glr_gallery_misc_cursor_demo_sgskip.py
- sphx_glr_gallery_misc_multiprocess_sgskip.py
- sphx_glr_gallery_misc_demo_ribbon_box.py
- sphx_glr_gallery_mplot3d_wire3d_zero_stride.py
- sphx_glr_gallery_mplot3d_custom_shaded_3d_surface.py
- sphx_glr_gallery_recipes_transparent_legends.py
- sphx_glr_gallery_recipes_create_subplots.py
- sphx_glr_gallery_recipes_placing_text_boxes.py
- sphx_glr_gallery_recipes_common_date_problems.py
- sphx_glr_gallery_recipes_fill_between_alpha.py
- sphx_glr_gallery_scales_log_test.py
- sphx_glr_gallery_scales_log_bar.py
- sphx_glr_gallery_scales_aspect_loglog.py
- sphx_glr_gallery_scales_log_demo.py
- sphx_glr_gallery_scales_scales.py
- sphx_glr_gallery_specialty_plots_mri_demo.py
- sphx_glr_gallery_specialty_plots_system_monitor.py
- sphx_glr_gallery_specialty_plots_advanced_hillshading.py
- sphx_glr_gallery_specialty_plots_topographic_hillshading.py
- sphx_glr_gallery_specialty_plots_leftventricle_bulleye.py

- sphx_glr_gallery_ticks_and_spines_tick_xlabel_top.py
- sphx_glr_gallery_ticks_and_spines_tick_label_right.py
- sphx_glr_gallery_ticks_and_spines_custom_ticker1.py
- sphx_glr_gallery_ticks_and_spines_tick_labels_from_values.py
- sphx_glr_gallery_ticks_and_spines_auto_ticks.py
- sphx_glr_gallery_ticks_and_spines_spines_dropped.py
- sphx_glr_gallery_ticks_and_spines_spines_bounds.py
- sphx_glr_gallery_ticks_and_spines_date_demo_rrule.py
- sphx_glr_gallery_ticks_and_spines_colorbar_tick_labelling_demo.py
- sphx_glr_gallery_ticks_and_spines_spines.py
- sphx_glr_gallery_ticks_and_spines_major_minor_demo.py
- sphx_glr_gallery_ticks_and_spines_date_demo_convert.py
- sphx_glr_gallery_ticks_and_spines_centered_ticklabels.py
- sphx_glr_gallery_ticks_and_spines_date_index_formatter2.py
- sphx_glr_gallery_ticks_and_spines_multiple_yaxis_with_spines.py
- sphx_glr_gallery_ticks_and_spines_scalarformatter.py
- sphx_glr_gallery_units_radian_demo.py
- sphx_glr_gallery_units_units_sample.py
- sphx_glr_gallery_units_units_scatter.py
- sphx_glr_gallery_units_bar_demo2.py
- sphx_glr_gallery_units_annotate_with_units.py
- sphx_glr_gallery_units_bar_unit_demo.py
- sphx_glr_gallery_units_artist_tests.py
- sphx_glr_gallery_units_evans_test.py
- sphx_glr_gallery_user_interfaces_lineprops_dialog_gtk_sgskip.py
- sphx_glr_gallery_user_interfaces_pylab_with_gtk_sgskip.py
- sphx_glr_gallery_user_interfaces_svg_tooltip_sgskip.py
- sphx_glr_gallery_userdemo_anchored_box01.py
- sphx_glr_gallery_userdemo_annotate_simple01.py
- sphx_glr_gallery_userdemo_annotate_simple02.py
- sphx_glr_gallery_userdemo_anchored_box03.py
- sphx_glr_gallery_userdemo_simple_legend02.py

- sphx_glr_gallery_userdemo_anchored_box02.py
- sphx_glr_gallery_userdemo_annotate_simple03.py
- sphx_glr_gallery_userdemo_annotate_simple_coord01.py
- sphx_glr_gallery_userdemo_annotate_simple_coord02.py
- sphx_glr_gallery_userdemo_annotate_simple_coord03.py
- sphx_glr_gallery_userdemo_colormap_normalizations_power.py
- sphx_glr_gallery_userdemo_connect_simple01.py
- sphx_glr_gallery_userdemo_colormap_normalizations_lognorm.py
- sphx_glr_gallery_userdemo_annotate_text_arrow.py
- sphx_glr_gallery_userdemo_colormap_normalizations_symlognorm.py
- sphx_glr_gallery_userdemo_annotate_simple04.py
- sphx_glr_gallery_userdemo_anchored_box04.py
- sphx_glr_gallery_userdemo_colormap_normalizations_bounds.py
- sphx_glr_gallery_userdemo_custom_boxstyle01.py
- sphx_glr_gallery_userdemo_colormap_normalizations_custom.py
- sphx_glr_gallery_userdemo_connectionstyle_demo.py
- sphx_glr_gallery_userdemo_custom_boxstyle02.py
- sphx_glr_gallery_userdemo_annotate_explain.py
- sphx_glr_gallery_userdemo_simple_annotate01.py
- sphx_glr_gallery_userdemo_colormap_normalizations.py
- sphx_glr_gallery_widgets_multicursor.py
- sphx_glr_gallery_widgets_textbox.py
- sphx_glr_gallery_widgets_check_buttons.py
- sphx_glr_gallery_widgets_buttons.py
- sphx_glr_gallery_widgets_span_selector.py
- sphx_glr_gallery_widgets_radio_buttons.py
- sphx_glr_gallery_widgets_slider_demo.py
- sphx_glr_gallery_widgets_rectangle_selector.py
- sphx_glr_gallery_widgets_polygon_selector_demo.py
- sphx_glr_gallery_widgets_lasso_selector_demo_sgskip.py
- [Sample plots in Matplotlib](#)
- [Usage Guide](#)

- [The Lifecycle of a Plot](#)
- [Styling withycler](#)
- [Artist tutorial](#)
- [Customizing Figure Layouts Using GridSpec and Other Functions](#)
- [Tight Layout guide](#)
- [Constrained Layout Guide](#)
- [Path Tutorial](#)
- [Transformations Tutorial](#)
- [Specifying Colors](#)
- [Customized Colorbars Tutorial](#)
- [Colormap Normalization](#)
- [Colormaps in Matplotlib](#)
- [Text in Matplotlib Plots](#)

75.1.127 matplotlib.pyplot.subplots_adjust

`matplotlib.pyplot.subplots_adjust(*args, **kwargs)`
Tune the subplot layout.

call signature:

```
subplots_adjust(left=None, bottom=None, right=None, top=None,  
                wspace=None, hspace=None)
```

The parameter meanings (and suggested defaults) are:

```
left  = 0.125 # the left side of the subplots of the figure  
right = 0.9   # the right side of the subplots of the figure  
bottom = 0.1  # the bottom of the subplots of the figure  
top   = 0.9   # the top of the subplots of the figure  
wspace = 0.2  # the amount of width reserved for space between subplots,  
              # expressed as a fraction of the average axis width  
hspace = 0.2  # the amount of height reserved for space between subplots,  
              # expressed as a fraction of the average axis height
```

The actual defaults are controlled by the rc file

Examples using `matplotlib.pyplot.subplots_adjust`

- [sphx_glr_gallery_pyplots_pyplot_scales.py](#)
- [sphx_glr_gallery_subplots_axes_and_figures_subplots_adjust.py](#)

- sphx_glr_gallery_subplots_axes_and_figures_figure_title.py
- sphx_glr_gallery_statistics_customized_violin.py
- sphx_glr_gallery_images_contours_and_fields_irregulardatagrid.py
- sphx_glr_gallery_text_labels_and_annotations_text_fontdict.py
- sphx_glr_gallery_text_labels_and_annotations_multiline.py
- sphx_glr_gallery_text_labels_and_annotations_legend_demo.py
- sphx_glr_gallery_axisartist_demo_parasite_axes2.py
- sphx_glr_gallery_misc_table_demo.py
- sphx_glr_gallery_misc_demo_agg_filter.py
- sphx_glr_gallery_ticks_and_spines_ticklabels_rotation.py
- sphx_glr_gallery_ticks_and_spines_spines.py
- sphx_glr_gallery_ticks_and_spines_tick-locators.py
- sphx_glr_gallery_widgets_textbox.py
- sphx_glr_gallery_widgets_check_buttons.py
- sphx_glr_gallery_widgets_buttons.py
- sphx_glr_gallery_widgets_radio_buttons.py
- sphx_glr_gallery_widgets_slider_demo.py
- [Pyplot tutorial](#)

75.1.128 matplotlib.pyplot.summer

matplotlib.pyplot.summer()

Set the colormap to "summer".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.129 matplotlib.pyplot.suptitle

matplotlib.pyplot.suptitle(*args, **kwargs)

Add a centered title to the figure.

kwargs are matplotlib.text.Text properties. Using figure coordinates, the defaults are:

x [0.5] The x location of the text in figure coords

y [0.98] The y location of the text in figure coords

horizontalalignment ['center'] The horizontal alignment of the text

verticalalignment ['top'] The vertical alignment of the text

If the `fontproperties` keyword argument is given then the rcParams defaults for `fontsize` (`figure.titlesize`) and `fontweight` (`figure.titleweight`) will be ignored in favour of the `FontProperties` defaults.

A `matplotlib.text.Text` instance is returned.

Example:

```
fig.suptitle('this is the figure title', fontsize=12)
```

Examples using `matplotlib.pyplot.suptitle`

- [sphx_glr_gallery_subplots_axes_and_figures_figure_title.py](#)
- [sphx_glr_gallery_userdemo_demo_gridspec04.py](#)
- [Pyplot tutorial](#)

75.1.130 `matplotlib.pyplot.switch_backend`

`matplotlib.pyplot.switch_backend(newbackend)`

Switch the default backend. This feature is experimental, and is only expected to work switching to an image backend. e.g., if you have a bunch of PostScript scripts that you want to run from an interactive ipython session, you may want to switch to the PS backend before running them to avoid having a bunch of GUI windows popup. If you try to interactively switch from one GUI backend to another, you will explode.

Calling this command will close all open windows.

75.1.131 `matplotlib.pyplot.table`

`matplotlib.pyplot.table(**kwargs)`

Add a table to the current axes.

Call signature:

```
table(cellText=None, cellColours=None,  
      cellLoc='right', colWidths=None,  
      rowLabels=None, rowColours=None, rowLoc='left',  
      colLabels=None, colColours=None, colLoc='center',  
      loc='bottom', bbox=None)
```

Returns a `matplotlib.table.Table` instance. Either `cellText` or `cellColours` must be provided. For finer grained control over tables, use the `Table` class and add it to the axes with `add_table()`.

Thanks to John Gill for providing the class and table.

`kwargs` control the `Table` properties:

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha	float (0.0 transparent through 1.0 opaque)
animated	bool
clip_box	a Bbox instance
clip_on	bool
clip_path	[(Path, Transform) Patch None]
contains	a callable function
figure	a Figure instance
fontsize	a float in points
gid	an id string
label	object
path_effects	AbstractPathEffect
picker	[None bool float callable]
rasterized	bool or None
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
transform	Transform
url	a url string
visible	bool
zorder	float

Examples using `matplotlib.pyplot.table`

- `sphx_glr_gallery_text_labels_and_annotations_font_table_ttf_sgskip.py`
- `sphx_glr_gallery_misc_table_demo.py`

75.1.132 `matplotlib.pyplot.text`

`matplotlib.pyplot.text(x, y, s, fontdict=None, withdash=False, **kwargs)`

Add text to the axes.

Add the text `s` to the axes at location `x, y` in data coordinates.

Parameters

`x, y` [scalars] The position to place the text. By default, this is in data coordinates. The coordinate system can be changed using the `transform` parameter.

`s` [str] The text.

`fontdict` [dictionary, optional, default: None] A dictionary to override the default text properties. If `fontdict` is None, the defaults are determined by your rc parameters.

`withdash` [boolean, optional, default: False] Creates a `TextWithDash` instance instead of a `Text` instance.

Returns

`text` [Text] The created `Text` instance.

Other Parameters

`**kwargs` [Text properties.] Other miscellaneous text parameters.

Examples

Individual keyword arguments can be used to override any given parameter:

```
>>> text(x, y, s, fontsize=12)
```

The default transform specifies that text is in data coords, alternatively, you can specify text in axis coords (0,0 is lower-left and 1,1 is upper-right). The example below places text in the center of the axes:

```
>>> text(0.5, 0.5, 'matplotlib', horizontalalignment='center',  
...      verticalalignment='center', transform=ax.transAxes)
```

You can put a rectangular box around the text instance (e.g., to set a background color) by using the keyword `bbox`. `bbox` is a dictionary of `Rectangle` properties. For example:

```
>>> text(x, y, s, bbox=dict(facecolor='red', alpha=0.5))
```

Examples using `matplotlib.pyplot.text`

- `sphinx_glr_gallery_pyplots_pyplot_mathtext.py`
- `sphinx_glr_gallery_pyplots_pyplot_text.py`
- `sphinx_glr_gallery_shapes_and_collections_artist_reference.py`
- `sphinx_glr_gallery_text_labels_and_annotations_fancytextbox_demo.py`
- `sphinx_glr_gallery_text_labels_and_annotations_text_fontdict.py`
- `sphinx_glr_gallery_text_labels_and_annotations_autowrap.py`
- `sphinx_glr_gallery_text_labels_and_annotations_usetex_fonteffects.py`
- `sphinx_glr_gallery_text_labels_and_annotations_text_rotation_relative_to_line.py`
- `sphinx_glr_gallery_text_labels_and_annotations_multiline.py`
- `sphinx_glr_gallery_text_labels_and_annotations_stix_fonts_demo.py`
- `sphinx_glr_gallery_text_labels_and_annotations_fonts_demo_kw.py`
- `sphinx_glr_gallery_text_labels_and_annotations_usetex_demo.py`

- sphx_glr_gallery_text_labels_and_annotations_fonts_demo.py
- sphx_glr_gallery_text_labels_and_annotations_arrow_demo.py
- sphx_glr_gallery_showcase_integral.py
- sphx_glr_gallery_showcase_bachelors_degrees_by_gender.py
- sphx_glr_gallery_event_handling_close_event.py
- sphx_glr_gallery_misc_transoffset.py
- sphx_glr_gallery_specialty_plots_anscombe.py
- sphx_glr_gallery_userdemo_pgf_texsystem_sgskip.py
- sphx_glr_gallery_userdemo_pgf_fonts_sgskip.py
- [Pyplot tutorial](#)
- [Path effects guide](#)

75.1.133 matplotlib.pyplot.thetagrids

`matplotlib.pyplot.thetagrids(*args, **kwargs)`

Get or set the theta locations of the gridlines in a polar plot.

If no arguments are passed, return a tuple (lines, labels) where lines is an array of radial gridlines (Line2D instances) and labels is an array of tick labels (Text instances):

```
lines, labels = thetagrids()
```

Otherwise the syntax is:

```
lines, labels = thetagrids(angles, labels=None, fmt='%d', frac = 1.1)
```

set the angles at which to place the theta grids (these gridlines are equal along the theta dimension).

angles is in degrees.

labels, if not None, is a len(angles) list of strings of the labels to use at each angle.

If labels is None, the labels will be `fmt%angle`.

frac is the fraction of the polar axes radius at which to place the label (1 is the edge). e.g., 1.05 is outside the axes and 0.95 is inside the axes.

Return value is a list of tuples (lines, labels):

- lines are Line2D instances
- labels are Text instances.

Note that on input, the labels argument is a list of strings, and on output it is a list of Text instances.

Examples:

```
# set the locations of the radial gridlines and labels
lines, labels = thetagrids( range(45,360,90) )

# set the locations and labels of the radial gridlines and labels
lines, labels = thetagrids( range(45,360,90), ('NE', 'NW', 'SW','SE') )
```

75.1.134 matplotlib.pyplot.tick_params

`matplotlib.pyplot.tick_params(axis='both', **kwargs)`

Change the appearance of ticks, tick labels, and gridlines.

Parameters

`axis` [{`'x'`, `'y'`, `'both'`}, optional] Which axis to apply the parameters to.

Other Parameters

`axis` [{`'x'`, `'y'`, `'both'`}] Axis on which to operate; default is `'both'`.

`reset` [bool] If True, set all parameters to defaults before processing other keyword arguments. Default is False.

`which` [{`'major'`, `'minor'`, `'both'`}] Default is `'major'`; apply arguments to which ticks.

`direction` [{`'in'`, `'out'`, `'inout'`}] Puts ticks inside the axes, outside the axes, or both.

`length` [float] Tick length in points.

`width` [float] Tick width in points.

`color` [color] Tick color; accepts any mpl color spec.

`pad` [float] Distance in points between tick and label.

`labelsize` [float or str] Tick label font size in points or as a string (e.g., `'large'`).

`labelcolor` [color] Tick label color; mpl color spec.

`colors` [color] Changes the tick color and the label color to the same value: mpl color spec.

`zorder` [float] Tick and label zorder.

`bottom`, `top`, `left`, `right` [bool] Whether to draw the respective ticks.

`labelbottom`, `labeltop`, `labeledleft`, `labelright` [bool] Whether to draw the respective tick labels.

`labelrotation` [float] Tick label rotation

`grid_color` [color] Changes the gridline color to the given mpl color spec.

`grid_alpha` [float] Transparency of gridlines: 0 (transparent) to 1 (opaque).

`grid_linewidth` [float] Width of gridlines in points.

`grid_linestyle` [string] Any valid Line2D line style spec.

Examples

Usage

```
ax.tick_params(direction='out', length=6, width=2, colors='r',
               grid_color='r', grid_alpha=0.5)
```

This will make all major ticks be red, pointing out of the box, and with dimensions 6 points by 2 points. Tick labels will also be red. Gridlines will be red and translucent.

Examples using `matplotlib.pyplot.tick_params`

- `sphx_glr_gallery_showcase_bachelors_degrees_by_gender.py`

75.1.135 `matplotlib.pyplot.ticklabel_format`

`matplotlib.pyplot.ticklabel_format(**kwargs)`

Change the `ScalarFormatter` used by default for linear axes.

Optional keyword arguments:

Key-word	Description
<code>style</code>	['sci' (or 'scientific') 'plain'] plain turns off scientific notation
<code>scilim</code>	(m, n), pair of integers; if style is 'sci', scientific notation will be used for its numbers outside the range 10^m to 10^n . Use (0,0) to include all numbers.
<code>use-Offset</code>	[bool offset]; if True, the offset will be calculated as needed; if False, no offset will be used; if a numeric offset is specified, it will be used.
<code>axis</code>	['x' 'y' 'both']
<code>use-Lo-cale</code>	If True, format the number according to the current locale. This affects things such as the character used for the decimal separator. If False, use C-style (English) formatting. The default setting is controlled by the <code>axes.formatter.use_locale</code> rparam.
<code>use-Math-Text</code>	If True, render the offset and scientific notation in <code>mathtext</code>

Only the major ticks are affected. If the method is called when the `ScalarFormatter` is not the `Formatter` being used, an `AttributeError` will be raised.

75.1.136 `matplotlib.pyplot.tight_layout`

`matplotlib.pyplot.tight_layout(pad=1.08, h_pad=None, w_pad=None, rect=None)`

Automatically adjust subplot parameters to give specified padding.

Parameters

`pad` [float] padding between the figure edge and the edges of subplots, as a fraction of the font-size.

`h_pad, w_pad` [float] padding (height/width) between edges of adjacent subplots. Defaults to `pad_inches`.

`rect` [if rect is given, it is interpreted as a rectangle] (left, bottom, right, top) in the normalized figure coordinate that the whole subplots area (including labels) will fit into. Default is (0, 0, 1, 1).

Examples using `matplotlib.pyplot.tight_layout`

- `sphinx_glr_gallery_subplots_axes_and_figures_demo_tight_layout.py`
- `sphinx_glr_gallery_lines_bars_and_markers_nan_test.py`
- `sphinx_glr_gallery_lines_bars_and_markers_linestyles.py`
- `sphinx_glr_gallery_images_contours_and_fields_interpolation_methods.py`
- `sphinx_glr_gallery_images_contours_and_fields_plot_streamplot.py`
- `sphinx_glr_gallery_images_contours_and_fields_image_annotated_heatmap.py`
- `sphinx_glr_gallery_shapes_and_collections_artist_reference.py`
- `sphinx_glr_gallery_text_labels_and_annotations_figlegend_demo.py`
- `sphinx_glr_gallery_text_labels_and_annotations_engineering_formatter.py`
- `sphinx_glr_gallery_misc_zorder_demo.py`
- `sphinx_glr_gallery_mplot3d_wire3d_zero_stride.py`
- `sphinx_glr_gallery_scales_symlog_demo.py`
- `sphinx_glr_gallery_specialty_plots_mri_with_eeg.py`
- `sphinx_glr_gallery_userdemo_pgf_texsystem_sgskip.py`
- `sphinx_glr_gallery_userdemo_pgf_fonts_sgskip.py`
- `sphinx_glr_gallery_userdemo_pgf_preamble_sgskip.py`
- [Tight Layout guide](#)

75.1.137 matplotlib.pyplot.title

matplotlib.pyplot.title(s, *args, **kwargs)

Set a title of the current axes.

Set one of the three available axes titles. The available titles are positioned above the axes in the center, flush with the left edge, and flush with the right edge.

See also:

See text() for adding text to the current axes

Parameters

label [str] Text to use for the title

fontdict [dict] A dictionary controlling the appearance of the title text, the default fontdict is:

```
{'fontsize': rcParams['axes.titlesize'], 'fontweight' : rc-
Params['axes.titleweight'], 'verticalalignment': 'baseline', 'hori-
zontalalignment': 'center'}
```

loc [{ 'center', 'left', 'right' }, str, optional] Which title to set, defaults to 'center'

Returns

text [Text] The matplotlib text instance representing the title

Other Parameters

kwargs [text properties] Other keyword arguments are text properties, see Text for a list of valid text properties.

Examples using matplotlib.pyplot.title

- sphx_glr_gallery_api_sankey_basics.py
- sphx_glr_gallery_pyplots_pyplot_mathtext.py
- sphx_glr_gallery_pyplots_pyplot_text.py
- sphx_glr_gallery_pyplots_pyplot_scales.py
- sphx_glr_gallery_subplots_axes_and_figures_invert_axes.py
- sphx_glr_gallery_subplots_axes_and_figures_subplot.py
- sphx_glr_gallery_subplots_axes_and_figures_geo_demo.py
- sphx_glr_gallery_subplots_axes_and_figures_figure_title.py
- sphx_glr_gallery_subplots_axes_and_figures_axes_demo.py
- sphx_glr_gallery_lines_bars_and_markers_masked_demo.py

- sphx_glr_gallery_lines_bars_and_markers_nan_test.py
- sphx_glr_gallery_lines_bars_and_markers_bar_stacked.py
- sphx_glr_gallery_lines_bars_and_markers_psd_demo.py
- sphx_glr_gallery_text_labels_and_annotations_titles_demo.py
- sphx_glr_gallery_text_labels_and_annotations_text_fontdict.py
- sphx_glr_gallery_text_labels_and_annotations_usetex_fonteffects.py
- sphx_glr_gallery_text_labels_and_annotations_multiline.py
- sphx_glr_gallery_text_labels_and_annotations_font_table_ttf_sgskip.py
- sphx_glr_gallery_style_sheets_plot_solarizedlight2.py
- sphx_glr_gallery_showcase_xkcd.py
- sphx_glr_gallery_event_handling_trifinder_event_demo.py
- sphx_glr_gallery_event_handling_ginput_manual_clabel_sgskip.py
- sphx_glr_gallery_misc_tight_bbox_test.py
- sphx_glr_gallery_misc_set_and_get.py
- sphx_glr_gallery_misc_findobj_demo.py
- sphx_glr_gallery_misc_zorder_demo.py
- sphx_glr_gallery_misc_contour_manual.py
- sphx_glr_gallery_misc_multipage_pdf.py
- sphx_glr_gallery_misc_table_demo.py
- sphx_glr_gallery_scales_custom_scale.py
- sphx_glr_gallery_user_interfaces_svg_histogram_sgskip.py
- [Usage Guide](#)
- [Pyplot tutorial](#)

75.1.138 matplotlib.pyplot.tricontour

`matplotlib.pyplot.tricontour(*args, **kwargs)`

Draw contours on an unstructured triangular grid. `tricontour()` and `tricontourf()` draw contour lines and filled contours, respectively. Except as noted, function signatures and return values are the same for both versions.

The triangulation can be specified in one of two ways; either:

`tricontour(triangulation, ...)`

where `triangulation` is a `matplotlib.tri.Triangulation` object, or

```

tricontour(x, y, ...)
tricontour(x, y, triangles, ...)
tricontour(x, y, triangles=triangles, ...)
tricontour(x, y, mask=mask, ...)
tricontour(x, y, triangles, mask=mask, ...)

```

in which case a Triangulation object will be created. See Triangulation for a explanation of these possibilities.

The remaining arguments may be:

```
tricontour(..., Z)
```

where Z is the array of values to contour, one per point in the triangulation. The level values are chosen automatically.

```
tricontour(..., Z, N)
```

contour up to N+1 automatically chosen contour levels (N intervals).

```
tricontour(..., Z, V)
```

draw contour lines at the values specified in sequence V, which must be in increasing order.

```
tricontourf(..., Z, V)
```

fill the (len(V)-1) regions between the values in V, which must be in increasing order.

```
tricontour(Z, **kwargs)
```

Use keyword args to control colors, linewidth, origin, cmap ... see below for more details.

C = tricontour(...) returns a TriContourSet object.

Optional keyword arguments:

colors: [None | string | (mpl_colors)] If None, the colormap specified by cmap will be used.

If a string, like 'r' or 'red', all levels will be plotted in this color.

If a tuple of matplotlib color args (string, float, rgb, etc), different levels will be plotted in different colors in the order specified.

alpha: float The alpha blending value

cmap: [None | Colormap] A cm Colormap instance or None. If cmap is None and colors is None, a default Colormap is used.

norm: [None | Normalize] A matplotlib.colors.Normalize instance for scaling data values to colors. If norm is None and colors is None, the default linear scaling is used.

levels [level0, level1, ..., leveln] A list of floating point numbers indicating the level curves to draw, in increasing order; e.g., to draw just the zero contour pass levels=[0]

origin: [None | 'upper' | 'lower' | 'image'] If None, the first value of Z will correspond to the lower left corner, location (0,0). If 'image', the rc value for image.origin will be used.

This keyword is not active if X and Y are specified in the call to contour.

extent: [None | (x0,x1,y0,y1)]

If origin is not None, then extent is interpreted as in `matplotlib.pyplot.imshow()`: it gives the outer pixel boundaries. In this case, the position of `Z[0,0]` is the center of the pixel, not a corner. If origin is None, then (x0, y0) is the position of `Z[0,0]`, and (x1, y1) is the position of `Z[-1,-1]`.

This keyword is not active if X and Y are specified in the call to contour.

locator: [None | `ticker.Locator` subclass] If locator is None, the default `MaxNLocator` is used. The locator is used to determine the contour levels if they are not given explicitly via the V argument.

extend: ['neither' | 'both' | 'min' | 'max'] Unless this is 'neither', contour levels are automatically added to one or both ends of the range so that all data are included. These added ranges are then mapped to the special colormap values which default to the ends of the colormap range, but can be set via `matplotlib.colors.Colormap.set_under()` and `matplotlib.colors.Colormap.set_over()` methods.

xunits, yunits: [None | registered units] Override axis units by specifying an instance of a `matplotlib.units.ConversionInterface`.

tricontour-only keyword arguments:

linewidths: [None | number | tuple of numbers] If linewidths is None, the default width in `lines.linewidth` in `matplotlibrc` is used.

If a number, all levels will be plotted with this linewidth.

If a tuple, different levels will be plotted with different linewidths in the order specified

linestyles: [None | 'solid' | 'dashed' | 'dashdot' | 'dotted'] If linestyles is None, the 'solid' is used.

linestyles can also be an iterable of the above strings specifying a set of linestyles to be used. If this iterable is shorter than the number of contour levels it will be repeated as necessary.

If contour is using a monochrome colormap and the contour level is less than 0, then the linestyle specified in `contour.negative_linestyle` in `matplotlibrc` will be used.

tricontourf-only keyword arguments:

antialiased: bool enable antialiasing

Note: `tricontourf` fills intervals that are closed at the top; that is, for boundaries `z1` and `z2`, the filled region is:

```
z1 < z <= z2
```

There is one exception: if the lowest boundary coincides with the minimum value of the `z` array, then that minimum value will be included in the lowest interval.

Examples using `matplotlib.pyplot.tricontour`

- `sphx_glr_gallery_images_contours_and_fields_irregulardatagrid.py`
- `sphx_glr_gallery_images_contours_and_fields_tricontour_smooth_user.py`
- `sphx_glr_gallery_images_contours_and_fields_trigradient_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_tricontour_smooth_delaunay.py`

75.1.139 `matplotlib.pyplot.tricontourf`

`matplotlib.pyplot.tricontourf(*args, **kwargs)`

Draw contours on an unstructured triangular grid. `tricontour()` and `tricontourf()` draw contour lines and filled contours, respectively. Except as noted, function signatures and return values are the same for both versions.

The triangulation can be specified in one of two ways; either:

```
tricontour(triangulation, ...)
```

where `triangulation` is a `matplotlib.tri.Triangulation` object, or

```
tricontour(x, y, ...)
tricontour(x, y, triangles, ...)
tricontour(x, y, triangles=triangles, ...)
tricontour(x, y, mask=mask, ...)
tricontour(x, y, triangles, mask=mask, ...)
```

in which case a `Triangulation` object will be created. See `Triangulation` for a explanation of these possibilities.

The remaining arguments may be:

```
tricontour(..., Z)
```

where `Z` is the array of values to contour, one per point in the triangulation. The level values are chosen automatically.

```
tricontour(..., Z, N)
```

contour up to `N+1` automatically chosen contour levels (`N` intervals).

```
tricontour(..., Z, V)
```

draw contour lines at the values specified in sequence V, which must be in increasing order.

```
tricontourf(..., Z, V)
```

fill the (len(V)-1) regions between the values in V, which must be in increasing order.

```
tricontour(Z, **kwargs)
```

Use keyword args to control colors, linewidth, origin, cmap ... see below for more details.

C = tricontour(...) returns a TriContourSet object.

Optional keyword arguments:

colors: [None | string | (mpl_colors)] If None, the colormap specified by cmap will be used.

If a string, like 'r' or 'red', all levels will be plotted in this color.

If a tuple of matplotlib color args (string, float, rgb, etc), different levels will be plotted in different colors in the order specified.

alpha: float The alpha blending value

cmap: [None | Colormap] A cm Colormap instance or None. If cmap is None and colors is None, a default Colormap is used.

norm: [None | Normalize] A matplotlib.colors.Normalize instance for scaling data values to colors. If norm is None and colors is None, the default linear scaling is used.

levels [level0, level1, ..., leveln] A list of floating point numbers indicating the level curves to draw, in increasing order; e.g., to draw just the zero contour pass levels=[0]

origin: [None | 'upper' | 'lower' | 'image'] If None, the first value of Z will correspond to the lower left corner, location (0,0). If 'image', the rc value for image.origin will be used.

This keyword is not active if X and Y are specified in the call to contour.

extent: [None | (x0,x1,y0,y1)]

If origin is not None, then extent is interpreted as in matplotlib.pyplot.imshow(): it gives the outer pixel boundaries. In this case, the position of Z[0,0] is the center of the pixel, not a corner. If origin is None, then (x0, y0) is the position of Z[0,0], and (x1, y1) is the position of Z[-1,-1].

This keyword is not active if X and Y are specified in the call to contour.

locator: [None | ticker.Locator subclass] If locator is None, the default MaxNLocator is used. The locator is used to determine the contour levels if they are not given explicitly via the V argument.

extend: ['neither' | 'both' | 'min' | 'max'] Unless this is 'neither', contour levels are automatically added to one or both ends of the range so that all data are included. These added ranges are then mapped to the special colormap values which default to the ends of the colormap range, but can be set via `matplotlib.colors.Colormap.set_under()` and `matplotlib.colors.Colormap.set_over()` methods.

xunits, yunits: [None | registered units] Override axis units by specifying an instance of a `matplotlib.units.ConversionInterface`.

tricontour-only keyword arguments:

linewidths: [None | number | tuple of numbers] If linewidths is None, the default width in `lines.linewidth` in `matplotlibrc` is used.

If a number, all levels will be plotted with this linewidth.

If a tuple, different levels will be plotted with different linewidths in the order specified

linestyles: [None | 'solid' | 'dashed' | 'dashdot' | 'dotted'] If linestyles is None, the 'solid' is used.

linestyles can also be an iterable of the above strings specifying a set of linestyles to be used. If this iterable is shorter than the number of contour levels it will be repeated as necessary.

If contour is using a monochrome colormap and the contour level is less than 0, then the linestyle specified in `contour.negative_linestyle` in `matplotlibrc` will be used.

tricontourf-only keyword arguments:

antialiased: bool enable antialiasing

Note: `tricontourf` fills intervals that are closed at the top; that is, for boundaries `z1` and `z2`, the filled region is:

$$z1 < z \leq z2$$

There is one exception: if the lowest boundary coincides with the minimum value of the `z` array, then that minimum value will be included in the lowest interval.

Examples using `matplotlib.pyplot.tricontourf`

- `sphx_glr_gallery_images_contours_and_fields_irregulardatagrid.py`
- `sphx_glr_gallery_images_contours_and_fields_triinterp_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_tricontour_smooth_user.py`
- `sphx_glr_gallery_images_contours_and_fields_tricontour_demo.py`
- `sphx_glr_gallery_images_contours_and_fields_tricontour_smooth_delaunay.py`

75.1.140 matplotlib.pyplot.tripcolor

matplotlib.pyplot.tripcolor(*args, **kwargs)

Create a pseudocolor plot of an unstructured triangular grid.

The triangulation can be specified in one of two ways; either:

```
tripcolor(triangulation, ...)
```

where triangulation is a matplotlib.tri.Triangulation object, or

```
tripcolor(x, y, ...)
tripcolor(x, y, triangles, ...)
tripcolor(x, y, triangles=triangles, ...)
tripcolor(x, y, mask=mask, ...)
tripcolor(x, y, triangles, mask=mask, ...)
```

in which case a Triangulation object will be created. See Triangulation for a explanation of these possibilities.

The next argument must be C, the array of color values, either one per point in the triangulation if color values are defined at points, or one per triangle in the triangulation if color values are defined at triangles. If there are the same number of points and triangles in the triangulation it is assumed that color values are defined at points; to force the use of color values at triangles use the kwarg facecolors=C instead of just C.

shading may be 'flat' (the default) or 'gouraud'. If shading is 'flat' and C values are defined at points, the color values used for each triangle are from the mean C of the triangle's three points. If shading is 'gouraud' then color values must be defined at points.

The remaining kwargs are the same as for pcolor().

Examples using matplotlib.pyplot.tripcolor

- sphx_glr_gallery_images_contours_and_fields_tripcolor_demo.py

75.1.141 matplotlib.pyplot.triplot

matplotlib.pyplot.triplot(*args, **kwargs)

Draw a unstructured triangular grid as lines and/or markers.

The triangulation to plot can be specified in one of two ways; either:

```
triplot(triangulation, ...)
```

where triangulation is a matplotlib.tri.Triangulation object, or

```

triplot(x, y, ...)
triplot(x, y, triangles, ...)
triplot(x, y, triangles=triangles, ...)
triplot(x, y, mask=mask, ...)
triplot(x, y, triangles, mask=mask, ...)

```

in which case a Triangulation object will be created. See Triangulation for a explanation of these possibilities.

The remaining args and kwargs are the same as for plot().

Return a list of 2 Line2D containing respectively:

- the lines plotted for triangles edges
- the markers plotted for triangles nodes

Examples using matplotlib.pyplot.triplot

- sphx_glr_gallery_images_contours_and_fields_triinterp_demo.py
- sphx_glr_gallery_images_contours_and_fields_trigradient_demo.py
- sphx_glr_gallery_images_contours_and_fields_triplot_demo.py
- sphx_glr_gallery_images_contours_and_fields_tricontour_smooth_delaunay.py
- sphx_glr_gallery_event_handling_trifinder_event_demo.py

75.1.142 matplotlib.pyplot.twinx

matplotlib.pyplot.twinx(ax=None)

Make a second axes that shares the x-axis. The new axes will overlay ax (or the current axes if ax is None). The ticks for ax2 will be placed on the right, and the ax2 instance is returned.

See also:

examples/api_examples/two_scales.py For an example

75.1.143 matplotlib.pyplot.twinx

matplotlib.pyplot.twinx(ax=None)

Make a second axes that shares the y-axis. The new axis will overlay ax (or the current axes if ax is None). The ticks for ax2 will be placed on the top, and the ax2 instance is returned.

75.1.144 matplotlib.pyplot.uninstall_repl_displayhook

matplotlib.pyplot.uninstall_repl_displayhook()

Uninstalls the matplotlib display hook.

75.1.145 `matplotlib.pyplot.violinplot`

```
matplotlib.pyplot.violinplot(dataset, positions=None, vert=True, widths=0.5,  
                             showmeans=False, showextrema=True, showmedians=False,  
                             points=100, bw_method=None, hold=None, data=None)
```

Make a violin plot.

Make a violin plot for each column of dataset or each vector in sequence dataset. Each filled area extends to represent the entire data range, with optional lines at the mean, the median, the minimum, and the maximum.

Parameters

- `dataset` [Array or a sequence of vectors.] The input data.
- `positions` [array-like, default = [1, 2, ..., n]] Sets the positions of the violins. The ticks and limits are automatically set to match the positions.
- `vert` [bool, default = True.] If true, creates a vertical violin plot. Otherwise, creates a horizontal violin plot.
- `widths` [array-like, default = 0.5] Either a scalar or a vector that sets the maximal width of each violin. The default is 0.5, which uses about half of the available horizontal space.
- `showmeans` [bool, default = False] If `True`, will toggle rendering of the means.
- `showextrema` [bool, default = True] If `True`, will toggle rendering of the extrema.
- `showmedians` [bool, default = False] If `True`, will toggle rendering of the medians.
- `points` [scalar, default = 100] Defines the number of points to evaluate each of the gaussian kernel density estimations at.
- `bw_method` [str, scalar or callable, optional] The method used to calculate the estimator bandwidth. This can be 'scott', 'silverman', a scalar constant or a callable. If a scalar, this will be used directly as `kde.factor`. If a callable, it should take a `GaussianKDE` instance as its only parameter and return a scalar. If `None` (default), 'scott' is used.

Returns

- `result` [dict] A dictionary mapping each component of the violinplot to a list of the corresponding collection instances created. The dictionary has the following keys:
- `bodies`: A list of the `matplotlib.collections.PolyCollection` instances containing the filled area of each violin.
 - `cmeans`: A `matplotlib.collections.LineCollection` instance created to identify the mean values of each of the violin's distribution.
 - `cmns`: A `matplotlib.collections.LineCollection` instance created to identify the bottom of each violin's distribution.

- `cmaxes`: A `matplotlib.collections.LineCollection` instance created to identify the top of each violin's distribution.
- `cbars`: A `matplotlib.collections.LineCollection` instance created to identify the centers of each violin's distribution.
- `cmedians`: A `matplotlib.collections.LineCollection` instance created to identify the median values of each of the violin's distribution.

Notes

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: 'dataset'.
-

75.1.146 `matplotlib.pyplot.viridis`

`matplotlib.pyplot.viridis()`

Set the colormap to "viridis".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.147 `matplotlib.pyplot.vlines`

`matplotlib.pyplot.vlines(x, ymin, ymax, colors='k', linestyle='solid', label="", hold=None, data=None, **kwargs)`

Plot vertical lines.

Plot vertical lines at each `x` from `ymin` to `ymax`.

Parameters

`x` [scalar or 1D array_like] x-indexes where to plot the lines.

`ymin, ymax` [scalar or 1D array_like] Respective beginning and end of each line. If scalars are provided, all lines will have same length.

`colors` [array_like of colors, optional, default: 'k']

`linestyle` [['solid' | 'dashed' | 'dashdot' | 'dotted'], optional]

`label` [string, optional, default: '']

Returns

`lines` [LineCollection]

Other Parameters

`**kwargs` [LineCollection properties.]

See also:

`hlines` horizontal lines

`axvline` vertical line across the axes

In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`: * All arguments with the following names: 'colors', 'x', 'ymax', 'ymin'.

75.1.148 `matplotlib.pyplot.waitforbuttonpress`

`matplotlib.pyplot.waitforbuttonpress(*args, **kwargs)`

Blocking call to interact with the figure.

This will return True if a key was pressed, False if a mouse button was pressed and None if timeout was reached without either being pressed.

If timeout is negative, does not timeout.

Examples using `matplotlib.pyplot.waitforbuttonpress`

- `sphx_glr_gallery_event_handling_ginput_manual_clabel_sgskip.py`

75.1.149 `matplotlib.pyplot.winter`

`matplotlib.pyplot.winter()`

Set the colormap to "winter".

This changes the default colormap as well as the colormap of the current image if there is one. See `help(colormaps)` for more information.

75.1.150 `matplotlib.pyplot.xcorr`

`matplotlib.pyplot.xcorr(x, y, normed=True, detrend=<function detrend_none>, usev-
lines=True, maxlags=10, hold=None, data=None, **kwargs)`

Plot the cross correlation between x and y.

The correlation with lag k is defined as $\sum_n x[n+k] * \text{conj}(y[n])$.

Parameters

`x` [sequence of scalars of length n]

`y` [sequence of scalars of length n]

`hold` [bool, optional, deprecated, default: True]

`detrend` [callable, optional, default: `mlab.detrend_none`] `x` is detrended by the `detrend` callable. Default is no normalization.

`normed` [bool, optional, default: `True`] If `True`, input vectors are normalised to unit length.

`usevlines` [bool, optional, default: `True`] If `True`, `Axes.vlines` is used to plot the vertical lines from the origin to the `acorr`. Otherwise, `Axes.plot` is used.

`maxlags` [int, optional] Number of lags to show. If `None`, will return all $2 * \text{len}(x) - 1$ lags. Default is 10.

Returns

`lags` [array (length $2 * \text{maxlags} + 1$)] lag vector.

`c` [array (length $2 * \text{maxlags} + 1$)] auto correlation vector.

`line` [LineCollection or Line2D] Artist added to the axes of the correlation
LineCollection if `usevlines` is `True` Line2D if `usevlines` is `False`

`b` [Line2D or None] Horizontal line at 0 if `usevlines` is `True` None `usevlines` is `False`

Other Parameters

`linestyle` [Line2D property, optional] Only used if `usevlines` is `False`.

`marker` [string, optional] Default is `'o'`.

Notes

The cross correlation is performed with `numpy.correlate()` with `mode = 2`.

Note: In addition to the above described arguments, this function can take a data keyword argument. If such a data argument is given, the following arguments are replaced by `data[<arg>]`:

- All arguments with the following names: `'x'`, `'y'`.
-

75.1.151 matplotlib.pyplot.xkcd

`matplotlib.pyplot.xkcd(scale=1, length=100, randomness=2)`

Turns on `xkcd` sketch-style drawing mode. This will only have effect on things drawn after this function is called.

For best results, the "Humor Sans" font should be installed: it is not included with matplotlib.

Parameters

`scale` [float, optional] The amplitude of the wiggle perpendicular to the source line.

`length` [float, optional] The length of the wiggle along the line.

`randomness` [float, optional] The scale factor by which the length is shrunken or expanded.

Notes

This function works by a number of rcParams, so it will probably override others you have set before.

If you want the effects of this function to be temporary, it can be used as a context manager, for example:

```
with plt.xkcd():
    # This figure will be in XKCD-style
    fig1 = plt.figure()
    # ...

# This figure will be in regular style
fig2 = plt.figure()
```

Examples using matplotlib.pyplot.xkcd

- sphx_glr_gallery_showcase_xkcd.py

75.1.152 matplotlib.pyplot.xlabel

`matplotlib.pyplot.xlabel(s, *args, **kwargs)`

Set the x-axis label of the current axes.

Call signature:

```
xlabel(label, fontdict=None, labelpad=None, **kwargs)
```

This is the pyplot equivalent of calling `set_xlabel` on the current axes. See there for a full parameter description.

Examples using matplotlib.pyplot.xlabel

- sphx_glr_gallery_pyplots_pyplot_mathtext.py
- sphx_glr_gallery_pyplots_pyplot_text.py
- sphx_glr_gallery_subplots_axes_and_figures_invert_axes.py
- sphx_glr_gallery_subplots_axes_and_figures_subplot.py
- sphx_glr_gallery_subplots_axes_and_figures_figure_title.py

- sphx_glr_gallery_subplots_axes_and_figures_axes_demo.py
- sphx_glr_gallery_lines_bars_and_markers_scatter_symbol.py
- sphx_glr_gallery_lines_bars_and_markers_nan_test.py
- sphx_glr_gallery_text_labels_and_annotations_text_fontdict.py
- sphx_glr_gallery_text_labels_and_annotations_multiline.py
- sphx_glr_gallery_style_sheets_plot_solarizedlight2.py
- sphx_glr_gallery_showcase_xkcd.py
- sphx_glr_gallery_misc_findobj_demo.py
- sphx_glr_gallery_misc_plotfile_demo.py
- sphx_glr_gallery_scales_custom_scale.py
- sphx_glr_gallery_userdemo_pgf_texsystem_sgskip.py
- sphx_glr_gallery_userdemo_pgf_fonts_sgskip.py
- sphx_glr_gallery_userdemo_pgf_preamble_sgskip.py
- [Usage Guide](#)
- [Pyplot tutorial](#)

75.1.153 matplotlib.pyplot.xlim

matplotlib.pyplot.xlim(*args, **kwargs)

Get or set the x limits of the current axes.

Call signatures:

```
xmin, xmax = xlim() # return the current xlim
xlim((xmin, xmax)) # set the xlim to xmin, xmax
xlim(xmin, xmax)   # set the xlim to xmin, xmax
```

If you do not specify args, you can pass xmin or xmax as kwargs, i.e.:

```
xlim(xmax=3) # adjust the max leaving min unchanged
xlim(xmin=1) # adjust the min leaving max unchanged
```

Setting limits turns autoscaling off for the x-axis.

Returns

xmin, xmax A tuple of the new x-axis limits.

Notes

Calling this function with no arguments (e.g. `xlim()`) is the pyplot equivalent of calling `get_xlim` on the current axes. Calling this function with arguments is the pyplot equivalent of calling `set_xlim` on the current axes. All arguments are passed though.

Examples using `matplotlib.pyplot.xlim`

- `sphx_glr_gallery_subplots_axes_and_figures_invert_axes.py`
- `sphx_glr_gallery_subplots_axes_and_figures_shared_axis_demo.py`
- `sphx_glr_gallery_subplots_axes_and_figures_axes_demo.py`
- `sphx_glr_gallery_lines_bars_and_markers_step_demo.py`
- `sphx_glr_gallery_lines_bars_and_markers_errorbar_limits_simple.py`
- `sphx_glr_gallery_shapes_and_collections_ellipse_demo.py`
- `sphx_glr_gallery_text_labels_and_annotations_usetex_fonteffects.py`
- `sphx_glr_gallery_text_labels_and_annotations_text_rotation_relative_to_line.py`
- `sphx_glr_gallery_text_labels_and_annotations_arrow_demo.py`
- Frame grabbing

75.1.154 `matplotlib.pyplot.xscale`

`matplotlib.pyplot.xscale(*args, **kwargs)`

Set the scaling of the x-axis.

Call signature:

`xscale(scale, **kwargs)`

Parameters

`scale` `[['linear' | 'log' | 'logit' | 'symlog']]` The scaling type.

`**kwargs` Additional parameters depend on scale. See Notes.

Notes

This is the pyplot equivalent of calling `set_xscale` on the current axes.

Different keywords may be accepted, depending on the scale:

`'linear'`

`'log'`

basex/basey: The base of the logarithm

nonposx/nonposy: ['mask' | 'clip'] non-positive values in x or y can be masked as invalid, or clipped to a very small positive number

subsx/subsy: Where to place the subticks between each major tick. Should be a sequence of integers. For example, in a log10 scale: [2, 3, 4, 5, 6, 7, 8, 9]

will place 8 logarithmically spaced minor ticks between each major tick.

'logit'

nonpos: ['mask' | 'clip'] values beyond]0, 1[can be masked as invalid, or clipped to a number very close to 0 or 1

'symlog'

basex/basey: The base of the logarithm

linthreshx/linthreshy: A single float which defines the range (-x, x), within which the plot is linear. This avoids having the plot go to infinity around zero.

subsx/subsy: Where to place the subticks between each major tick. Should be a sequence of integers. For example, in a log10 scale: [2, 3, 4, 5, 6, 7, 8, 9]

will place 8 logarithmically spaced minor ticks between each major tick.

linscalex/linscaley: This allows the linear range (-linthresh to linthresh) to be stretched relative to the logarithmic range. Its value is the number of decades to use for each half of the linear range. For example, when linscale == 1.0 (the default), the space used for the positive and negative halves of the linear range will be equal to one decade in the logarithmic range.

Examples using `matplotlib.pyplot.xscale`

- `sphx_glr_gallery_scales_symlog_demo.py`

75.1.155 `matplotlib.pyplot.xticks`

`matplotlib.pyplot.xticks(*args, **kwargs)`

Get or set the current tick locations and labels of the x-axis.

Call signatures:

```
locs, labels = xticks()           # Get locations and labels
xticks(locs, [labels], **kwargs) # Set locations and labels
```


Parameters

`locs` [array_like] A list of positions at which ticks should be placed. You can pass an empty list to disable xticks.

`labels` [array_like, optional] A list of explicit labels to place at the given locs.

****kwargs** Text properties can be used to control the appearance of the labels.

Returns

`locs` An array of label locations.

`labels` A list of Text objects.

Notes

Calling this function with no arguments (e.g. `xticks()`) is the pyplot equivalent of calling `get_xticks` and `get_xticklabels` on the current axes. Calling this function with arguments is the pyplot equivalent of calling `set_xticks` and `set_xticklabels` on the current axes.

Examples

Get the current locations and labels:

```
>>> locs, labels = xticks()
```

Set label locations:

```
>>> xticks(np.arange(0, 1, step=0.2))
```

Set text labels:

```
>>> xticks(np.arange(5), ('Tom', 'Dick', 'Harry', 'Sally', 'Sue'))
```

Set text labels and properties:

```
>>> xticks(np.arange(12), calendar.month_name[1:13], rotation=20)
```

Disable xticks:

```
>>> xticks([])
```

Examples using `matplotlib.pyplot.xticks`

- `sphx_glr_gallery_subplots_axes_and_figures_axes_demo.py`
- `sphx_glr_gallery_lines_bars_and_markers_bar_stacked.py`
- `sphx_glr_gallery_lines_bars_and_markers_linestyles.py`

- sphx_glr_gallery_text_labels_and_annotations_multiline.py
- sphx_glr_gallery_text_labels_and_annotations_usetex_demo.py
- sphx_glr_gallery_text_labels_and_annotations_arrow_demo.py
- sphx_glr_gallery_showcase_xkcd.py
- sphx_glr_gallery_showcase_bachelors_degrees_by_gender.py
- sphx_glr_gallery_axes_grid1_demo_colorbar_of_inset_axes.py
- sphx_glr_gallery_misc_table_demo.py
- sphx_glr_gallery_ticks_and_spines_ticklabels_rotation.py
- sphx_glr_gallery_ticks_and_spines_custom_ticker1.py

75.1.156 matplotlib.pyplot.ylabel

matplotlib.pyplot.ylabel(s, *args, **kwargs)

Set the y-axis label of the current axes.

Call signature:

```
ylabel(label, fontdict=None, labelpad=None, **kwargs)
```

This is the pyplot equivalent of calling `set_ylabel` on the current axes. See there for a full parameter description.

Examples using matplotlib.pyplot.ylabel

- sphx_glr_gallery_pyplots_pyplot_simple.py
- sphx_glr_gallery_pyplots_pyplot_mathtext.py
- sphx_glr_gallery_pyplots_pyplot_text.py
- sphx_glr_gallery_subplots_axes_and_figures_invert_axes.py
- sphx_glr_gallery_subplots_axes_and_figures_subplot.py
- sphx_glr_gallery_subplots_axes_and_figures_figure_title.py
- sphx_glr_gallery_subplots_axes_and_figures_axes_demo.py
- sphx_glr_gallery_lines_bars_and_markers_scatter_symbol.py
- sphx_glr_gallery_lines_bars_and_markers_nan_test.py
- sphx_glr_gallery_lines_bars_and_markers_bar_stacked.py
- sphx_glr_gallery_text_labels_and_annotations_text_fontdict.py
- sphx_glr_gallery_text_labels_and_annotations_multiline.py
- sphx_glr_gallery_text_labels_and_annotations_usetex_demo.py

- sphx_glr_gallery_style_sheets_plot_solarizedlight2.py
- sphx_glr_gallery_showcase_xkcd.py
- sphx_glr_gallery_misc_tight_bbox_test.py
- sphx_glr_gallery_misc_findobj_demo.py
- sphx_glr_gallery_misc_plotfile_demo.py
- sphx_glr_gallery_misc_table_demo.py
- sphx_glr_gallery_scales_symlog_demo.py
- sphx_glr_gallery_scales_custom_scale.py
- sphx_glr_gallery_userdemo_pgf_preamble_sgskip.py
- [Usage Guide](#)
- [Pyplot tutorial](#)

75.1.157 matplotlib.pyplot.ylim

matplotlib.pyplot.ylim(*args, **kwargs)

Get or set the y-limits of the current axes.

Call signatures:

```
ymin, ymax = ylim() # return the current ylim
ylim((ymin, ymax)) # set the ylim to ymin, ymax
ylim(ymin, ymax)   # set the ylim to ymin, ymax
```

If you do not specify args, you can alternatively pass ymin or ymax as kwargs, i.e.:

```
ylim(ymax=3) # adjust the max leaving min unchanged
ylim(ymin=1) # adjust the min leaving max unchanged
```

Setting limits turns autoscaling off for the y-axis.

Returns

ymin, ymax A tuple of the new y-axis limits.

Notes

Calling this function with no arguments (e.g. `ylim()`) is the pyplot equivalent of calling `get_ylim` on the current axes. Calling this function with arguments is the pyplot equivalent of calling `set_ylim` on the current axes. All arguments are passed though.

Examples using `matplotlib.pyplot.ylim`

- `sphinx_glr_gallery_lines_bars_and_markers_step_demo.py`
- `sphinx_glr_gallery_lines_bars_and_markers_errorbar_limits_simple.py`
- `sphinx_glr_gallery_shapes_and_collections_ellipse_demo.py`
- `sphinx_glr_gallery_text_labels_and_annotations_usetex_fonteffects.py`
- `sphinx_glr_gallery_text_labels_and_annotations_arrow_demo.py`
- `sphinx_glr_gallery_showcase_integral.py`
- Frame grabbing
- `sphinx_glr_gallery_misc_findobj_demo.py`
- [Pyplot tutorial](#)

75.1.158 `matplotlib.pyplot.yscale`

`matplotlib.pyplot.yscale(*args, **kwargs)`

Set the scaling of the y-axis.

Call signature:

```
yscale(scale, **kwargs)
```

Parameters

`scale` `[['linear' | 'log' | 'logit' | 'symlog']]` The scaling type.

`**kwargs` Additional parameters depend on scale. See Notes.

Notes

This is the pyplot equivalent of calling `set_yscale` on the current axes.

Different keywords may be accepted, depending on the scale:

`'linear'`

`'log'`

`basex/basey`: The base of the logarithm

`nonposx/nonposy`: `['mask' | 'clip']` non-positive values in x or y can be masked as invalid, or clipped to a very small positive number

`subsx/subsy`: Where to place the subticks between each major tick. Should be a sequence of integers. For example, in a log10 scale: `[2, 3, 4, 5, 6, 7, 8, 9]`

will place 8 logarithmically spaced minor ticks between each major tick.

'logit'

nonpos: ['mask' | 'clip'] values beyond]0, 1[can be masked as invalid, or clipped to a number very close to 0 or 1

'symlog'

basex/basey: The base of the logarithm

linthreshx/linthreshy: A single float which defines the range (-x, x), within which the plot is linear. This avoids having the plot go to infinity around zero.

subsx/subsy: Where to place the subticks between each major tick. Should be a sequence of integers. For example, in a log10 scale: [2, 3, 4, 5, 6, 7, 8, 9]

will place 8 logarithmically spaced minor ticks between each major tick.

linscalex/linscaley: This allows the linear range (-linthresh to linthresh) to be stretched relative to the logarithmic range. Its value is the number of decades to use for each half of the linear range. For example, when linscale == 1.0 (the default), the space used for the positive and negative halves of the linear range will be equal to one decade in the logarithmic range.

Examples using `matplotlib.pyplot.yscale`

- [sphx_glr_gallery_pyplots_pyplot_scales.py](#)
- [sphx_glr_gallery_scales_symlog_demo.py](#)
- [Pyplot tutorial](#)

75.1.159 `matplotlib.pyplot.yticks`

`matplotlib.pyplot.yticks(*args, **kwargs)`

Get or set the current tick locations and labels of the y-axis.

Call signatures:

<pre>locs, labels = yticks() # Get locations and labels</pre> <pre>yticks(locs, [labels], **kwargs) # Set locations and labels</pre>
--

Parameters

`locs` [array_like] A list of positions at which ticks should be placed. You can pass an empty list to disable yticks.

`labels` [array_like, optional] A list of explicit labels to place at the given locs.

****kwargs** Text properties can be used to control the appearance of the labels.

Returns

locs An array of label locations.

labels A list of Text objects.

Notes

Calling this function with no arguments (e.g. `yticks()`) is the pyplot equivalent of calling `get_yticks` and `get_yticklabels` on the current axes. Calling this function with arguments is the pyplot equivalent of calling `set_yticks` and `set_yticklabels` on the current axes.

Examples

Get the current locations and labels:

```
>>> locs, labels = yticks()
```

Set label locations:

```
>>> yticks(np.arange(0, 1, step=0.2))
```

Set text labels:

```
>>> yticks(np.arange(5), ('Tom', 'Dick', 'Harry', 'Sally', 'Sue'))
```

Set text labels and properties:

```
>>> yticks(np.arange(12), calendar.month_name[1:13], rotation=45)
```

Disable yticks:

```
>>> yticks([])
```

Examples using `matplotlib.pyplot.yticks`

- `sphx_glr_gallery_subplots_axes_and_figures_axes_demo.py`
- `sphx_glr_gallery_lines_bars_and_markers_bar_stacked.py`
- `sphx_glr_gallery_lines_bars_and_markers_linestyles.py`
- `sphx_glr_gallery_text_labels_and_annotations_stix_fonts_demo.py`
- `sphx_glr_gallery_text_labels_and_annotations_usetex_demo.py`
- `sphx_glr_gallery_text_labels_and_annotations_arrow_demo.py`
- `sphx_glr_gallery_showcase_xkcd.py`

- sphx_glr_gallery_showcase_bachelors_degrees_by_gender.py
- sphx_glr_gallery_axes_grid1_demo_colorbar_of_inset_axes.py
- sphx_glr_gallery_misc_tight_bbox_test.py
- sphx_glr_gallery_misc_table_demo.py

76.1 mplot3d API

Contents

- mplot3d API
 - axes3d
 - axis3d
 - art3d
 - Art3D Utility Functions
 - proj3d

76.1.1 axes3d

Note: Significant effort went into bringing axes3d to feature-parity with regular axes objects for version 1.1.0. However, more work remains. Please report any functions that do not behave as expected as a bug. In addition, help and patches would be greatly appreciated!

<code>axes3d.Axes3D(fig[, rect])</code>	3D axes object.
---	-----------------

`mpl_toolkits.mplot3d.axes3d.Axes3D`

`class mpl_toolkits.mplot3d.axes3d.Axes3D(fig, rect=None, *args, **kwargs)`

3D axes object.

`add_collection3d(col, zs=0, zdir='z')`

Add a 3D collection object to the plot.

2D collection types are converted to a 3D version by modifying the object and adding z

coordinate information.

Supported are:

- PolyCollection
- LineCollection
- PatchCollection

`add_contour__set(cset, extend3d=False, stride=5, zdir='z', offset=None)`

`add_contourf__set(cset, zdir='z', offset=None)`

`auto_scale_xyz(X, Y, Z=None, had_data=None)`

`autoscale(enable=True, axis='both', tight=None)`

Convenience method for simple axis view autoscaling. See `matplotlib.axes.Axes.autoscale()` for full explanation. Note that this function behaves the same, but for all three axes. Therefore, 'z' can be passed for axis, and 'both' applies to all three axes.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

`autoscale_view(tight=None, scalex=True, scaley=True, scalez=True)`

Autoscale the view limits using the data limits. See `matplotlib.axes.Axes.autoscale_view()` for documentation. Note that this function applies to the 3D axes, and as such adds the `scalez` to the function arguments.

Changed in version 1.1.0: Function signature was changed to better match the 2D version. `tight` is now explicitly a kwarg and placed first.

Changed in version 1.2.1: This is now fully functional.

`bar(left, height, zs=0, zdir='z', *args, **kwargs)`

Add 2D `bar(s)`.

Argument	Description
<code>left</code>	The x coordinates of the left sides of the bars.
<code>height</code>	The height of the bars.
<code>zs</code>	Z coordinate of bars, if one value is specified they will all be placed at the same z.
<code>zdir</code>	Which direction to use as z ('x', 'y' or 'z') when plotting a 2D set.

Keyword arguments are passed onto `bar()`.

Returns a `Patch3DCollection`

`bar3d(x, y, z, dx, dy, dz, color=None, zsort='average', shade=True, *args, **kwargs)`

Generate a 3D barplot.

This method creates three dimensional barplot where the width, depth, height, and color of the bars can all be uniquely set.

Parameters

`x, y, z` [array-like] The coordinates of the anchor point of the bars.

`dx, dy, dz` [scalar or array-like] The width, depth, and height of the bars, respectively.

`color` [sequence of valid color specifications, optional] The color of the bars can be specified globally or individually. This parameter can be:

- A single color value, to color all bars the same color.
- An array of colors of length N bars, to color each bar independently.
- An array of colors of length 6, to color the faces of the bars similarly.
- An array of colors of length 6 * N bars, to color each face independently.

When coloring the faces of the boxes specifically, this is the order of the coloring:

1. -Z (bottom of box)
2. +Z (top of box)
3. -Y
4. +Y
5. -X
6. +X

`zsort` [str, optional] The z-axis sorting scheme passed onto `Poly3DCollection()`

`shade` [bool, optional (default = True)] When true, this shades the dark sides of the bars (relative to the plot's source of light).

Any additional keyword arguments are passed onto

:func:'~mpl_toolkits.mplot3d.art3d.Poly3DCollection'

Returns

`collection` [Poly3DCollection] A collection of three dimensional polygons representing the bars.

`can_pan()`

Return True if this axes supports the pan/zoom button functionality.

3D axes objects do not use the pan/zoom button.

`can_zoom()`

Return True if this axes supports the zoom box button functionality.

3D axes objects do not use the zoom box button.

`cla()`

Clear axes

`clabel(*args, **kwargs)`

This function is currently not implemented for 3D axes. Returns None.

`contour(X, Y, Z, *args, **kwargs)`

Create a 3D contour plot.

Argument	Description
X, Y, Z	Data values as <code>numpy.array</code> s
extend3d	Whether to extend contour in 3D (default: False)
stride	Stride (step size) for extending contour
zdir	The direction to use: x, y or z (default)
offset	If specified plot a projection of the contour lines on this position in plane normal to zdir

The positional and other keyword arguments are passed on to `contour()`

Returns a contour

`contour3D(X, Y, Z, *args, **kwargs)`

Create a 3D contour plot.

Argument	Description
X, Y, Z	Data values as <code>numpy.array</code> s
extend3d	Whether to extend contour in 3D (default: False)
stride	Stride (step size) for extending contour
zdir	The direction to use: x, y or z (default)
offset	If specified plot a projection of the contour lines on this position in plane normal to zdir

The positional and other keyword arguments are passed on to `contour()`

Returns a contour

`contourf(X, Y, Z, *args, **kwargs)`

Create a 3D contourf plot.

Argument	Description
X, Y,	Data values as numpy.arrays
Z	
zdir	The direction to use: x, y or z (default)
offset	If specified plot a projection of the filled contour on this position in plane normal to zdir

The positional and keyword arguments are passed on to `contourf()`

Returns a `contourf`

Changed in version 1.1.0: The `zdir` and `offset` kwargs were added.

`contourf3D(X, Y, Z, *args, **kwargs)`

Create a 3D `contourf` plot.

Argument	Description
X, Y,	Data values as numpy.arrays
Z	
zdir	The direction to use: x, y or z (default)
offset	If specified plot a projection of the filled contour on this position in plane normal to zdir

The positional and keyword arguments are passed on to `contourf()`

Returns a `contourf`

Changed in version 1.1.0: The `zdir` and `offset` kwargs were added.

`convert_zunits(z)`

For artists in an axes, if the zaxis has units support, convert z using zaxis unit type

New in version 1.2.1.

`disable_mouse_rotation()`

Disable mouse button callbacks.

`draw(renderer)`

Draw everything (plot lines, axes, labels)

`format_coord(xd, yd)`

Given the 2D view coordinates attempt to guess a 3D coordinate. Looks for the nearest edge to the point and then assumes that the point is at the same z location as the nearest point on the edge.

`format_zdata(z)`

Return z string formatted. This function will use the `fmt_zdata` attribute if it is callable, else will fall back on the zaxis major formatter

`get_autoscale_on()`

Get whether autoscaling is applied for all axes on plot commands

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

`get_autoscalez_on()`

Get whether autoscaling for the z-axis is applied on plot commands

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

`get_axis_position()`

`get_axisbelow()`

Get whether axis below is true or not.

For axes3d objects, this will always be True

New in version 1.1.0: This function was added for completeness.

`get_children()`

return a list of child artists

`get_frame_on()`

Get whether the 3D axes panels are drawn.

New in version 1.1.0.

`get_proj()`

Create the projection matrix from the current viewing position.

elev stores the elevation angle in the z plane
azim stores the azimuth angle in the x,y plane

dist is the distance of the eye viewing point from the object point.

`get_w_lims()`

Get 3D world limits.

`get_xlim()`

Get the x-axis range

Returns

xlimits [tuple] Returns the current x-axis limits as the tuple (left, right).

Notes

The x-axis may be inverted, in which case the left value will be greater than the right value.

Changed in version 1.1.0: This function now correctly refers to the 3D x-limits

`get_xlim3d()`

Get the x-axis range

Returns

`xlims` [tuple] Returns the current x-axis limits as the tuple (left, right).

Notes

The x-axis may be inverted, in which case the left value will be greater than the right value.

Changed in version 1.1.0: This function now correctly refers to the 3D x-limits

`get_ylim()`

Get the y-axis range

Returns

`ylimits` [tuple] Returns the current y-axis limits as the tuple (bottom, top).

Notes

The y-axis may be inverted, in which case the bottom value will be greater than the top value.

Changed in version 1.1.0: This function now correctly refers to the 3D y-limits.

`get_ylim3d()`

Get the y-axis range

Returns

`ylimits` [tuple] Returns the current y-axis limits as the tuple (bottom, top).

Notes

The y-axis may be inverted, in which case the bottom value will be greater than the top value.

Changed in version 1.1.0: This function now correctly refers to the 3D y-limits.

`get_zbound()`

Returns the z-axis numerical bounds where:

<code>lowerBound < upperBound</code>

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

`get_zlabel()`

Get the z-label text string.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

`get_zlim()`

Get 3D z limits.

`get_zlim3d()`

Get 3D z limits.

`get_zmajorticklabels()`

Get the ztick labels as a list of Text instances

New in version 1.1.0.

`get_zminorticklabels()`

Get the ztick labels as a list of Text instances

Note: Minor ticks are not supported. This function was added only for completeness.

New in version 1.1.0.

`get_zscale()`

`get_zticklabels(minor=False)`

Get ztick labels as a list of Text instances. See `matplotlib.axes.Axes.get_yticklabels()` for more details.

Note: Minor ticks are not supported.

New in version 1.1.0.

`get_zticklines()`

Get ztick lines as a list of Line2D instances. Note that this function is provided merely for completeness. These lines are re-calculated as the display changes.

New in version 1.1.0.

`get_zticks(minor=False)`

Return the z ticks as a list of locations See `matplotlib.axes.Axes.get_yticks()` for more details.

Note: Minor ticks are not supported.

New in version 1.1.0.

`grid(b=True, **kwargs)`

Set / unset 3D grid.

Note: Currently, this function does not behave the same as `matplotlib.axes.Axes.grid()`, but it is intended to eventually support that behavior.

Changed in version 1.1.0: This function was changed, but not tested. Please report any bugs.

`have_units()`

Return True if units are set on the x, y, or z axes

`invert_zaxis()`

Invert the z-axis.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

`locator_params(axis='both', tight=None, **kwargs)`

Convenience method for controlling tick locators.

See `matplotlib.axes.Axes.locator_params()` for full documentation Note that this is for `Axes3D` objects, therefore, setting axis to 'both' will result in the parameters being set for all three axes. Also, axis can also take a value of 'z' to apply parameters to the z axis.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

`margins(*args, **kw)`

Convenience method to set or retrieve autoscaling margins.

signatures:: `margins()`

returns `xmargin`, `ymargin`, `zmargin`

`margins(margin)`

`margins(xmargin, ymargin, zmargin)`

`margins(x=xmargin, y=ymargin, z=zmargin)`

`margins(..., tight=False)`

All forms above set the `xmargin`, `ymargin` and `zmargin` parameters. All keyword parameters are optional. A single argument specifies `xmargin`, `ymargin` and `zmargin`. The `tight` parameter is passed to `autoscale_view()`, which is executed after a margin is changed; the default here is `True`, on the assumption that when margins are specified, no additional padding to match tick marks is usually desired. Setting `tight` to `None` will preserve the previous setting.

Specifying any margin changes only the autoscaling; for example, if `xmargin` is not `None`, then `xmargin` times the X data interval will be added to each end of that interval before it is used in autoscaling.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

`mouse_init(rotate_btn=1, zoom_btn=3)`

Initializes mouse button callbacks to enable 3D rotation of the axes. Also optionally sets the mouse buttons for 3D rotation and zooming.

Argument	Description
rotate_btn	The integer or list of integers specifying which mouse button or buttons to use for 3D rotation of the axes. Default = 1.
zoom_btn	The integer or list of integers specifying which mouse button or buttons to use to zoom the 3D axes. Default = 3.

name = '3d'

plot(xs, ys, *args, **kwargs)
Plot 2D or 3D data.

Argument	Description
xs, ys	x, y coordinates of vertices
zs	z value(s), either one for all points or one for each point.
zdir	Which direction to use as z ('x', 'y' or 'z') when plotting a 2D set.

Other arguments are passed on to plot()
plot3D(xs, ys, *args, **kwargs)
Plot 2D or 3D data.

Argument	Description
xs, ys	x, y coordinates of vertices
zs	z value(s), either one for all points or one for each point.
zdir	Which direction to use as z ('x', 'y' or 'z') when plotting a 2D set.

Other arguments are passed on to plot()
plot_surface(X, Y, Z, *args, **kwargs)
Create a surface plot.

By default it will be colored in shades of a solid color, but it also supports color mapping by supplying the cmap argument.

Note: The rcount and ccount kwargs, which both default to 50, determine the maximum number of samples used in each direction. If the input data is larger, it will be downsampled (by slicing) to these numbers of points.

Parameters

X, Y, Z [2d arrays] Data values.
rcount, ccount [int] Maximum number of samples used in each direction.
If the input data is larger, it will be downsampled (by slicing) to these numbers of points. Defaults to 50.

New in version 2.0.

`rstride`, `cstride` [int] Downsampling stride in each direction. These arguments are mutually exclusive with `rcount` and `ccount`. If only one of `rstride` or `cstride` is set, the other defaults to 10.

'classic' mode uses a default of `rstride = cstride = 10` instead of the new default of `rcount = ccount = 50`.

`color` [color-like] Color of the surface patches.

`cmap` [Colormap] Colormap of the surface patches.

`facecolors` [array-like of colors.] Colors of each individual patch.

`norm` [Normalize] Normalization for the colormap.

`vmin`, `vmax` [float] Bounds for the normalization.

`shade` [bool] Whether to shade the face colors.

****kwargs** : Other arguments are forwarded to `Poly3DCollection`.

`plot_trisurf(*args, **kwargs)`

Argument	Description
X, Y, Z	Data values as 1D arrays
color	Color of the surface patches
cmap	A colormap for the surface patches.
norm	An instance of <code>Normalize</code> to map values to colors
vmin	Minimum value to map
vmax	Maximum value to map
shade	Whether to shade the facecolors

The (optional) triangulation can be specified in one of two ways; either:

```
plot_trisurf(triangulation, ...)
```

where `triangulation` is a `Triangulation` object, or:

```
plot_trisurf(X, Y, ...)
plot_trisurf(X, Y, triangles, ...)
plot_trisurf(X, Y, triangles=triangles, ...)
```

in which case a `Triangulation` object will be created. See `Triangulation` for a explanation of these possibilities.

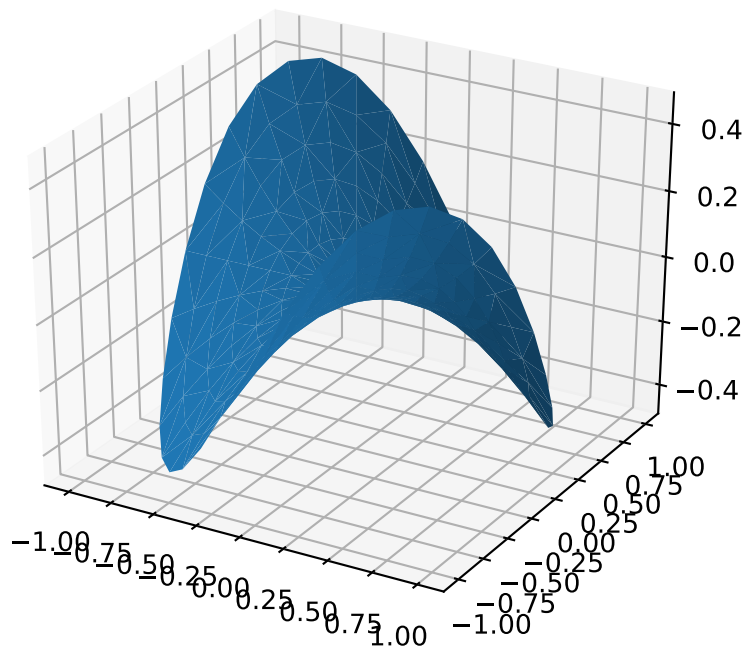
The remaining arguments are:

```
plot_trisurf(..., Z)
```

where Z is the array of values to contour, one per point in the triangulation.

Other arguments are passed on to `Poly3DCollection`

Examples:



New in version 1.2.0: This plotting function was added for the v1.2.0 release.

```
plot_wireframe(X, Y, Z, *args, **kwargs)
```

Plot a 3D wireframe.

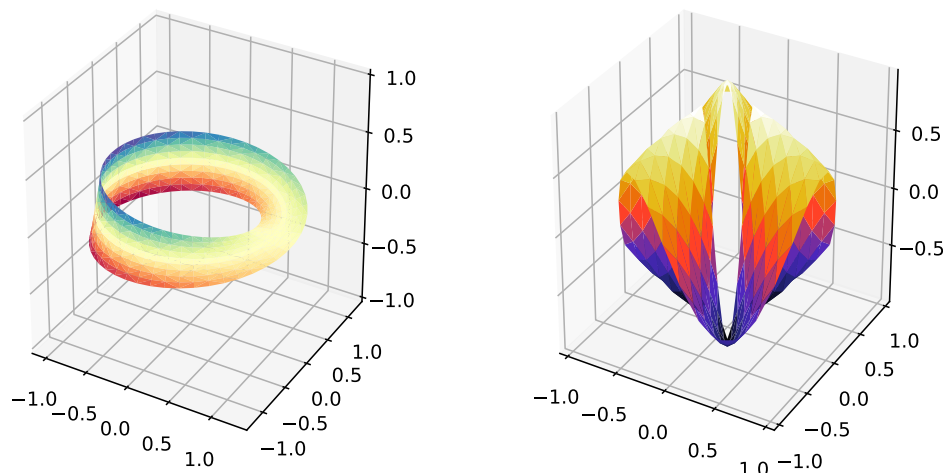
Note: The `rcount` and `ccount` kwargs, which both default to 50, determine the maximum number of samples used in each direction. If the input data is larger, it will be downsampled (by slicing) to these numbers of points.

Parameters

`X, Y, Z` [2d arrays] Data values.

`rcount, ccount` [int] Maximum number of samples used in each direction.

If the input data is larger, it will be downsampled (by slicing) to these numbers of points. Setting a count to zero causes the data to be not sampled in the corresponding direction, producing a 3D line plot rather than a wireframe plot. Defaults to 50.



New in version 2.0.

`rstride`, `cstride` [int] Downsampling stride in each direction. These arguments are mutually exclusive with `rcount` and `ccount`. If only one of `rstride` or `cstride` is set, the other defaults to 1. Setting a stride to zero causes the data to be not sampled in the corresponding direction, producing a 3D line plot rather than a wireframe plot.

'classic' mode uses a default of `rstride = cstride = 1` instead of the new default of `rcount = ccount = 50`.

****kwargs** : Other arguments are forwarded to `Line3DCollection`.

`quiver(*args, **kwargs)`

Plot a 3D field of arrows.

call signatures:

`quiver(X, Y, Z, U, V, W, **kwargs)`

Arguments:

X, Y, Z: The x, y and z coordinates of the arrow locations (default is tail of arrow; see `pivot` kwarg)

U, V, W: The x, y and z components of the arrow vectors

The arguments could be array-like or scalars, so long as they can be broadcast together. The arguments can also be masked arrays. If an element in any of argument is masked, then that corresponding quiver element will not be plotted.

Keyword arguments:

length: [1.0 | float] The length of each quiver, default to 1.0, the unit is the same with the axes

arrow_length_ratio: [0.3 | float] The ratio of the arrow head with respect to the quiver, default to 0.3

pivot: ['tail' | 'middle' | 'tip'] The part of the arrow that is at the grid point; the arrow rotates about this point, hence the name pivot. Default is 'tail'

normalize: bool When True, all of the arrows will be the same length. This defaults to False, where the arrows will be different lengths depending on the values of u,v,w.

Any additional keyword arguments are delegated to LineCollection

quiver3D(*args, **kwargs)
Plot a 3D field of arrows.

call signatures:

`quiver(X, Y, Z, U, V, W, **kwargs)`

Arguments:

X, Y, Z: The x, y and z coordinates of the arrow locations (default is tail of arrow; see pivot kwarg)

U, V, W: The x, y and z components of the arrow vectors

The arguments could be array-like or scalars, so long as they can be broadcast together. The arguments can also be masked arrays. If an element in any of argument is masked, then that corresponding quiver element will not be plotted.

Keyword arguments:

length: [1.0 | float] The length of each quiver, default to 1.0, the unit is the same with the axes

arrow_length_ratio: [0.3 | float] The ratio of the arrow head with respect to the quiver, default to 0.3

pivot: ['tail' | 'middle' | 'tip'] The part of the arrow that is at the grid point; the arrow rotates about this point, hence the name pivot. Default is 'tail'

normalize: bool When True, all of the arrows will be the same length. This defaults to False, where the arrows will be different lengths depending on the values of u,v,w.

Any additional keyword arguments are delegated to LineCollection

scatter(xs, ys, zs=0, zdir='z', s=20, c=None, depthshade=True, *args, **kwargs)
Create a scatter plot.

Argument	Description
xs, ys	Positions of data points.
zs	Either an array of the same length as xs and ys or a single value to place all points in the same plane. Default is 0.
zdir	Which direction to use as z ('x', 'y' or 'z') when plotting a 2D set.
s	Size in points ² . It is a scalar or an array of the same length as x and y.
c	A color. c can be a single color format string, or a sequence of color specifications of length N, or a sequence of N numbers to be mapped to colors using the cmap and norm specified via kwargs (see below). Note that c should not be a single numeric RGB or RGBA sequence because that is indistinguishable from an array of values to be colormapped. c can be a 2-D array in which the rows are RGB or RGBA, however, including the case of a single row to specify the same color for all points.
depthshade	Whether or not to shade the scatter markers to give the appearance of depth. Default is True.

Keyword arguments are passed on to `scatter()`.

Returns a `Patch3DCollection`

`scatter3D(xs, ys, zs=0, zdir='z', s=20, c=None, depthshade=True, *args, **kwargs)`
 Create a scatter plot.

Argument	Description
xs, ys	Positions of data points.
zs	Either an array of the same length as xs and ys or a single value to place all points in the same plane. Default is 0.
zdir	Which direction to use as z ('x', 'y' or 'z') when plotting a 2D set.
s	Size in points ² . It is a scalar or an array of the same length as x and y.
c	A color. c can be a single color format string, or a sequence of color specifications of length N, or a sequence of N numbers to be mapped to colors using the cmap and norm specified via kwargs (see below). Note that c should not be a single numeric RGB or RGBA sequence because that is indistinguishable from an array of values to be colormapped. c can be a 2-D array in which the rows are RGB or RGBA, however, including the case of a single row to specify the same color for all points.
depthshade	Whether or not to shade the scatter markers to give the appearance of depth. Default is True.

Keyword arguments are passed on to `scatter()`.

Returns a `Patch3DCollection`

`set_autoscale_on(b)`

Set whether autoscaling is applied on plot commands

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

Parameters

b [bool]

`set_autoscalez_on(b)`

Set whether autoscaling for the z-axis is applied on plot commands

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

Parameters

b [bool]

`set_axis_off()`

Turn off the axis.

`set_axis_on()`

Turn on the axis.

`set_axisbelow(b)`

Set whether axis ticks and gridlines are above or below most artists.

For axes3d objects, this will ignore any settings and just use True

New in version 1.1.0: This function was added for completeness.

Parameters

b [bool]

`set_frame_on(b)`

Set whether the 3D axes panels are drawn.

New in version 1.1.0.

Parameters

b [bool]

`set_proj_type(proj_type)`

Set the projection type.

Parameters

proj_type [str] Type of projection, accepts 'persp' and 'ortho'.

`set_title(label, fontdict=None, loc='center', **kwargs)`

Set a title for the axes.

Set one of the three available axes titles. The available titles are positioned above the axes in the center, flush with the left edge, and flush with the right edge.

Parameters

label [str] Text to use for the title

`fontdict` [dict] A dictionary controlling the appearance of the title text, the default `fontdict` is:

```
{'fontsize': rcParams['axes.titlesize'],
 'fontweight' : rcParams['axes.titleweight'],
 'verticalalignment': 'baseline',
 'horizontalalignment': 'center'}
```

`loc` [{`'center'`, `'left'`, `'right'`}, str, optional] Which title to set, defaults to `'center'`

`pad` [float] The offset of the title from the top of the axes, in points. Default is None to use `rcParams['axes.titlepad']`.

Returns

`text` [Text] The matplotlib text instance representing the title

Other Parameters

`**kwargs` [Text properties] Other keyword arguments are text properties, see Text for a list of valid text properties.

`set_top_view()`

`set_xlim(left=None, right=None, emit=True, auto=False, **kw)`

Set 3D x limits.

See `matplotlib.axes.Axes.set_xlim()` for full documentation.

`set_xlim3d(left=None, right=None, emit=True, auto=False, **kw)`

Set 3D x limits.

See `matplotlib.axes.Axes.set_xlim()` for full documentation.

`set_xscale(value, **kwargs)`

Set the x-axis scale.

Parameters

`value` [{`"linear"`, `"log"`, `"symlog"`, `"logit"`}] scaling strategy to apply

See also:

`matplotlib.scale.LinearScale` linear transform

`matplotlib.scale.LogTransform` log transform

`matplotlib.scale.SymmetricalLogTransform` symlog transform

`matplotlib.scale.LogisticTransform` logit transform .. versionadded :: 1.1.0 This function was added, but not tested. Please report any bugs.

Notes

Different kwargs are accepted, depending on the scale. See the scale module for more information.

`set_ylim(bottom=None, top=None, emit=True, auto=False, **kw)`

Set 3D y limits.

See `matplotlib.axes.Axes.set_ylim()` for full documentation.

`set_ylim3d(bottom=None, top=None, emit=True, auto=False, **kw)`

Set 3D y limits.

See `matplotlib.axes.Axes.set_ylim()` for full documentation.

`set_yscale(value, **kwargs)`

Set the y-axis scale.

Parameters

value [{`"linear"`, `"log"`, `"symlog"`, `"logit"`}] scaling strategy to apply

See also:

`matplotlib.scale.LinearScale` linear transform

`matplotlib.scale.LogTransform` log transform

`matplotlib.scale.SymmetricalLogTransform` symlog transform

`matplotlib.scale.LogisticTransform` logit transform .. versionadded :: 1.1.0 This function was added, but not tested. Please report any bugs.

Notes

Different kwargs are accepted, depending on the scale. See the scale module for more information.

`set_zbound(lower=None, upper=None)`

Set the lower and upper numerical bounds of the z-axis. This method will honor axes inversion regardless of parameter order. It will not change the `_autoscaleZon` attribute.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

`set_zlabel(zlabel, fontdict=None, labelpad=None, **kwargs)`

Set zlabel. See doc for `set_ylabel()` for description.

`set_zlim(bottom=None, top=None, emit=True, auto=False, **kw)`

Set 3D z limits.

See `matplotlib.axes.Axes.set_ylim()` for full documentation

`set_zlim3d(bottom=None, top=None, emit=True, auto=False, **kw)`

Set 3D z limits.

See `matplotlib.axes.Axes.set_ylim()` for full documentation

`set_zmargin(m)`

Set padding of Z data limits prior to autoscaling.

`m` times the data interval will be added to each end of that interval before it is used in autoscaling.

accepts: float in range 0 to 1

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

`set_zscale(value, **kwargs)`

Set the scaling of the z-axis: 'linear' | 'log' | 'logit' | 'symlog'

ACCEPTS: ['linear' | 'log' | 'logit' | 'symlog']

Different kwargs are accepted, depending on the scale: 'linear'

'log'

basex/basey: The base of the logarithm

nonposx/nonposy: ['mask' | 'clip'] non-positive values in x or y can be masked as invalid, or clipped to a very small positive number

subsx/subsy: Where to place the subticks between each major tick. Should be a sequence of integers. For example, in a log10 scale: [2, 3, 4, 5, 6, 7, 8, 9]

will place 8 logarithmically spaced minor ticks between each major tick.

'logit'

nonpos: ['mask' | 'clip'] values beyond]0, 1[can be masked as invalid, or clipped to a number very close to 0 or 1

'symlog'

basex/basey: The base of the logarithm

linthreshx/linthreshy: A single float which defines the range (-x, x), within which the plot is linear. This avoids having the plot go to infinity around zero.

subsx/subsy: Where to place the subticks between each major tick. Should be a sequence of integers. For example, in a log10 scale: [2, 3, 4, 5, 6, 7, 8, 9]

will place 8 logarithmically spaced minor ticks between each major tick.

linscalex/linscaley: This allows the linear range (-linthresh to linthresh) to be stretched relative to the logarithmic range. Its value is the number of decades to use for each half of the linear range. For example, when `linscale == 1.0` (the default), the space used for the positive and negative halves of the linear range will be equal to one decade in the logarithmic range.

Note: Currently, Axes3D objects only supports linear scales. Other scales may or may not work, and support for these is improving with each release.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

`set_zticklabels(*args, **kwargs)`

Set z-axis tick labels. See `matplotlib.axes.Axes.set_yticklabels()` for more details.

Note: Minor ticks are not supported by Axes3D objects.

New in version 1.1.0.

`set_zticks(*args, **kwargs)`

Set z-axis tick locations. See `matplotlib.axes.Axes.set_yticks()` for more details.

Note: Minor ticks are not supported.

New in version 1.1.0.

`text(x, y, z, s, zdir=None, **kwargs)`

Add text to the plot. `kwargs` will be passed on to `Axes.text`, except for the `zdir` keyword, which sets the direction to be used as the `z` direction.

`text2D(x, y, s, fontdict=None, withdash=False, **kwargs)`

Add text to the axes.

Add the text `s` to the axes at location `x, y` in data coordinates.

Parameters

`x, y` [scalars] The position to place the text. By default, this is in data coordinates. The coordinate system can be changed using the transform parameter.

`s` [str] The text.

`fontdict` [dictionary, optional, default: None] A dictionary to override the default text properties. If `fontdict` is None, the defaults are determined by your rc parameters.

`withdash` [boolean, optional, default: False] Creates a `TextWithDash` instance instead of a `Text` instance.

Returns

`text` [Text] The created Text instance.

Other Parameters

`**kwargs` [Text properties.] Other miscellaneous text parameters.

Examples

Individual keyword arguments can be used to override any given parameter:

```
>>> text(x, y, s, fontsize=12)
```

The default transform specifies that text is in data coords, alternatively, you can specify text in axis coords (0,0 is lower-left and 1,1 is upper-right). The example below places text in the center of the axes:

```
>>> text(0.5, 0.5, 'matplotlib', horizontalalignment='center',
...      verticalalignment='center', transform=ax.transAxes)
```

You can put a rectangular box around the text instance (e.g., to set a background color) by using the keyword `bbox`. `bbox` is a dictionary of Rectangle properties. For example:

```
>>> text(x, y, s, bbox=dict(facecolor='red', alpha=0.5))
```

`text3D(x, y, z, s, zdir=None, **kwargs)`

Add text to the plot. `kwargs` will be passed on to `Axes.text`, except for the `zdir` keyword, which sets the direction to be used as the `z` direction.

`tick_params(axis='both', **kwargs)`

Convenience method for changing the appearance of ticks and tick labels.

See `matplotlib.axes.Axes.tick_params()` for more complete documentation.

The only difference is that setting `axis` to `'both'` will mean that the settings are applied to all three axes. Also, the `axis` parameter also accepts a value of `'z'`, which would mean to apply to only the `z`-axis.

Also, because of how `Axes3D` objects are drawn very differently from regular 2D axes, some of these settings may have ambiguous meaning. For simplicity, the `'z'` axis will accept settings as if it was like the `'y'` axis.

Note: While this function is currently implemented, the core part of the `Axes3D` object may ignore some of these settings. Future releases will fix this. Priority will be given to those who file bugs.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

`ticklabel_format(**kwargs)`

Convenience method for manipulating the `ScalarFormatter` used by default for linear axes in `Axes3D` objects.

See `matplotlib.axes.Axes.ticklabel_format()` for full documentation. Note that this version applies to all three axes of the `Axes3D` object. Therefore, the `axis` argument will also accept a value of `'z'` and the value of `'both'` will apply to all three axes.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

`tricontour(*args, **kwargs)`
Create a 3D contour plot.

Argument	Description
X, Y, Z	Data values as <code>numpy.array</code> s
<code>extend3d</code>	Whether to extend contour in 3D (default: <code>False</code>)
<code>stride</code>	Stride (step size) for extending contour
<code>zdir</code>	The direction to use: <code>x</code> , <code>y</code> or <code>z</code> (default)
<code>offset</code>	If specified plot a projection of the contour lines on this position in plane normal to <code>zdir</code>

Other keyword arguments are passed on to `tricontour()`

Returns a contour

Changed in version 1.3.0: Added support for custom triangulations

EXPERIMENTAL: This method currently produces incorrect output due to a long-standing bug in 3D `PolyCollection` rendering.

`tricontourf(*args, **kwargs)`
Create a 3D contourf plot.

Argument	Description
X, Y, Z	Data values as <code>numpy.array</code> s
<code>zdir</code>	The direction to use: <code>x</code> , <code>y</code> or <code>z</code> (default)
<code>offset</code>	If specified plot a projection of the contour lines on this position in plane normal to <code>zdir</code>

Other keyword arguments are passed on to `tricontour()`

Returns a contour

Changed in version 1.3.0: Added support for custom triangulations

EXPERIMENTAL: This method currently produces incorrect output due to a long-standing bug in 3D `PolyCollection` rendering.

`tunit_cube(vals=None, M=None)`

`tunit_edges(vals=None, M=None)`

`unit_cube(vals=None)`

`update_datalim(xys, **kwargs)`

Update the data lim bbox with seq of xy tups or equiv. 2-D array

`view_init(elev=None, azimuth=None)`

Set the elevation and azimuth of the axes.

This can be used to rotate the axes programmatically.

'elev' stores the elevation angle in the z plane. 'azim' stores the azimuth angle in the x,y plane.

if elev or azim are None (default), then the initial value is used which was specified in the Axes3D constructor.

`voxels([x, y, z], /, filled, **kwargs)`

Plot a set of filled voxels

All voxels are plotted as 1x1x1 cubes on the axis, with filled[0,0,0] placed with its lower corner at the origin. Occluded faces are not plotted.

Call signatures:

```
voxels(filled, facecolors=fc, edgecolors=ec, **kwargs)
voxels(x, y, z, filled, facecolors=fc, edgecolors=ec, **kwargs)
```

New in version 2.1.

Parameters

filled [3D np.array of bool] A 3d array of values, with truthy values indicating which voxels to fill

x, y, z [3D np.array, optional] The coordinates of the corners of the voxels. This should broadcast to a shape one larger in every dimension than the shape of filled. These can be used to plot non-cubic voxels.

If not specified, defaults to increasing integers along each axis, like those returned by `indices()`. As indicated by the / in the function signature, these arguments can only be passed positionally.

facecolors, edgecolors [array_like, optional] The color to draw the faces and edges of the voxels. Can only be passed as keyword arguments. This parameter can be:

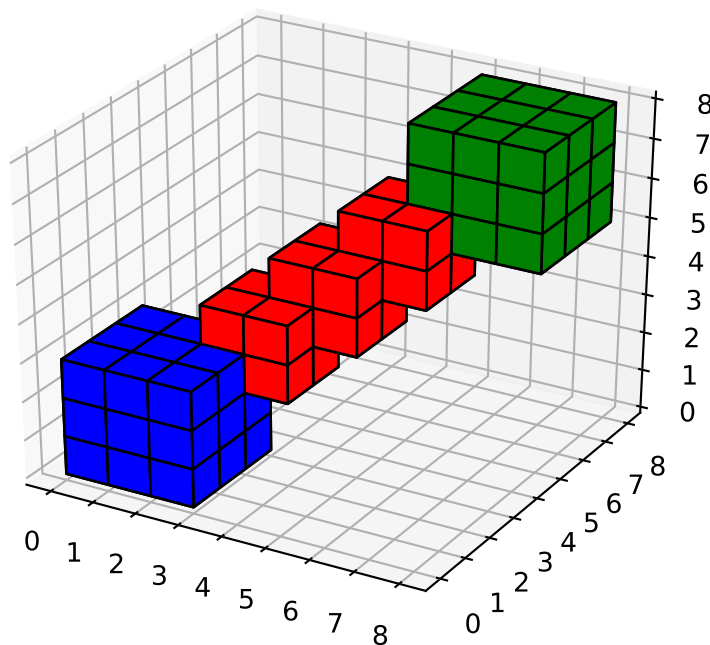
- A single color value, to color all voxels the same color. This can be either a string, or a 1D rgb/rgba array
- None, the default, to use a single color for the faces, and the style default for the edges.
- A 3D ndarray of color names, with each item the color for the corresponding voxel. The size must match the voxels.
- A 4D ndarray of rgb/rgba data, with the components along the last axis.

****kwargs** Additional keyword arguments to pass onto `Poly3DCollection()`

Returns

faces [dict] A dictionary indexed by coordinate, where `faces[i,j,k]` is a `Poly3DCollection` of the faces drawn for the voxel filled `[i,j,k]`. If no faces were drawn for a given voxel, either because it was not asked to be drawn, or it is fully occluded, then `(i,j,k)` not in `faces`.

Examples



`zaxis__date(tz=None)`

Sets up z-axis ticks and labels that treat the z data as dates.

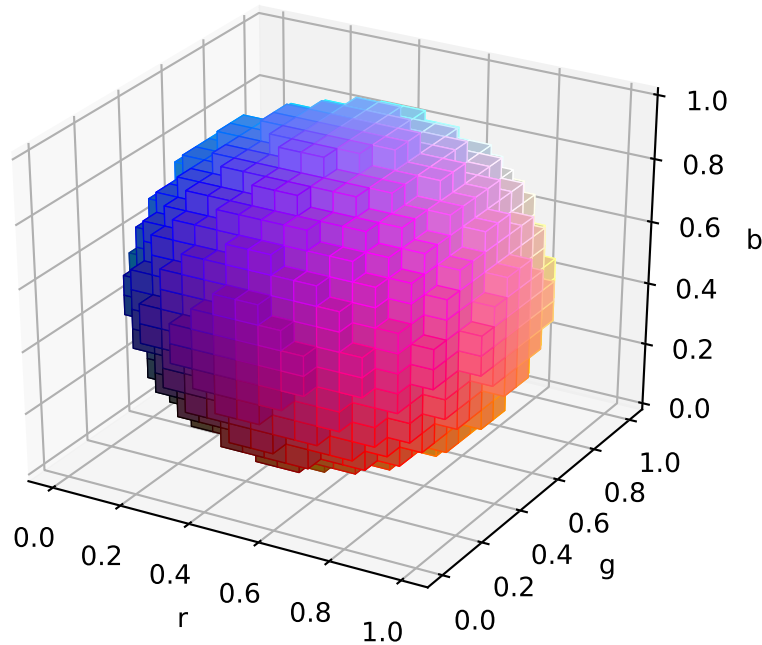
`tz` is a timezone string or `tzinfo` instance. Defaults to `rc` value.

Note: This function is merely provided for completeness. `Axes3D` objects do not officially support dates for ticks, and so this may or may not work as expected.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

`zaxis__inverted()`

Returns `True` if the z-axis is inverted.



New in version 1.1.0: This function was added, but not tested. Please report any bugs.

76.1.2 axis3d

Note: See `mpl_toolkits.mplot3d.axis3d._axinfo` for a dictionary containing constants that may be modified for controlling the look and feel of mplot3d axes (e.g., label spacing, font colors and panel colors). Historically, axis3d has suffered from having hard-coded constants that precluded user adjustments, and this dictionary was implemented in version 1.1 as a stop-gap measure.

```
axis3d.Axis(adir, v_intervalx, d_intervalx, ...)
```

```
mpl_toolkits.mplot3d.axis3d.Axis
```

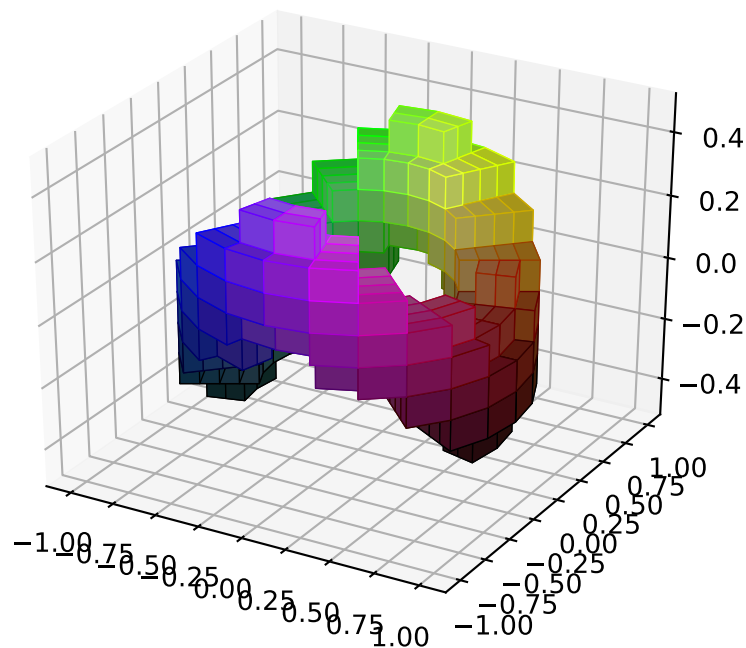
```
class mpl_toolkits.mplot3d.axis3d.Axis(adir, v_intervalx, d_intervalx, axes, *args,
                                         **kwargs)
```

```
    draw(renderer)
```

```
        Draw the axis lines, grid lines, tick lines and labels
```

```
    draw_pane(renderer)
```

```
76.1. mplot3d API
    get_major_ticks(numticks=None)
        get the tick instances; grow as necessary
    get_rotate_label(text)
```

rotated if longer than 4 chars.

`set_view_interval(vmin, vmax, ignore=False)`

If `ignore` is `False`, the order of `vmin`, `vmax` does not matter; the original axis orientation will be preserved. In addition, the view limits can be expanded, but will not be reduced. This method is for `mpl` internal use; for normal use, see `set_xlim()`.

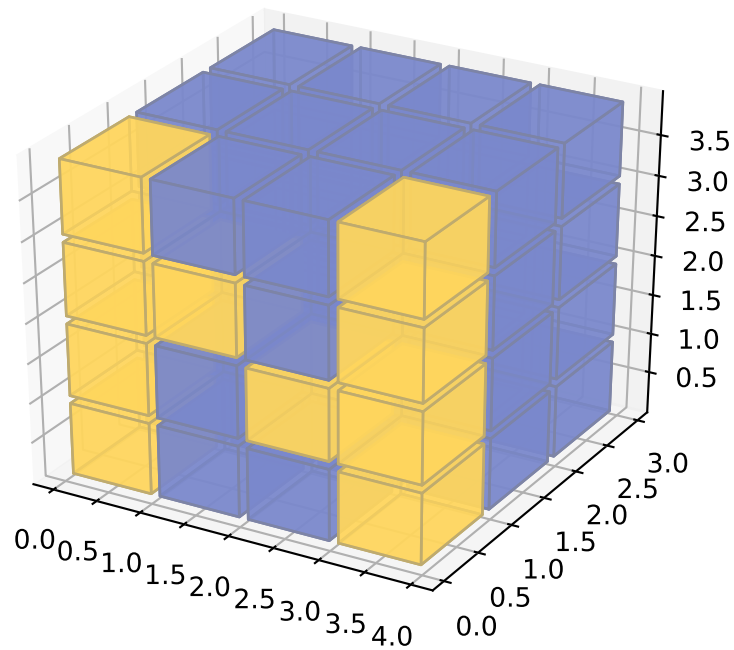
76.1.3 art3d

<code>art3d.Line3D(xs, ys, zs, *args, **kwargs)</code>	3D line object.
<code>art3d.Line3DCollection(segments, *args, **kwargs)</code>	A collection of 3D lines.
<code>art3d.Patch3D(*args, **kwargs)</code>	3D patch object.
<code>art3d.Patch3DCollection(*args, **kwargs)</code>	A collection of 3D patches.
<code>art3d.Path3DCollection(*args, **kwargs)</code>	A collection of 3D paths.
<code>art3d.PathPatch3D(path, **kwargs)</code>	3D PathPatch object.
<code>art3d.Poly3DCollection(verts, *args, **kwargs)</code>	A collection of 3D polygons.
<code>art3d.Text3D([x, y, z, text, zdir])</code>	Text object with 3D position and (in the future) direction.

`mpl_toolkits.mplot3d.art3d.Line3D`

`class mpl_toolkits.mplot3d.art3d.Line3D(xs, ys, zs, *args, **kwargs)`
3D line object.

Keyword arguments are passed onto `Line2D()`.



`set_sort_zpos(val)`
Set the position to use for z-sorting.

`mpl_toolkits.mplot3d.art3d.Patch3D`

`class mpl_toolkits.mplot3d.art3d.Patch3D(*args, **kwargs)`
3D patch object.

`do_3d_projection(renderer)`

`draw(renderer)`
Draw the Patch to the given renderer.

`get_facecolor()`
Return the face color of the Patch.

`get_path()`
Return the path of this patch

`set_3d_properties(verts, zs=0, zdir='z')`

`mpl_toolkits.mplot3d.art3d.Patch3DCollection`

```
class mpl_toolkits.mplot3d.art3d.Patch3DCollection(*args, **kwargs)
```

A collection of 3D patches.

Create a collection of flat 3D patches with its normal vector pointed in `zdir` direction, and located at `zs` on the `zdir` axis. `'zs'` can be a scalar or an array-like of the same length as the number of patches in the collection.

Constructor arguments are the same as for `PatchCollection`. In addition, keywords `zs=0` and `zdir='z'` are available.

Also, the keyword argument `"depthshade"` is available to indicate whether or not to shade the patches in order to give the appearance of depth (default is `True`). This is typically desired in scatter plots.

```
do_3d_projection(renderer)
```

```
set_3d_properties(zs, zdir)
```

```
set_sort_zpos(val)
```

Set the position to use for z-sorting.

`mpl_toolkits.mplot3d.art3d.Path3DCollection`

```
class mpl_toolkits.mplot3d.art3d.Path3DCollection(*args, **kwargs)
```

A collection of 3D paths.

Create a collection of flat 3D paths with its normal vector pointed in `zdir` direction, and located at `zs` on the `zdir` axis. `'zs'` can be a scalar or an array-like of the same length as the number of paths in the collection.

Constructor arguments are the same as for `PathCollection`. In addition, keywords `zs=0` and `zdir='z'` are available.

Also, the keyword argument `"depthshade"` is available to indicate whether or not to shade the patches in order to give the appearance of depth (default is `True`). This is typically desired in scatter plots.

```
do_3d_projection(renderer)
```

```
set_3d_properties(zs, zdir)
```

```
set_sort_zpos(val)
```

Set the position to use for z-sorting.

`mpl_toolkits.mplot3d.art3d.PathPatch3D`

`class mpl_toolkits.mplot3d.art3d.PathPatch3D(path, **kwargs)`

3D PathPatch object.

`do_3d_projection(renderer)`

`set_3d_properties(path, zs=0, zdir='z')`

`mpl_toolkits.mplot3d.art3d.Poly3DCollection`

`class mpl_toolkits.mplot3d.art3d.Poly3DCollection(verts, *args, **kwargs)`

A collection of 3D polygons.

Create a Poly3DCollection.

verts should contain 3D coordinates.

Keyword arguments: zsort, see `set_zsort` for options.

Note that this class does a bit of magic with the `__facecolors` and `__edgecolors` properties.

`do_3d_projection(renderer)`

Perform the 3D projection for this object.

`draw(renderer)`

Derived classes drawing method

`get_edgecolor()`

`get_edgecolors()`

`get_facecolor()`

`get_facecolors()`

`get_vector(segments3d)`

Optimize points for projection

`set_3d_properties()`

`set_alpha(alpha)`

Set the alpha transparencies of the collection. alpha must be a float or None.

ACCEPTS: float or None

`set_edgecolor(colors)`

Set the edgecolor(s) of the collection. c can be a matplotlib color spec (all patches have

same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'face', the edge color will always be the same as the face color. If it is 'none', the patch boundary will not be drawn.

ACCEPTS: matplotlib color spec or sequence of specs

`set_edgecolors(colors)`

Set the `edgecolor(s)` of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'face', the edge color will always be the same as the face color. If it is 'none', the patch boundary will not be drawn.

ACCEPTS: matplotlib color spec or sequence of specs

`set_facecolor(colors)`

Set the `facecolor(s)` of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'none', the patch will not be filled.

ACCEPTS: matplotlib color spec or sequence of specs

`set_facecolors(colors)`

Set the `facecolor(s)` of the collection. `c` can be a matplotlib color spec (all patches have same color), or a sequence of specs; if it is a sequence the patches will cycle through the sequence.

If `c` is 'none', the patch will not be filled.

ACCEPTS: matplotlib color spec or sequence of specs

`set_sort_zpos(val)`

Set the position to use for z-sorting.

`set_verts(vertices, closed=True)`

Set 3D vertices.

`set_verts_and_codes(vertices, codes)`

Sets 3D vertices with path codes

`set_zsort(zsort)`

Set z-sorting behaviour: boolean: if True use default 'average' string: 'average', 'min' or 'max'

`mpl_toolkits.mplot3d.art3d.Text3D`

`class mpl_toolkits.mplot3d.art3d.Text3D(x=0, y=0, z=0, text="", zdir='z', **kwargs)`

Text object with 3D position and (in the future) direction.

x, y, z Position of text text Text string to display zdir Direction of text

Keyword arguments are passed onto Text().

draw(renderer)

Draws the Text object to the given renderer.

set_3d_properties(z=0, zdir='z')

76.1.4 Art3D Utility Functions

art3d.get_colors(c, num)	Stretch the color argument to provide the required number num
art3d.get_dir_vector(zdir)	
art3d.get_patch_verts(patch)	Return a list of vertices for the path of a patch.
art3d.juggle_axes(xs, ys, zs, zdir)	Reorder coordinates so that 2D xs, ys can be plotted in the plane orthogonal to zdir.
art3d.line_2d_to_3d(line[, zs, zdir])	Convert a 2D line to 3D.
art3d.line_collection_2d_to_3d(col[, zs, zdir])	Convert a LineCollection to a Line3DCollection object.
art3d.norm_angle(a)	Return angle between -180 and +180
art3d.norm_text_angle(a)	Return angle between -90 and +90
art3d.patch_2d_to_3d(patch[, z, zdir])	Convert a Patch to a Patch3D object.
art3d.patch_collection_2d_to_3d(col[, zs, ...])	Convert a PatchCollection into a Patch3DCollection object (or a PathCollection into a Path3DCollection object).
art3d.path_to_3d_segment(path[, zs, zdir])	Convert a path to a 3D segment.
art3d.path_to_3d_segment_with_codes(path[, ...])	Convert a path to a 3D segment with path codes.
art3d.pathpatch_2d_to_3d(pathpatch[, z, zdir])	Convert a PathPatch to a PathPatch3D object.
art3d.paths_to_3d_segments(paths[, zs, zdir])	Convert paths from a collection object to 3D segments.
art3d.paths_to_3d_segments_with_codes(paths)	Convert paths from a collection object to 3D segments with path codes.
art3d.poly_collection_2d_to_3d(col[, zs, zdir])	Convert a PolyCollection to a Poly3DCollection object.
art3d.rotate_axes(xs, ys, zs, zdir)	Reorder coordinates so that the axes are rotated with zdir along the original z axis.
art3d.text_2d_to_3d(obj[, z, zdir])	Convert a Text to a Text3D object.
art3d.zalpha(colors, zs)	Modify the alphas of the color list according to depth

`mpl_toolkits.mplot3d.art3d.get_colors`

`mpl_toolkits.mplot3d.art3d.get_colors(c, num)`
Stretch the color argument to provide the required number num

`mpl_toolkits.mplot3d.art3d.get_dir_vector`

`mpl_toolkits.mplot3d.art3d.get_dir_vector(zdir)`

`mpl_toolkits.mplot3d.art3d.get_patch_verts`

`mpl_toolkits.mplot3d.art3d.get_patch_verts(patch)`
Return a list of vertices for the path of a patch.

`mpl_toolkits.mplot3d.art3d.juggle_axes`

`mpl_toolkits.mplot3d.art3d.juggle_axes(xs, ys, zs, zdir)`
Reorder coordinates so that 2D xs, ys can be plotted in the plane orthogonal to zdir. zdir is normally x, y or z. However, if zdir starts with a '-' it is interpreted as a compensation for rotate_axes.

`mpl_toolkits.mplot3d.art3d.line_2d_to_3d`

`mpl_toolkits.mplot3d.art3d.line_2d_to_3d(line, zs=0, zdir='z')`
Convert a 2D line to 3D.

`mpl_toolkits.mplot3d.art3d.line_collection_2d_to_3d`

`mpl_toolkits.mplot3d.art3d.line_collection_2d_to_3d(col, zs=0, zdir='z')`
Convert a LineCollection to a Line3DCollection object.

`mpl_toolkits.mplot3d.art3d.norm_angle`

`mpl_toolkits.mplot3d.art3d.norm_angle(a)`
Return angle between -180 and +180

`mpl_toolkits.mplot3d.art3d.norm_text_angle`

`mpl_toolkits.mplot3d.art3d.norm_text_angle(a)`
Return angle between -90 and +90

`mpl_toolkits.mplot3d.art3d.patch_2d_to_3d`

`mpl_toolkits.mplot3d.art3d.patch_2d_to_3d(patch, z=0, zdir='z')`
Convert a Patch to a Patch3D object.

`mpl_toolkits.mplot3d.art3d.patch_collection_2d_to_3d`

`mpl_toolkits.mplot3d.art3d.patch_collection_2d_to_3d(col, zs=0, zdir='z', depthshade=True)`
Convert a PatchCollection into a Patch3DCollection object (or a PathCollection into a Path3DCollection object).

Keywords:

`za` The location or locations to place the patches in the collection along the `zdir` axis. Defaults to 0.

`zdir` The axis in which to place the patches. Default is "z".

`depthshade` Whether to shade the patches to give a sense of depth. Defaults to True.

`mpl_toolkits.mplot3d.art3d.path_to_3d_segment`

`mpl_toolkits.mplot3d.art3d.path_to_3d_segment(path, zs=0, zdir='z')`
Convert a path to a 3D segment.

`mpl_toolkits.mplot3d.art3d.path_to_3d_segment_with_codes`

`mpl_toolkits.mplot3d.art3d.path_to_3d_segment_with_codes(path, zs=0, zdir='z')`
Convert a path to a 3D segment with path codes.

`mpl_toolkits.mplot3d.art3d.pathpatch_2d_to_3d`

`mpl_toolkits.mplot3d.art3d.pathpatch_2d_to_3d(pathpatch, z=0, zdir='z')`
Convert a PathPatch to a PathPatch3D object.

Examples using `mpl_toolkits.mplot3d.art3d.pathpatch_2d_to_3d`

- `sphx_glr_gallery_mplot3d_pathpatch3d.py`

`mpl_toolkits.mplot3d.art3d.paths_to_3d_segments`

`mpl_toolkits.mplot3d.art3d.paths_to_3d_segments(paths, zs=0, zdir='z')`
Convert paths from a collection object to 3D segments.

`mpl_toolkits.mplot3d.art3d.paths_to_3d_segments_with_codes`

`mpl_toolkits.mplot3d.art3d.paths_to_3d_segments_with_codes(paths, zs=0, zdir='z')`
Convert paths from a collection object to 3D segments with path codes.

`mpl_toolkits.mplot3d.art3d.poly_collection_2d_to_3d`

`mpl_toolkits.mplot3d.art3d.poly_collection_2d_to_3d(col, zs=0, zdir='z')`
Convert a PolyCollection to a Poly3DCollection object.

`mpl_toolkits.mplot3d.art3d.rotate_axes`

`mpl_toolkits.mplot3d.art3d.rotate_axes(xs, ys, zs, zdir)`
Reorder coordinates so that the axes are rotated with zdir along the original z axis. Prepending the axis with a '-' does the inverse transform, so zdir can be x, -x, y, -y, z or -z

`mpl_toolkits.mplot3d.art3d.text_2d_to_3d`

`mpl_toolkits.mplot3d.art3d.text_2d_to_3d(obj, z=0, zdir='z')`
Convert a Text to a Text3D object.

`mpl_toolkits.mplot3d.art3d.zalpha`

`mpl_toolkits.mplot3d.art3d.zalpha(colors, zs)`
Modify the alphas of the color list according to depth

76.1.5 proj3d

<code>proj3d.inv_transform(xs, ys, zs, M)</code>	
<code>proj3d.line2d(p0, p1)</code>	Return 2D equation of line in the form $ax+by+c = 0$
<code>proj3d.line2d_dist(l, p)</code>	Distance from line to point line is a tuple of coefficients a,b,c
<code>proj3d.line2d_seg_dist(p1, p2, p0)</code>	distance(s) from line defined by p1 - p2 to point(s) p0
<code>proj3d.mod(v)</code>	3d vector length
<code>proj3d.persp_transformation(zfront, zback)</code>	
<code>proj3d.proj_points(points, M)</code>	
<code>proj3d.proj_trans_clip_points(points, M)</code>	
<code>proj3d.proj_trans_points(points, M)</code>	
<code>proj3d.proj_transform(xs, ys, zs, M)</code>	Transform the points by the projection matrix

Continued on next page

Table 5 – continued from previous page

<code>proj3d.proj_transform_clip(xs, ys, zs, M)</code>	Transform the points by the projection matrix and return the clipping result returns <code>txs,tys,tzs,tis</code>
<code>proj3d.proj_transform_vec(vec, M)</code>	
<code>proj3d.proj_transform_vec_clip(vec, M)</code>	
<code>proj3d.rot_x(V, alpha)</code>	
<code>proj3d.transform(xs, ys, zs, M)</code>	Transform the points by the projection matrix
<code>proj3d.vec_pad_ones(xs, ys, zs)</code>	
<code>proj3d.view_transformation(E, R, V)</code>	
<code>proj3d.world_transformation(xmin, xmax, ...)</code>	

`mpl_toolkits.mplot3d.proj3d.inv_transform`

`mpl_toolkits.mplot3d.proj3d.inv_transform(xs, ys, zs, M)`

`mpl_toolkits.mplot3d.proj3d.line2d`

`mpl_toolkits.mplot3d.proj3d.line2d(p0, p1)`
Return 2D equation of line in the form $ax+by+c = 0$

`mpl_toolkits.mplot3d.proj3d.line2d_dist`

`mpl_toolkits.mplot3d.proj3d.line2d_dist(l, p)`
Distance from line to point line is a tuple of coefficients a,b,c

`mpl_toolkits.mplot3d.proj3d.line2d_seg_dist`

`mpl_toolkits.mplot3d.proj3d.line2d_seg_dist(p1, p2, p0)`
distance(s) from line defined by $p1 - p2$ to point(s) $p0$
 $p0[0] = x(s)$ $p0[1] = y(s)$
intersection point $p = p1 + u*(p2-p1)$ and intersection point lies within segment if u is between 0 and 1

`mpl_toolkits.mplot3d.proj3d.mod`

`mpl_toolkits.mplot3d.proj3d.mod(v)`
3d vector length

`mpl_toolkits.mplot3d.proj3d.persp_transformation`

`mpl_toolkits.mplot3d.proj3d.persp_transformation(zfront, zback)`

`mpl_toolkits.mplot3d.proj3d.proj_points`

`mpl_toolkits.mplot3d.proj3d.proj_points(points, M)`

`mpl_toolkits.mplot3d.proj3d.proj_trans_clip_points`

`mpl_toolkits.mplot3d.proj3d.proj_trans_clip_points(points, M)`

`mpl_toolkits.mplot3d.proj3d.proj_trans_points`

`mpl_toolkits.mplot3d.proj3d.proj_trans_points(points, M)`

`mpl_toolkits.mplot3d.proj3d.proj_transform`

`mpl_toolkits.mplot3d.proj3d.proj_transform(xs, ys, zs, M)`

Transform the points by the projection matrix

`mpl_toolkits.mplot3d.proj3d.proj_transform_clip`

`mpl_toolkits.mplot3d.proj3d.proj_transform_clip(xs, ys, zs, M)`

Transform the points by the projection matrix and return the clipping result returns
`txs,tys,tzs,tis`

`mpl_toolkits.mplot3d.proj3d.proj_transform_vec`

`mpl_toolkits.mplot3d.proj3d.proj_transform_vec(vec, M)`

`mpl_toolkits.mplot3d.proj3d.proj_transform_vec_clip`

`mpl_toolkits.mplot3d.proj3d.proj_transform_vec_clip(vec, M)`

`mpl_toolkits.mplot3d.proj3d.rot_x`

`mpl_toolkits.mplot3d.proj3d.rot_x(V, alpha)`

`mpl_toolkits.mplot3d.proj3d.transform`

`mpl_toolkits.mplot3d.proj3d.transform(xs, ys, zs, M)`

Transform the points by the projection matrix

`mpl_toolkits.mplot3d.proj3d.vec_pad_ones`

`mpl_toolkits.mplot3d.proj3d.vec_pad_ones(xs, ys, zs)`

`mpl_toolkits.mplot3d.proj3d.view_transformation`

`mpl_toolkits.mplot3d.proj3d.view_transformation(E, R, V)`

`mpl_toolkits.mplot3d.proj3d.world_transformation`

`mpl_toolkits.mplot3d.proj3d.world_transformation(xmin, xmax, ymin, ymax, zmin, zmax)`

76.2 Matplotlib axes_grid Toolkit

Note: AxesGrid toolkit has been a part of matplotlib since v 0.99. Originally, the toolkit had a single namespace of `axes_grid`. In more recent version, the toolkit has divided into two separate namespace (`axes_grid1` and `axisartist`). While `axes_grid` namespace is maintained for the backward compatibility, use of `axes_grid1` and `axisartist` is recommended. For the documentation on `axes_grid`, see the [previous version of the docs](#).

Part VII

The Matplotlib Developers' Guide

CONTRIBUTING

This project is a community effort, and everyone is welcome to contribute.

The project is hosted on <https://github.com/matplotlib/matplotlib>

77.1 Submitting a bug report

If you find a bug in the code or documentation, do not hesitate to submit a ticket to the [Bug Tracker](#). You are also welcome to post feature requests or pull requests.

If you are reporting a bug, please do your best to include the following:

1. A short, top-level summary of the bug. In most cases, this should be 1-2 sentences.
2. A short, self-contained code snippet to reproduce the bug, ideally allowing a simple copy and paste to reproduce. Please do your best to reduce the code snippet to the minimum required.
3. The actual outcome of the code snippet.
4. The expected outcome of the code snippet.
5. The Matplotlib version, Python version and platform that you are using. You can grab the version with the following commands:

```
>>> import matplotlib
>>> matplotlib.__version__
'1.5.3'
>>> import platform
>>> platform.python_version()
'2.7.12'
```

We have preloaded the issue creation page with a Markdown template that you can use to organize this information.

Thank you for your help in keeping bug reports complete, targeted and descriptive.

77.2 Retrieving and installing the latest version of the code

When developing Matplotlib, sources must be downloaded, built, and installed into a local environment on your machine.

Follow the instructions detailed [here](#) to set up your environment to build Matplotlib from source.

Warning: When working on Matplotlib sources, having multiple versions installed by different methods into the same environment may not always work as expected.

To work on Matplotlib sources, it is strongly recommended to set up an alternative development environment, using something like [virtual environments in python](#), or a [conda environment](#).

If you choose to use an already existing environment, and not a clean virtual or conda environment, uninstall the current version of Matplotlib in that environment using the same method used to install it.

If working on Matplotlib documentation only, the above steps are not absolutely necessary.

We use [Git](#) for version control and [GitHub](#) for hosting our main repository.

You can check out the latest sources with the command (see Set up your fork for more details):

```
git clone https://github.com/matplotlib/matplotlib.git
```

and navigate to the matplotlib directory. If you have the proper privileges, you can use `git@` instead of `https://`, which works through the ssh protocol and might be easier to use if you are using 2-factor authentication.

77.2.1 Building Matplotlib for image comparison tests

Matplotlib's test suite makes heavy use of image comparison tests, meaning the result of a plot is compared against a known good result. Unfortunately, different versions of FreeType produce differently formed characters, causing these image comparisons to fail. To make them reproducible, Matplotlib can be built with a special local copy of FreeType. This is recommended for all Matplotlib developers.

Copy `setup.cfg.template` to `setup.cfg` and edit it to contain:

```
[test]
local_freetype = True
tests = True
```

or set the `MPLLOCALFREETYPE` environmental variable to any true value.

77.2.2 Installing Matplotlib in developer mode

To install Matplotlib (and compile the C-extensions) run the following command from the top-level directory

```
python -mpip install -ve .
```

This installs Matplotlib in 'editable/develop mode', i.e., builds everything and places the correct link entries in the install directory so that python will be able to import Matplotlib from the source directory. Thus, any changes to the *.py files will be reflected the next time you import the library. If you change the C-extension source (which might happen if you change branches) you will need to run

```
python setup.py build
```

or re-run `python -mpip install -ve ..`

Alternatively, if you do

```
python -mpip install -v .
```

all of the files will be copied to the installation directory however, you will have to rerun this command every time the source is changed. Additionally you will need to copy `setup.cfg.template` to `setup.cfg` and edit it to contain

```
[test]
local_freetype = True
tests = True
```

In either case you can then run the tests to check your work environment is set up properly:

```
python tests.py
```

Note: Additional dependencies for testing: `pytest` (version 3.6 or later), `mock` (if Python 2), `Ghostscript`, `Inkscape`

See also:

- Developer's tips for testing

77.3 Contributing code

77.3.1 How to contribute

The preferred way to contribute to Matplotlib is to fork the [main repository](#) on GitHub, then submit a "pull request" (PR).

The best practices for using GitHub to make PRs to Matplotlib are documented in the Development workflow section.

A brief overview is:

1. [Create an account](#) on GitHub if you do not already have one.
2. Fork the [project repository](#): click on the 'Fork' button near the top of the page. This creates a copy of the code under your account on the GitHub server.
3. Clone this copy to your local disk:

```
$ git clone https://github.com/YourLogin/matplotlib.git
```

4. Create a branch to hold your changes:

```
$ git checkout -b my-feature origin/master
```

and start making changes. Never work in the master branch!

5. Work on this copy, on your computer, using Git to do the version control. When you're done editing e.g., `lib/matplotlib/collections.py`, do:

```
$ git add lib/matplotlib/collections.py
$ git commit
```

to record your changes in Git, then push them to GitHub with:

```
$ git push -u origin my-feature
```

Finally, go to the web page of your fork of the Matplotlib repo, and click 'Pull request' to send your changes to the maintainers for review. You may want to consider sending an email to the mailing list for more visibility.

See also:

- [Git documentation](#)
- Development workflow.
- Working with Matplotlib source code

77.3.2 Contributing pull requests

It is recommended to check that your contribution complies with the following rules before submitting a pull request:

- If your pull request addresses an issue, please use the title to describe the issue and mention the issue number in the pull request description to ensure that a link is created to the original issue.
- All public methods should have informative docstrings with sample usage when appropriate. Use the [numpy docstring standard](#).

- Formatting should follow the recommendations of [PEP8](#). You should consider installing/enabling automatic PEP8 checking in your editor. Part of the test suite is checking PEP8 compliance, things go smoother if the code is mostly PEP8 compliant to begin with.
- Each high-level plotting function should have a simple example in the Example section of the docstring. This should be as simple as possible to demonstrate the method. More complex examples should go in the examples tree.
- Changes (both new features and bugfixes) should be tested. See Developer's tips for testing for more details.
- Import the following modules using the standard scipy conventions:

```
import numpy as np
import numpy.ma as ma
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.cbook as cbook
import matplotlib.patches as mpatches
```

- If your change is a major new feature, add an entry to the What's new section by adding a new file in doc/users/next_whats_new (see doc/users/next_whats_new/README.rst for more information).
- If you change the API in a backward-incompatible way, please document it in doc/api/api_changes, by adding a new file describing your changes (see doc/api/api_changes/README.rst for more information)
- See below for additional points about Keyword argument processing, if applicable for your pull request.

In addition, you can check for common programming errors with the following tools:

- Code with a good unittest coverage (at least 70%, better 100%), check with:

```
python -mpip install coverage
python tests.py --with-coverage
```

- No pyflakes warnings, check with:

```
python -mpip install pyflakes
pyflakes path/to/module.py
```

Note: The current state of the Matplotlib code base is not compliant with all of those guidelines, but we expect that enforcing those constraints on all new contributions will move the overall code base quality in the right direction.

See also:

- Coding guidelines

- Developer’s tips for testing
- Writing documentation

77.3.3 Issues for New Contributors

New contributors should look for the following tags when looking for issues. We strongly recommend that new contributors tackle issues labeled `good first issue` as they are easy, well documented issues, that do not require an understanding of the different submodules of Matplotlib. This helps the contributor become familiar with the contribution workflow, and for the core devs to become acquainted with the contributor; besides which, we frequently underestimate how easy an issue is to solve!

77.4 Other ways to contribute

Code is not the only way to contribute to Matplotlib. For instance, documentation is also a very important part of the project and often doesn’t get as much attention as it deserves. If you find a typo in the documentation, or have made improvements, do not hesitate to send an email to the mailing list or submit a GitHub pull request. Full documentation can be found under the `doc/` directory.

It also helps us if you spread the word: reference the project from your blog and articles or link to it from your website!

77.5 Coding guidelines

77.5.1 New modules and files: installation

- If you have added new files or directories, or reorganized existing ones, make sure the new files are included in the match patterns in `MANIFEST.in`, and/or in `package_data` in `setup.py`.

77.5.2 C/C++ extensions

- Extensions may be written in C or C++.
- Code style should conform to PEP7 (understanding that PEP7 doesn’t address C++, but most of its admonitions still apply).
- Python/C interface code should be kept separate from the core C/C++ code. The interface code should be named `FOO_wrap.cpp` or `FOO_wrapper.cpp`.
- Header file documentation (aka docstrings) should be in Numpydoc format. We don’t plan on using automated tools for these docstrings, and the Numpydoc format is well understood in the scientific Python community.

77.5.3 Keyword argument processing

Matplotlib makes extensive use of `**kwargs` for pass-through customizations from one function to another. A typical example is in `matplotlib.pyplot.text()`. The definition of the pylab text function is a simple pass-through to `matplotlib.axes.Axes.text()`:

```
# in pylab.py
def text(*args, **kwargs):
    ret = gca().text(*args, **kwargs)
    draw_if_interactive()
    return ret
```

`text()` in simplified form looks like this, i.e., it just passes all args and kwargs on to `matplotlib.text.Text.__init__()`:

```
# in axes/_axes.py
def text(self, x, y, s, fontdict=None, withdash=False, **kwargs):
    t = Text(x=x, y=y, text=s, **kwargs)
```

and `__init__()` (again with liberties for illustration) just passes them on to the `matplotlib.artist.Artist.update()` method:

```
# in text.py
def __init__(self, x=0, y=0, text='', **kwargs):
    Artist.__init__(self)
    self.update(kwargs)
```

`update` does the work looking for methods named like `set_property` if `property` is a keyword argument. i.e., no one looks at the keywords, they just get passed through the API to the artist constructor which looks for suitably named methods and calls them with the value.

As a general rule, the use of `**kwargs` should be reserved for pass-through keyword arguments, as in the example above. If all the keyword args are to be used in the function, and not passed on, use the key/value keyword args in the function definition rather than the `**kwargs` idiom.

In some cases, you may want to consume some keys in the local function, and let others pass through. You can pop the ones to be used locally and pass on the rest. For example, in `plot()`, `scalex` and `scaley` are local arguments and the rest are passed on as `Line2D()` keyword arguments:

```
# in axes/_axes.py
def plot(self, *args, **kwargs):
    scalex = kwargs.pop('scalex', True)
    scaley = kwargs.pop('scaley', True)
    if not self._hold: self.cla()
    lines = []
    for line in self._get_lines(*args, **kwargs):
        self.add_line(line)
        lines.append(line)
```

Note: there is a use case when kwargs are meant to be used locally in the function (not passed on), but you still need the `**kwargs` idiom. That is when you want to use `*args` to allow variable numbers of non-keyword args. In this case, python will not allow you to use named keyword args after the `*args` usage, so you will be forced to use `**kwargs`. An example is `matplotlib.contour.ContourLabeler.clabel()`:

```
# in contour.py
def clabel(self, *args, **kwargs):
    fontsize = kwargs.get('fontsize', None)
    inline = kwargs.get('inline', 1)
    self.fmt = kwargs.get('fmt', '%1.3f')
    colors = kwargs.get('colors', None)
    if len(args) == 0:
        levels = self.levels
        indices = range(len(self.levels))
    elif len(args) == 1:
        ...etc...
```

77.5.4 Using logging for debug messages

Matplotlib uses the standard python `logging` library to write verbose warnings, information, and debug messages. Please use it! In all those places you write `print()` statements to do your debugging, try using `log.debug()` instead!

To include `logging` in your module, at the top of the module, you need to import logging. Then calls in your code like:

```
_log = logging.getLogger(__name__) # right after the imports

# code
# more code
_log.info('Here is some information')
_log.debug('Here is some more detailed information')
```

will log to a logger named `matplotlib.yourmodulename`.

If an end-user of Matplotlib sets up `logging` to display at levels more verbose than `logger.WARNING` in their code as follows:

```
import logging
fmt = '%(name)s:%(lineno)5d - %(levelname)s - %(message)s'
logging.basicConfig(level=logging.DEBUG, format=fmt)
import matplotlib.pyplot as plt
```

Then they will receive messages like:

```
matplotlib.backends: 89 - INFO - backend MacOSX version unknown
matplotlib.yourmodulename: 347 - INFO - Here is some information
matplotlib.yourmodulename: 348 - DEBUG - Here is some more detailed information
```

Which logging level to use?

There are five levels at which you can emit messages. `logging.critical` and `logging.error` are really only there for errors that will end the use of the library but not kill the interpreter. `logging.warning` overlaps with the warnings library. The [logging tutorial](#) suggests that the difference between `logging.warning` and `warnings.warn` is that `warnings.warn` be used for things the user must change to stop the warning, whereas `logging.warning` can be more persistent.

By default, `logging` displays all log messages at levels higher than `logging.WARNING` to `sys.stderr`.

Calls to `logging.info` are not displayed by default. They are for information that the user may want to know if the program behaves oddly. For instance, if an object isn't drawn because its position is NaN, that can usually be ignored, but a mystified user could set `logging.basicConfig(level=logging.INFO)` and get an error message that says why.

`logging.debug` is the least likely to be displayed, and hence can be the most verbose.

77.5.5 Developing a new backend

If you are working on a custom backend, the backend setting in `matplotlibrc` ([Customizing Matplotlib with style sheets and rcParams](#)) supports an external backend via the module directive. If `my_backend.py` is a Matplotlib backend in your `PYTHONPATH`, you can set it on one of several ways

- in `matplotlibrc`:

```
backend : module://my_backend
```

- with the `MPLBACKEND` environment variable:

```
> export MPLBACKEND="module://my_backend"
> python simple_plot.py
```

- with the use directive in your script:

```
import matplotlib
matplotlib.use('module://my_backend')
```

77.5.6 Writing examples

We have hundreds of examples in subdirectories of `matplotlib/examples`, and these are automatically generated when the website is built to show up in the examples section of the website.

Any sample data that the example uses should be kept small and distributed with Matplotlib in the `lib/matplotlib/mpl-data/sample_data/` directory. Then in your example code you can load it into a file handle with:

```
import matplotlib.cbook as cbook
fh = cbook.get_sample_data('mydata.dat')
```

DEVELOPER'S TIPS FOR TESTING

Matplotlib's testing infrastructure depends on `pytest`. The tests are in `lib/matplotlib/tests`, and customizations to the `pytest` testing infrastructure are in `matplotlib.testing`.

78.1 Requirements

Install the latest version of Matplotlib as documented in Retrieving and installing the latest version of the code. In particular, follow the instructions to use a local FreeType build.

The following software is required to run the tests:

- `pytest` (≥ 3.6)
- `mock`, when running Python 2
- `Ghostscript` (to render PDF files)
- `Inkscape` (to render SVG files)

Optionally you can install:

- `pytest-cov` ($\geq 2.3.1$) to collect coverage information
- `pytest-pep8` to test coding standards
- `pytest-timeout` to limit runtime in case of stuck tests
- `pytest-xdist` to run tests in parallel

78.2 Running the tests

Running the tests is simple. Make sure you have `pytest` installed and run:

```
py.test
```

or:

```
python tests.py
```

in the root directory of the distribution. The script takes a set of commands, such as:

<code>--pep8</code>	Perform pep8 checks (requires pytest-pep8)
<code>-m "not network"</code>	Disable tests that require network access

Additional arguments are passed on to `pytest`. See the `pytest` documentation for [supported arguments](#). Some of the more important ones are given here:

<code>--verbose</code>	Be more verbose
<code>--n NUM</code>	Run tests in parallel over NUM processes (requires pytest-xdist)
<code>--timeout=SECONDS</code>	Set timeout for results from each test process (requires pytest-timeout)
<code>--capture=no</code> or <code>-s</code>	Do not capture stdout

To run a single test from the command line, you can provide a file path, optionally followed by the function separated by two colons, e.g., (tests do not need to be installed, but Matplotlib should be):

```
py.test lib/matplotlib/tests/test_simplification.py::test_clipping
```

or, if tests are installed, a dot-separated path to the module, optionally followed by the function separated by two colons, such as:

```
py.test --pyargs matplotlib.tests.test_simplification::test_clipping
```

If you want to run the full test suite, but want to save wall time try running the tests in parallel:

```
py.test --verbose -n 5
```

Depending on your version of Python and `pytest-xdist`, you may need to set `PYTHONHASHSEED` to a fixed value when running in parallel:

```
PYTHONHASHSEED=0 py.test --verbose -n 5
```

An alternative implementation that does not look at command line arguments and works from within Python is to run the tests from the Matplotlib library function `matplotlib.test()`:

```
import matplotlib
matplotlib.test()
```

78.3 Writing a simple test

Many elements of Matplotlib can be tested using standard tests. For example, here is a test from `matplotlib.tests.test_basic`:

```
def test_simple():
    """
    very simple example test
    """
    assert 1 + 1 == 2
```

Pytest determines which functions are tests by searching for files whose names begin with "test_" and then within those files for functions beginning with "test" or classes beginning with "Test".

Some tests have internal side effects that need to be cleaned up after their execution (such as created figures or modified rc params). The pytest fixture `mpl_test_settings()` will automatically clean these up; there is no need to do anything further.

78.4 Random data in tests

Random data can be a very convenient way to generate data for examples, however the randomness is problematic for testing (as the tests must be deterministic!). To work around this set the seed in each test. For numpy use:

```
import numpy as np
np.random.seed(19680801)
```

and Python's random number generator:

```
import random
random.seed(19680801)
```

The seed is John Hunter's birthday.

78.5 Writing an image comparison test

Writing an image based test is only slightly more difficult than a simple test. The main consideration is that you must specify the "baseline", or expected, images in the `image_comparison()` decorator. For example, this test generates a single image and automatically tests it:

```
import numpy as np
import matplotlib
from matplotlib.testing.decorators import image_comparison
import matplotlib.pyplot as plt

@image_comparison(baseline_images=['spines_axes_positions'],
                  extensions=['png'])
def test_spines_axes_positions():
    # SF bug 2852168
    fig = plt.figure()
```

(continues on next page)

(continued from previous page)

```

x = np.linspace(0,2*np.pi,100)
y = 2*np.sin(x)
ax = fig.add_subplot(1,1,1)
ax.set_title('centered spines')
ax.plot(x,y)
ax.spines['right'].set_position(('axes',0.1))
ax.yaxis.set_ticks_position('right')
ax.spines['top'].set_position(('axes',0.25))
ax.xaxis.set_ticks_position('top')
ax.spines['left'].set_color('none')
ax.spines['bottom'].set_color('none')

```

The first time this test is run, there will be no baseline image to compare against, so the test will fail. Copy the output images (in this case `result_images/test_category/spines_axes_positions.png`) to the correct subdirectory of `baseline_images` tree in the source directory (in this case `lib/matplotlib/tests/baseline_images/test_category`). Put this new file under source code revision control (with git add). When rerunning the tests, they should now pass.

The `image_comparison()` decorator defaults to generating png, pdf and svg output, but in interest of keeping the size of the library from ballooning we should only include the svg or pdf outputs if the test is explicitly exercising a feature dependent on that backend.

There are two optional keyword arguments to the `image_comparison` decorator:

- `extensions`: If you only wish to test additional image formats (rather than just png), pass any additional file types in the list of the extensions to test. When copying the new baseline files be sure to only copy the output files, not their conversions to png. For example only copy the files ending in pdf, not in `_pdf.png`.
- `tol`: This is the image matching tolerance, the default 1e-3. If some variation is expected in the image between runs, this value may be adjusted.

78.6 Known failing tests

If you're writing a test, you may mark it as a known failing test with the `pytest.mark.xfail()` decorator. This allows the test to be added to the test suite and run on the buildbots without causing undue alarm. For example, although the following test will fail, it is an expected failure:

```

import pytest

@pytest.mark.xfail
def test_simple_fail():
    '''very simple example test that should fail'''
    assert 1 + 1 == 3

```

Note that the first argument to the `xfail()` decorator is a fail condition, which can be a value such as `True`, `False`, or may be a dynamically evaluated expression. If a condition is supplied, then a

reason must also be supplied with the `reason='message'` keyword argument.

78.7 Creating a new module in `matplotlib.tests`

We try to keep the tests categorized by the primary module they are testing. For example, the tests related to the `matplotlib.text` module are in `test_matplotlib_text.py`.

78.8 Using Travis CI

Travis CI is a hosted CI system "in the cloud".

Travis is configured to receive notifications of new commits to GitHub repos (via GitHub "service hooks") and to run builds or tests when it sees these new commits. It looks for a YAML file called `.travis.yml` in the root of the repository to see how to test the project.

Travis CI is already enabled for the [main matplotlib GitHub repository](#) – for example, see [its Travis page](#).

If you want to enable Travis CI for your personal Matplotlib GitHub repo, simply enable the repo to use Travis CI in either the Travis CI UI or the GitHub UI (Admin | Service Hooks). For details, see [the Travis CI Getting Started page](#). This generally isn't necessary, since any pull request submitted against the main Matplotlib repository will be tested.

Once this is configured, you can see the Travis CI results at https://travis-ci.org/your_GitHub_user_name/matplotlib – here's an [example](#).

78.9 Using tox

Tox is a tool for running tests against multiple Python environments, including multiple versions of Python (e.g., 2.7, 3.5, 3.6) and even different Python implementations altogether (e.g., CPython, PyPy, Jython, etc.)

Testing all versions of Python (2.7, 3.6, 3.7, ...) requires having multiple versions of Python installed on your system and on the PATH. Depending on your operating system, you may want to use your package manager (such as `apt-get`, `yum` or MacPorts) to do this.

tox makes it easy to determine if your working copy introduced any regressions before submitting a pull request. Here's how to use it:

```
$ pip install tox
$ tox
```

You can also run tox on a subset of environments:

```
$ tox -e py27,py36,py37
```

Tox processes everything serially so it can take a long time to test several environments. To speed it up, you might try using a new, parallelized version of tox called detox. Give this a try:

```
$ pip install -U -i http://pypi.testrun.org detox
$ detox
```

Tox is configured using a file called tox.ini. You may need to edit this file if you want to add new environments to test (e.g., py33) or if you want to tweak the dependencies or the way the tests are run. For more info on the tox.ini file, see the [Tox Configuration Specification](#).

WRITING DOCUMENTATION

79.1 Getting started

79.1.1 General file structure

All documentation is built from the `doc/` directory. This directory contains both reStructuredText (ReST; `.rst`) files that contain pages in the documentation and configuration files for [Sphinx](#).

The `.rst` files are kept in `doc/users`, `doc/devel`, `doc/api` and `doc/faq`. The main entry point is `doc/index.rst`, which pulls in the `index.rst` file for the users guide, developers guide, api reference, and FAQs. The documentation suite is built as a single document in order to make the most effective use of cross referencing.

[Sphinx](#) also creates `.rst` files that are staged in `doc/api` from the docstrings of the classes in the Matplotlib library. Except for `doc/api/api_changes/`, these `.rst` files are created when the documentation is built.

Similarly, the contents of `docs/gallery` and `docs/tutorials` are generated by the [Sphinx Gallery](#) from the sources in `examples` and `tutorials`. These sources consist of python scripts that have ReST documentation built into their comments. Don't directly edit the `.rst` files in `docs/gallery` and `docs/tutorials` as they are regenerated when the documentation are built.

79.1.2 Installing dependencies

The documentation for Matplotlib is generated from reStructuredText (ReST) using the [Sphinx](#) documentation generation tool. There are several extra requirements that are needed to build the documentation. They are listed in `doc-requirements.txt` and listed below:

- `Sphinx>=1.3, !=1.5.0, !=1.6.4`
- `colorspacious`
- `IPython`
- `mock`
- `numpydoc>=0.4`
- `Pillow`

- sphinx-gallery>=0.1.12
- graphviz

Note:

- You'll need a minimal working LaTeX distribution for many examples to run.
 - [Graphviz](#) is not a Python package, and needs to be installed separately.
-

79.1.3 Building the docs

The documentation sources are found in the `doc/` directory in the trunk. The configuration file for Sphinx is `doc/conf.py`. It controls which directories Sphinx parses, how the docs are built, and how the extensions are used. To build the documentation in html format, `cd` into `doc/` and run:

```
make html
```

Other useful invocations include

```
# Delete built files. May help if you get errors about missing paths or
# broken links.
make clean

# Build pdf docs.
make latexpdf
```

The `SPHINXOPTS` variable is set to `-W` by default to turn warnings into errors. To unset it, use

```
make SPHINXOPTS= html
```

You can use the `O` variable to set additional options:

- `make O=-j4 html` runs a parallel build with 4 processes.
- `make O=-Dplot_formats=png:100 html` saves figures in low resolution.
- `make O=-Dplot_gallery=0 html` skips the gallery build.

Multiple options can be combined using e.g. `make O='-j4 -Dplot_gallery=0' html`.

On Windows, options needs to be set as environment variables, e.g. `set O=-W -j4 & make html`.

79.2 Writing ReST pages

Most documentation is either in the docstring of individual classes and methods, in explicit `.rst` files, or in examples and tutorials. All of these use the [ReST](#) syntax. Users should look at the [ReST](#) documentation for a full description. But some specific hints and conventions Matplotlib uses are useful for creating documentation.

79.2.1 Formatting and style conventions

It is useful to strive for consistency in the Matplotlib documentation. Here are some formatting and style conventions that are used.

Section name formatting

For everything but top-level chapters, use Upper lower for section titles, e.g., Possible hangups rather than Possible Hangups

Function arguments

Function arguments and keywords within docstrings should be referred to using the **emphasis** role. This will keep Matplotlib's documentation consistent with Python's documentation:

Here is a description of **argument**

Do not use the ``default role``:

Do not describe ``argument`` like this. As per the next section, this syntax will (unsuccessfully) attempt to resolve the argument as a link to a class or method in the library.

nor the ```literal``` role:

Do not describe ```argument``` like this.

79.2.2 Referring to other documents and sections

Sphinx allows internal references between documents.

Documents can be linked with the `:doc:` directive:

See the `:doc:`/faq/installing_faq``

See the tutorial `:doc:`/tutorials/introductory/sample_plots``

See the example `:doc:`/gallery/lines_bars_and_markers/simple_plot``

will render as:

See the Installation

See the tutorial Sample plots in Matplotlib

See the example /gallery/lines_bars_and_markers/simple_plot

Sections can also be given reference names. For instance from the Installation link:

```
.. __clean-install:
```

How to completely remove Matplotlib

=====

Occasionally, problems with Matplotlib can be solved with a clean...

and refer to it using the standard reference syntax:

```
See :ref:`clean-install`
```

will give the following link: How to completely remove Matplotlib

To maximize internal consistency in section labeling and references, use hyphen separated, descriptive labels for section references. Keep in mind that contents may be reorganized later, so avoid top level names in references like user or devel or faq unless necessary, because for example the FAQ "what is a backend?" could later become part of the users guide, so the label:

```
.. __what-is-a-backend:
```

is better than:

```
.. __faq-backend:
```

In addition, since underscores are widely used by Sphinx itself, use hyphens to separate words.

79.2.3 Referring to other code

To link to other methods, classes, or modules in Matplotlib you can use back ticks, for example:

```
`~matplotlib.collections.LineCollection`
```

returns a link to the documentation of `LineCollection`. For the full path of the class to be shown, omit the tilde:

```
`matplotlib.collections.LineCollection`
```

to get `matplotlib.collections.LineCollection`. It is often not necessary to fully specify the class hierarchy unless there is a namespace collision between two packages:

```
`~.LineCollection`
```

links just as well: `LineCollection`.

Other packages can also be linked via intersphinx:

```
`numpy.mean`
```

will return this link: [numpy.mean](#). This works for Python, Numpy, Scipy, and Pandas (full list is in `doc/conf.py`). Sometimes it is tricky to get external Sphinx linking to work; to check that something exists to link to the following shell command outputs a list of all objects that can be referenced (in this case for Numpy):

```
python -m sphinx.ext.intersphinx 'https://docs.scipy.org/doc/numpy/objects.inv'
```

79.2.4 Including figures and files

Image files can directly included in pages with the `image::` directive. e.g., `users/navigation_toolbar.rst` displays the toolbar icons with a call to a static image:

```
.. image:: ../_static/toolbar.png
```

as rendered on the page: [Interactive navigation](#).

Files can be included verbatim. For instance the `matplotlibrc` file is important for customizing Matplotlib, and is included verbatim in the tutorial in [Customizing Matplotlib with style sheets and rcParams](#):

```
.. literalinclude:: ../_static/matplotlibrc
```

This is rendered at the bottom of [Customizing Matplotlib with style sheets and rcParams](#). Note that this is in a tutorial; see [Writing examples and tutorials](#) below.

The examples directory is also copied to `doc/gallery` by `sphinx-gallery`, so plots from the examples directory can be included using

```
.. plot:: gallery/pylab_examples/simple_plot.py
```

Note that the python script that generates the plot is referred to, rather than any plot that is created. `Sphinx-gallery` will provide the correct reference when the documentation is built.

79.3 Writing docstrings

Much of the documentation lives in "docstrings". These are comment blocks in source code that explain how the code works. All new or edited docstrings should conform to the [numpydoc](#) guidelines. These split the docstring into a number of sections - see the [numpy documentation howto](#) for more details and a guide to how docstrings should be formatted. Much of the [ReST](#) syntax discussed above ([:ref:writing-rest-pages](#)) can be used for links and references. These docstrings eventually populate the `doc/api` directory and form the reference documentation for the library.

79.3.1 Example docstring

An example docstring looks like:

```
def hlines(self, y, xmin, xmax, colors='k', linestyle='solid',
            label='', **kwargs):
    """
    Plot horizontal lines at each *y* from *xmin* to *xmax*.

    Parameters
    -----
    y : scalar or sequence of scalar
        y-indexes where to plot the lines.

    xmin, xmax : scalar or 1D array_like
        Respective beginning and end of each line. If scalars are
        provided, all lines will have same length.

    colors : array_like of colors, optional, default: 'k'

    linestyle : {'solid', 'dashed', 'dashdot', 'dotted'}, optional

    label : string, optional, default: ''

    Returns
    -----
    lines : `~matplotlib.collections.LineCollection`

    Other Parameters
    -----
    **kwargs : `~matplotlib.collections.LineCollection` properties.

    See also
    -----
    vlines : vertical lines
    axhline: horizontal line across the axes
    """
```

See the `hlines` documentation for how this renders.

The [Sphinx](#) website also contains plenty of [documentation](#) concerning ReST markup and working with Sphinx in general.

Note: Some parts of the documentation do not yet conform to the current documentation style. If in doubt, follow the rules given here and not what you may see in the source code. Pull requests updating docstrings to the current style are very welcome.

79.3.2 Formatting conventions

The basic docstring conventions are covered in the [numpy documentation howto](#) and the [Sphinx documentation](#). Some Matplotlib-specific formatting conventions to keep in mind:

- Matplotlib does not have a convention whether to use single-quotes or double-quotes. There is a mixture of both in the current code.
- Long parameter lists should be wrapped using a `\` for continuation and starting on the new line without any indent:

```
def add_axes(self, *args, **kwargs):
    """
    ...

    Parameters
    -----
    projection :
        {'aitoff', 'hammer', 'lambert', 'mollweide', 'polar', \
'rectilinear'}, optional
        The projection type of the axes.

    ...
    """
```

Alternatively, you can describe the valid parameter values in a dedicated section of the docstring.

- Generally, do not add markup to types for Parameters and Returns. This is usually not needed because Sphinx will link them automatically and would unnecessarily clutter the docstring. However, it does seem to fail in some situations. If you encounter such a case, you are allowed to add markup:

```
Returns
-----
lines : ~matplotlib.collections.LineCollection`
```

- rcParams can be referenced with the custom `:rc:` role: `:rc:`foo`` yields `rcParams["foo"]`.

79.3.3 Deprecated formatting conventions

- Formerly, we have used square brackets for explicit parameter lists `['solid' | 'dashed' | 'dotted']`. With numpydoc we have switched to their standard using curly braces `{'solid', 'dashed', 'dotted'}`.

79.3.4 Linking to other code

To link to other methods, classes, or modules in Matplotlib you can encase the name to refer to in back ticks, for example:

```
`~matplotlib.collections.LineCollection`
```

It is also possible to add links to code in Python, Numpy, Scipy, or Pandas. Sometimes it is tricky to get external Sphinx linking to work; to check that a something exists to link to the following shell command outputs a list of all objects that can be referenced (in this case for Numpy):

```
python -m sphinx.ext.intersphinx 'https://docs.scipy.org/doc/numpy/objects.inv'
```

79.3.5 Function arguments

Function arguments and keywords within docstrings should be referred to using the **emphasis** role. This will keep Matplotlib's documentation consistent with Python's documentation:

```
Here is a description of *argument*
```

Do not use the ``default role``:

```
Do not describe `argument` like this.
```

nor the ```literal``` role:

```
Do not describe ``argument`` like this.
```

79.3.6 Setters and getters

Artist properties are implemented using setter and getter methods (because Matplotlib predates the introductions of the `property` decorator in Python). By convention, these setters and getters are named `set_PROPERTYNAME` and `get_PROPERTYNAME`; the list of properties thusly defined on an artist and their values can be listed by the `setp` and `getp` functions.

Property setter methods should indicate the values they accept using a (legacy) special block in the docstring, starting with `ACCEPTS`, as follows:

```
# in lines.py
def set_linestyle(self, linestyle):
    """
    Set the linestyle of the line

    ACCEPTS: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | '' | '' ]
    """
```

The ACCEPTS block is used to render a table of all properties and their acceptable values in the docs; it can also be displayed using, e.g., `plt.setp(Line2D)` (all properties) or `plt.setp(Line2D, 'linestyle')` (just one property).

There are cases in which the ACCEPTS string is not useful in the generated Sphinx documentation, e.g. if the valid parameters are already defined in the numpydoc parameter list. You can hide the ACCEPTS string from Sphinx by making it a ReST comment (i.e. use `.. ACCEPTS:`):

```
def set_linestyle(self, linestyle):
    """
    An ACCEPTS string invisible to Sphinx.

    .. ACCEPTS: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
    """
```

79.3.7 Keyword arguments

Note: The information in this section is being actively discussed by the development team, so use the docstring interpolation only if necessary. This section has been left in place for now because this interpolation is part of the existing documentation.

Since Matplotlib uses a lot of pass-through kwargs, e.g., in every function that creates a line (`plot`, `semilogx`, `semilogy`, etc...), it can be difficult for the new user to know which kwargs are supported. Matplotlib uses a docstring interpolation scheme to support documentation of every function that takes a `**kwargs`. The requirements are:

1. single point of configuration so changes to the properties don't require multiple docstring edits.
2. as automated as possible so that as properties change, the docs are updated automatically.

The function `matplotlib.artist.kwdoc` and the decorator `matplotlib.docstring.dedent_interpd` facilitate this. They combine Python string interpolation in the docstring with the Matplotlib artist introspection facility that underlies `setp` and `getp`. The `kwdoc` function gives the list of properties as a docstring. In order to use this in another docstring, first update the `matplotlib.docstring.interpd` object, as seen in this example from `matplotlib.lines`:

```
# in lines.py
docstring.interpd.update(Line2D=artist.kwdoc(Line2D))
```

Then in any function accepting `Line2D` pass-through kwargs, e.g., `matplotlib.axes.Axes.plot`:

```
# in axes.py
@docstring.dedent_interpd
def plot(self, *args, **kwargs):
    """
    Some stuff omitted
```

(continues on next page)

(continued from previous page)

The kwargs are Line2D properties:

```
%(Line2D)s
```

```
kwargs scalex and scaley, if defined, are passed on
to autoscale_view to determine whether the x and y axes are
autoscaled; default True. See Axes.autoscale_view for more
information
"""
```

Note there is a problem for Artist `__init__` methods, e.g., `matplotlib.patches.Patch.__init__`, which supports Patch kwargs, since the artist inspector cannot work until the class is fully defined and we can't modify the `Patch.__init__.__doc__` docstring outside the class definition. There are some manual hacks in this case, violating the "single entry point" requirement above – see the `docstring.interpd.update` calls in `matplotlib.patches`.

79.3.8 Adding figures

As above (see Including figures and files), figures in the examples gallery can be referenced with a `:plot:` directive pointing to the python script that created the figure. For instance the legend docstring references the file `examples/api/legend.py`:

```
"""
...

Examples
-----

.. plot:: gallery/api/legend.py
"""
```

Note that `examples/api/legend.py` has been mapped to `gallery/api/legend.py`, a redirection that may be fixed in future re-organization of the docs.

Plots can also be directly placed inside docstrings. Details are in Plot directive documentation. A short example is:

```
"""
...

Examples
-----

.. plot::
   import matplotlib.image as mpimg
   img = mpimg.imread('_static/stinkbug.png')
```

(continues on next page)

(continued from previous page)

```
imgplot = plt.imshow(img)
"""
```

An advantage of this style over referencing an example script is that the code will also appear in interactive docstrings.

79.4 Writing examples and tutorials

Examples and tutorials are python scripts that are run by [Sphinx Gallery](#) to create a gallery of images in the `/doc/gallery` and `/doc/tutorials` directories respectively. To exclude an example from having an plot generated insert `"sgskip"` somewhere in the filename.

The format of these files is relatively straightforward. Properly formatted comment blocks are treated as [ReST](#) text, the code is displayed, and figures are put into the built page.

For instance the example `/gallery/lines_bars_and_markers/simple_plot` example is generated from `/examples/lines_bars_and_markers/simple_plot.py`, which looks like:

```
"""
=====
Simple Plot
=====

Create a simple plot.
"""
import matplotlib.pyplot as plt
import numpy as np

# Data for plotting
t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2 * np.pi * t)

# Note that using plt.subplots below is equivalent to using
# fig = plt.figure and then ax = fig.add_subplot(111)
fig, ax = plt.subplots()
ax.plot(t, s)

ax.set(xlabel='time (s)', ylabel='voltage (mV)',
       title='About as simple as it gets, folks')
ax.grid()
plt.show()
```

The first comment block is treated as [ReST](#) text. The other comment blocks render as comments in `/gallery/lines_bars_and_markers/simple_plot`.

Tutorials are made with the exact same mechanism, except they are longer, and typically have more than one comment block (i.e. [Usage Guide](#)). The first comment block can be the same as the

example above. Subsequent blocks of ReST text are delimited by a line of `###` characters:

```
"""
=====
Simple Plot
=====

Create a simple plot.
"""
...
ax.grid()
plt.show()

#####
→#####
# Second plot
# =====
#
# This is a second plot that is very nice

fig, ax = plt.subplots()
ax.plot(np.sin(range(50)))
```

In this way text, code, and figures are output in a "notebook" style.

79.5 Miscellaneous

79.5.1 Adding animations

There is a Matplotlib Google/Gmail account with username `mplgithub` which was used to setup the github account but can be used for other purposes, like hosting Google docs or Youtube videos. You can embed a Matplotlib animation in the docs by first saving the animation as a movie using `matplotlib.animation.Animation.save()`, and then uploading to [matplotlib's Youtube channel](#) and inserting the embedding string youtube provides like:

```
.. raw:: html

<iframe width="420" height="315"
  src="http://www.youtube.com/embed/32cjc6V0OZY"
  frameborder="0" allowfullscreen>
</iframe>
```

An example save command to generate a movie looks like this

```
ani = animation.FuncAnimation(fig, animate, np.arange(1, len(y)),
    interval=25, blit=True, init_func=init)
```

(continues on next page)

(continued from previous page)

```
ani.save('double_pendulum.mp4', fps=15)
```

Contact Michael Droettboom for the login password to upload youtube videos of google docs to the mplgithub account.

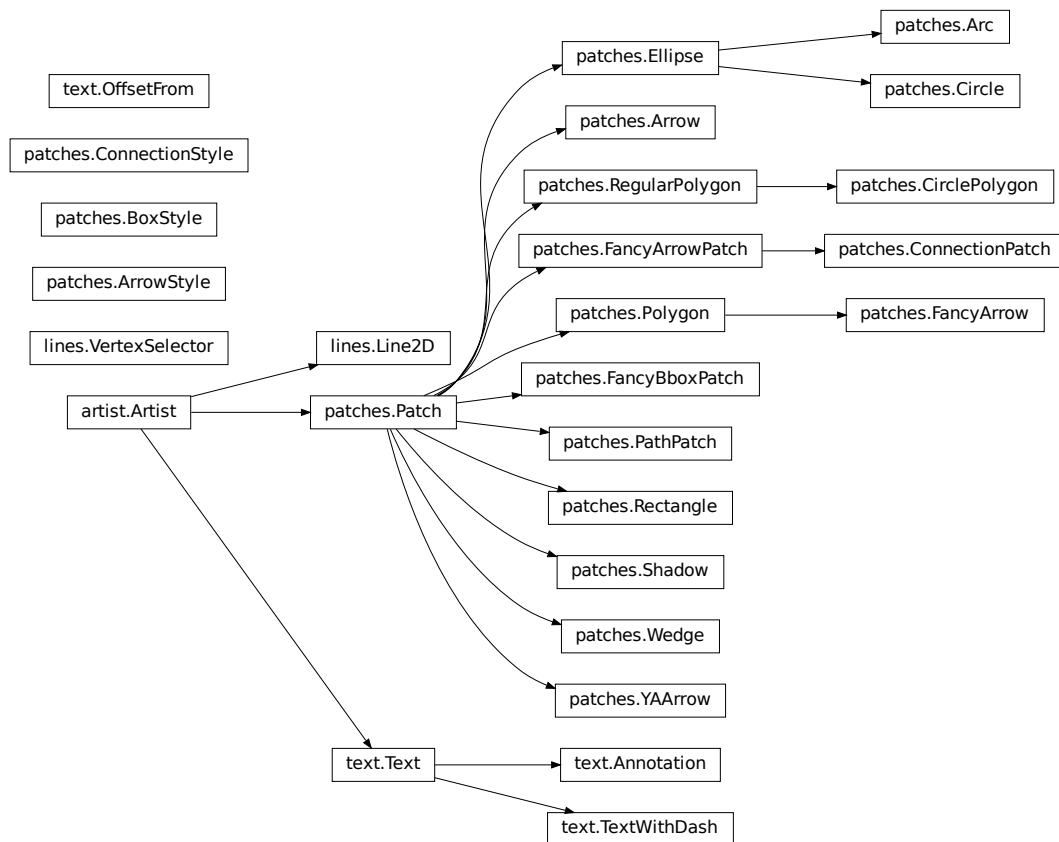
79.5.2 Generating inheritance diagrams

Class inheritance diagrams can be generated with the `inheritance-diagram` directive. To use it, provide the directive with a number of class or module names (separated by whitespace). If a module name is provided, all classes in that module will be used. All of the ancestors of these classes will be included in the inheritance diagram.

A single option is available: `parts` controls how many of parts in the path to the class are shown. For example, if `parts == 1`, the class `matplotlib.patches.Patch` is shown as `Patch`. If `parts == 2`, it is shown as `patches.Patch`. If `parts == 0`, the full path is shown.

Example:

```
.. inheritance-diagram:: matplotlib.patches matplotlib.lines matplotlib.text
   :parts: 2
```



79.5.3 Emacs helpers

There is an emacs mode `rst.el` which automates many important ReST tasks like building and updating table-of-contents, and promoting or demoting section headings. Here is the basic .emacs configuration:

```
(require 'rst)
(setq auto-mode-alist
  (append '(("\\.txt$" . rst-mode)
            ("\\.rst$" . rst-mode)
            ("\\.rest$" . rst-mode)) auto-mode-alist))
```

Some helpful functions:

C-c TAB - rst-toc-insert

Insert table of contents at point

(continues on next page)

(continued from previous page)

C-c C-u - rst-toc-update

Update the table of contents at point

C-c C-l rst-shift-region-left

Shift region to the left

C-c C-r rst-shift-region-right

Shift region to the right

PLOT DIRECTIVE DOCUMENTATION

A directive for including a matplotlib plot in a Sphinx document.

By default, in HTML output, plot will include a .png file with a link to a high-res .png and .pdf. In LaTeX output, it will include a .pdf.

The source code for the plot may be included in one of three ways:

1. A path to a source file as the argument to the directive:

```
.. plot:: path/to/plot.py
```

When a path to a source file is given, the content of the directive may optionally contain a caption for the plot:

```
.. plot:: path/to/plot.py
```

This is the caption for the plot

Additionally, one may specify the name of a function to call (with no arguments) immediately after importing the module:

```
.. plot:: path/to/plot.py plot_function1
```

2. Included as inline content to the directive:

```
.. plot::  
  
    import matplotlib.pyplot as plt  
    import matplotlib.image as mpimg  
    import numpy as np  
    img = mpimg.imread('_static/stinkbug.png')  
    imgplot = plt.imshow(img)
```

3. Using doctest syntax:

```
.. plot::  
    A plotting example:
```

(continues on next page)

(continued from previous page)

```
>>> import matplotlib.pyplot as plt
>>> plt.plot([1,2,3], [4,5,6])
```

80.1 Options

The plot directive supports the following options:

- `format` [{`'python'`, `'doctest'`}] Specify the format of the input
- `include-source` [bool] Whether to display the source code. The default can be changed using the `plot_include_source` variable in `conf.py`
- `encoding` [str] If this source file is in a non-UTF8 or non-ASCII encoding, the encoding must be specified using the `:encoding:` option. The encoding will not be inferred using the `-*- coding -*-` metacomment.
- `context` [bool or str] If provided, the code will be run in the context of all previous plot directives for which the `:context:` option was specified. This only applies to inline code plot directives, not those run from files. If the `:context: reset` option is specified, the context is reset for this and future plots, and previous figures are closed prior to running the code. `:context:close-figs` keeps the context but closes previous figures before running the code.
- `nofigs` [bool] If specified, the code block will be run, but no figures will be inserted. This is usually useful with the `:context:` option.

Additionally, this directive supports all of the options of the image directive, except for `target` (since plot will add its own target). These include `alt`, `height`, `width`, `scale`, `align` and `class`.

80.2 Configuration options

The plot directive has the following configuration options:

- `plot_include_source` Default value for the include-source option
- `plot_html_show_source_link` Whether to show a link to the source in HTML.
- `plot_pre_code` Code that should be executed before each plot. If not specified or `None` it will default to a string containing:

```
import numpy as np
from matplotlib import pyplot as plt
```

- `plot_basedir` Base directory, to which `plot::` file names are relative to. (If `None` or empty, file names are relative to the directory where the file containing the directive is.)
- `plot_formats` File formats to generate. List of tuples or strings:

`[(suffix, dpi), suffix, ...]`

that determine the file format and the DPI. For entries whose DPI was omitted, sensible defaults are chosen. When passing from the command line through `sphinx_build` the list should be passed as `suffix:dpi,suffix:dpi,`

`plot_html_show_formats` Whether to show links to the files in HTML.

`plot_rcparams` A dictionary containing any non-standard rcParams that should be applied before each plot.

`plot_apply_rcparams` By default, rcParams are applied when context option is not used in a plot directive. This configuration option overrides this behavior and applies rcParams before each plot.

`plot_working_directory` By default, the working directory will be changed to the directory of the example, so the code can get at its data files, if any. Also its path will be added to `sys.path` so it can import any helper modules sitting beside it. This configuration option can be used to specify a central directory (also added to `sys.path`) where data files and helper modules for all code are located.

`plot_template` Provide a customized template for preparing restructured text.

exception `matplotlib.sphinxext.plot_directive.PlotError`

`matplotlib.sphinxext.plot_directive.mark_plot_labels(app, document)`

To make plots referenceable, we need to move the reference from the "htmlonly" (or "latex-only") node to the actual figure node itself.

`matplotlib.sphinxext.plot_directive.out_of_date(original, derived)`

Returns True if derivative is out-of-date wrt original, both of which are full file paths.

`matplotlib.sphinxext.plot_directive.plot_directive(name, arguments, options, content, lineno, content_offset, block_text, state, state_machine)`

Implementation of the `.. plot::` directive.

See the module docstring for details.

`matplotlib.sphinxext.plot_directive.remove_coding(text)`

Remove the coding comment, which `six.exec_` doesn't like.

`matplotlib.sphinxext.plot_directive.render_figures(code, code_path, output_dir, output_base, context, function_name, config, context_reset=False, close_figs=False)`

Run a pyplot script and save the images in `output_dir`.

Save the images under `output_dir` with file names derived from `output_base`

`matplotlib.sphinxext.plot_directive.run_code(code, code_path, ns=None, function_name=None)`

Import a Python module from a path, and run the function given by name, if `function_name`

is not None.

`matplotlib.sphinxext.plot_directive.split_code_at_show(text)`

Split code at `plt.show()`

`matplotlib.sphinxext.plot_directive.unescape_doctest(text)`

Extract code from a piece of text, which contains either Python code or doctests.

DEVELOPER'S GUIDE FOR CREATING SCALES AND TRANSFORMATIONS

Matplotlib supports the addition of custom procedures that transform the data before it is displayed.

There is an important distinction between two kinds of transformations. Separable transformations, working on a single dimension, are called "scales", and non-separable transformations, that handle data in two or more dimensions at a time, are called "projections".

From the user's perspective, the scale of a plot can be set with `set_xscale()` and `set_yscale()`. Projections can be chosen using the `projection` keyword argument to the `plot()` or `subplot()` functions, e.g.:

```
plot(x, y, projection="custom")
```

This document is intended for developers and advanced users who need to create new scales and projections for matplotlib. The necessary code for scales and projections can be included anywhere: directly within a plot script, in third-party code, or in the matplotlib source tree itself.

81.1 Creating a new scale

Adding a new scale consists of defining a subclass of `matplotlib.scale.ScaleBase`, that includes the following elements:

- A transformation from data coordinates into display coordinates.
- An inverse of that transformation. This is used, for example, to convert mouse positions from screen space back into data space.
- A function to limit the range of the axis to acceptable values (`limit_range_for_scale()`). A log scale, for instance, would prevent the range from including values less than or equal to zero.
- Locators (major and minor) that determine where to place ticks in the plot, and optionally, how to adjust the limits of the plot to some "good" values. Unlike `limit_range_for_scale()`, which is always enforced, the range setting here is only used when automatically setting the range of the plot.
- Formatters (major and minor) that specify how the tick labels should be drawn.

Once the class is defined, it must be registered with matplotlib so that the user can select it.

A full-fledged and heavily annotated example is in `examples/api/custom_scale_example.py`. There are also some classes in `matplotlib.scale` that may be used as starting points.

81.2 Creating a new projection

Adding a new projection consists of defining a projection axes which subclasses `matplotlib.axes.Axes` and includes the following elements:

- A transformation from data coordinates into display coordinates.
- An inverse of that transformation. This is used, for example, to convert mouse positions from screen space back into data space.
- Transformations for the gridlines, ticks and ticklabels. Custom projections will often need to place these elements in special locations, and matplotlib has a facility to help with doing so.
- Setting up default values (overriding `cla()`), since the defaults for a rectilinear axes may not be appropriate.
- Defining the shape of the axes, for example, an elliptical axes, that will be used to draw the background of the plot and for clipping any data elements.
- Defining custom locators and formatters for the projection. For example, in a geographic projection, it may be more convenient to display the grid in degrees, even if the data is in radians.
- Set up interactive panning and zooming. This is left as an "advanced" feature left to the reader, but there is an example of this for polar plots in `matplotlib.projections.polar`.
- Any additional methods for additional convenience or features.

Once the projection axes is defined, it can be used in one of two ways:

- By defining the class attribute name, the projection axes can be registered with `matplotlib.projections.register_projection()` and subsequently simply invoked by name:

```
plt.axes(projection='my_proj_name')
```

- For more complex, parameterisable projections, a generic "projection" object may be defined which includes the method `__as_mpl_axes`. `__as_mpl_axes` should take no arguments and return the projection's axes subclass and a dictionary of additional arguments to pass to the subclass' `__init__` method. Subsequently a parameterised projection can be initialised with:

```
plt.axes(projection=MyProjection(param1=param1_value))
```

where `MyProjection` is an object which implements a `__as_mpl_axes` method.

A full-fledged and heavily annotated example is in `examples/api/custom_projection_example.py`. The polar plot functionality in `matplotlib.projections.polar` may also be of interest.

81.3 API documentation

- `matplotlib.scale`
- `matplotlib.projections`
- `matplotlib.projections.polar`

DEVELOPER'S TIPS FOR WRITING CODE FOR PYTHON 2 AND 3

As of matplotlib 1.4, the `six` library is used to support Python 2 and 3 from a single code base. The 2to3 tool is no longer used.

This document describes some of the issues with that approach and some recommended solutions. It is not a complete guide to Python 2 and 3 compatibility.

82.1 Welcome to the `__future__`

The top of every py file should include the following:

```
from __future__ import (absolute_import, division,
                        print_function, unicode_literals)
import six
```

This will make the Python 2 interpreter behave as close to Python 3 as possible.

All matplotlib files should also import `six`, whether they are using it or not, just to make moving code between modules easier, as `six` gets used a lot.

82.2 Finding places to use `six`

The only way to make sure code works on both Python 2 and 3 is to make sure it is covered by unit tests.

However, the 2to3 commandline tool can also be used to locate places that require special handling with `six`.

(The `modernize` tool may also be handy, though I've never used it personally).

The `six` documentation serves as a good reference for the sorts of things that need to be updated.

82.3 The dreaded `\u` escapes

When `from __future__ import unicode_literals` is used, all string literals (not preceded with a `b`) will become unicode literals.

Normally, one would use "raw" string literals to encode strings that contain a lot of slashes that we don't want Python to interpret as special characters. A common example in matplotlib is when it deals with TeX and has to represent things like `r"\usepackage{foo}"`. Unfortunately, on Python 2 there is no way to represent `u` in a raw unicode string literal, since it will always be interpreted as the start of a unicode character escape, such as `u20af`. The only solution is to use a regular (non-raw) string literal and repeat all slashes, e.g. `"\\usepackage{foo}"`.

The following shows the problem on Python 2:

```
>>> ur'\u'
File "<stdin>", line 1
SyntaxError: (unicode error) 'rawunicodeescape' codec can't decode bytes in
position 0-1: truncated \uXXXX
>>> ur'\\u'
u'\\\\u'
>>> u'\u'
File "<stdin>", line 1
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in
position 0-1: truncated \uXXXX escape
>>> u'\\u'
u'\\u'
```

This bug has been fixed in Python 3, however, we can't take advantage of that and still support Python 2:

```
>>> r'\u'
'\\u'
>>> r'\\u'
'\\\\u'
>>> '\u'
File "<stdin>", line 1
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in
position 0-1: truncated \uXXXX escape
>>> '\\u'
'\\u'
```

82.4 Iteration

The behavior of the methods for iterating over the items, values and keys of a dictionary has changed in Python 3. Additionally, other built-in functions such as `zip`, `range` and `map` have changed to return iterators rather than temporary lists.

In many cases, the performance implications of iterating vs. creating a temporary list won't matter, so it's tempting to use the form that is simplest to read. However, that results in code that behaves differently on Python 2 and 3, leading to subtle bugs that may not be detected by the regression tests. Therefore, unless the loop in question is provably simple and doesn't call into other code, the six versions that ensure the same behavior on both Python 2 and 3 should be used. The following

table shows the mapping of equivalent semantics between Python 2, 3 and six for `dict.items()`:

Python 2	Python 3	six
<code>d.items()</code>	<code>list(d.items())</code>	<code>list(six.iteritems(d))</code>
<code>d.iteritems()</code>	<code>d.items()</code>	<code>six.iteritems(d)</code>

82.5 Numpy-specific things

When specifying dtypes, all strings must be byte strings on Python 2 and unicode strings on Python 3. The best way to handle this is to force cast them using `str()`. The same is true of structure specifiers in the `struct` built-in module.

WORKING WITH MATPLOTLIB SOURCE CODE

Contents:

83.1 Introduction

These pages describe a [git](#) and [github](#) workflow for the [Matplotlib](#) project.

There are several different workflows here, for different ways of working with Matplotlib.

This is not a comprehensive git reference, it's just a workflow for our own project. It's tailored to the github hosting service. You may well find better or quicker ways of getting stuff done with git, but these should get you started.

For general resources for learning git, see [git resources](#).

83.2 Install git

83.2.1 Overview

Debian / Ubuntu	<code>sudo apt-get install git</code>
Fedora	<code>sudo yum install git</code>
Windows	Download and install msysGit
OS X	Use the git-osx-installer

83.2.2 In detail

See the [git](#) page for the most recent information.

Have a look at the [github install help](#) pages available from [github help](#)

There are good instructions here: <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

83.3 Following the latest source

These are the instructions if you just want to follow the latest Matplotlib source, but you don't need to do any development for now.

The steps are:

- Install git
- get local copy of the [Matplotlib github](#) git repository
- update local copy from time to time

83.3.1 Get the local copy of the code

From the command line:

```
git clone git://github.com/matplotlib/matplotlib.git
```

You now have a copy of the code tree in the new matplotlib directory.

83.3.2 Updating the code

From time to time you may want to pull down the latest code. Do this with:

```
cd matplotlib
git pull
```

The tree in matplotlib will now have the latest changes from the initial repository.

83.4 Making a patch

You've discovered a bug or something else you want to change in [Matplotlib](#) .. — excellent!

You've worked out a way to fix it — even better!

You want to tell us about it — best of all!

The easiest way is to make a patch or set of patches. Here we explain how. Making a patch is the simplest and quickest, but if you're going to be doing anything more than simple quick things, please consider following the Git for development model instead.

83.4.1 Making patches

Overview

```
# tell git who you are
git config --global user.email you@yourdomain.example.com
git config --global user.name "Your Name Comes Here"
# get the repository if you don't have it
git clone git://github.com/matplotlib/matplotlib.git
# make a branch for your patching
cd matplotlib
git branch the-fix-im-thinking-of
git checkout the-fix-im-thinking-of
# hack, hack, hack
# Tell git about any new files you've made
git add somewhere/tests/test_my_bug.py
# commit work in progress as you go
git commit -am 'BF - added tests for Funny bug'
# hack hack, hack
git commit -am 'BF - added fix for Funny bug'
# make the patch files
git format-patch -M -C master
```

Then, send the generated patch files to the [Matplotlib mailing list](#) — where we will thank you warmly.

In detail

1. Tell git who you are so it can label the commits you've made:

```
git config --global user.email you@yourdomain.example.com
git config --global user.name "Your Name Comes Here"
```

2. If you don't already have one, clone a copy of the [Matplotlib](#) repository:

```
git clone git://github.com/matplotlib/matplotlib.git
cd matplotlib
```

3. Make a 'feature branch'. This will be where you work on your bug fix. It's nice and safe and leaves you with access to an unmodified copy of the code in the main branch:

```
git branch the-fix-im-thinking-of
git checkout the-fix-im-thinking-of
```

4. Do some edits, and commit them as you go:

```
# hack, hack, hack
# Tell git about any new files you've made
git add somewhere/tests/test_my_bug.py
# commit work in progress as you go
```

(continues on next page)

(continued from previous page)

```
git commit -am 'BF - added tests for Funny bug'
# hack hack, hack
git commit -am 'BF - added fix for Funny bug'
```

Note the -am options to commit. The m flag just signals that you're going to type a message on the command line. The a flag — you can just take on faith — or see [why the -a flag?](#).

5. When you have finished, check you have committed all your changes:

```
git status
```

6. Finally, make your commits into patches. You want all the commits since you branched from the master branch:

```
git format-patch -M -C master
```

You will now have several files named for the commits:

```
0001-BF-added-tests-for-Funny-bug.patch
0002-BF-added-fix-for-Funny-bug.patch
```

Send these files to the [Matplotlib mailing list](#).

When you are done, to switch back to the main copy of the code, just return to the master branch:

```
git checkout master
```

83.4.2 Moving from patching to development

If you find you have done some patches, and you have one or more feature branches, you will probably want to switch to development mode. You can do this with the repository you have.

Fork the [Matplotlib](#) repository on github — Making your own copy (fork) of Matplotlib. Then:

```
# checkout and refresh master branch from main repo
git checkout master
git pull origin master
# rename pointer to main repository to 'upstream'
git remote rename origin upstream
# point your repo to default read / write to your fork on github
git remote add origin git@github.com:your-user-name/matplotlib.git
# push up any branches you've made and want to keep
git push origin the-fix-im-thinking-of
```

Then you can, if you want, follow the Development workflow.

83.5 Git for development

Contents:

83.5.1 Making your own copy (fork) of Matplotlib

You need to do this only once. The instructions here are very similar to the instructions at <https://help.github.com/forking/> — please see that page for more detail. We’re repeating some of it here just to give the specifics for the [Matplotlib](#) project, and to suggest some default names.

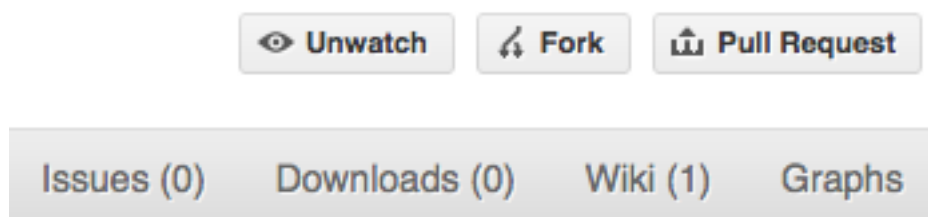
Set up and configure a github account

If you don’t have a github account, go to the github page, and make one.

You then need to configure your account to allow write access — see the [Generating SSH keys help](#) on [github help](#).

Create your own forked copy of Matplotlib

1. Log into your github account.
2. Go to the [Matplotlib](#) github home at [Matplotlib github](#).
3. Click on the fork button:



Now, after a short pause, you should find yourself at the home page for your own forked copy of [Matplotlib](#).

83.5.2 Set up your fork

First you follow the instructions for [Making your own copy \(fork\) of Matplotlib](#).

Overview

```
git clone https://github.com/your-user-name/matplotlib.git
cd matplotlib
git remote add upstream git://github.com/matplotlib/matplotlib.git
```


In detail

Clone your fork

1. Clone your fork to the local computer with `git clone https://github.com/your-user-name/matplotlib.git`
2. Investigate. Change directory to your new repo: `cd matplotlib`. Then `git branch -a` to show you all branches. You'll get something like:

```
* master
remotes/origin/master
```

This tells you that you are currently on the master branch, and that you also have a remote connection to origin/master. What remote repository is remote/origin? Try `git remote -v` to see the URLs for the remote. They will point to your github fork.

Now you want to connect to the upstream [Matplotlib github](#) repository, so you can merge in changes from trunk.

Linking your repository to the upstream repo

```
cd matplotlib
git remote add upstream git://github.com/matplotlib/matplotlib.git
```

upstream here is just the arbitrary name we're using to refer to the main [Matplotlib](#) repository at [Matplotlib github](#).

Note that we've used `git://` for the URL rather than `https://` or `git@`. The `git://` URL is read only. This means we that we can't accidentally (or deliberately) write to the upstream repo, and we are only going to use it to merge into our own code.

Just for your own satisfaction, show yourself that you now have a new 'remote', with `git remote -v` show, giving you something like:

```
upstream    git://github.com/matplotlib/matplotlib.git (fetch)
upstream    git://github.com/matplotlib/matplotlib.git (push)
origin      https://github.com/your-user-name/matplotlib.git (fetch)
origin      https://github.com/your-user-name/matplotlib.git (push)
```

83.5.3 Configure git

Overview

Your personal git configurations are saved in the `.gitconfig` file in your home directory.

Here is an example `.gitconfig` file:

```
[user]
    name = Your Name
    email = you@yourdomain.example.com

[alias]
    ci = commit -a
    co = checkout
    st = status
    stat = status
    br = branch
    wdiff = diff --color-words

[core]
    editor = vim

[merge]
    summary = true
```

You can edit this file directly or you can use the git config --global command:

```
git config --global user.name "Your Name"
git config --global user.email you@yourdomain.example.com
git config --global alias.ci "commit -a"
git config --global alias.co checkout
git config --global alias.st "status -a"
git config --global alias.stat "status -a"
git config --global alias.br branch
git config --global alias.wdiff "diff --color-words"
git config --global core.editor vim
git config --global merge.summary true
```

To set up on another computer, you can copy your ~/.gitconfig file, or run the commands above.

In detail

user.name and user.email

It is good practice to tell `git` who you are, for labeling any changes you make to the code. The simplest way to do this is from the command line:

```
git config --global user.name "Your Name"
git config --global user.email you@yourdomain.example.com
```

This will write the settings into your git configuration file, which should now contain a user section with your name and email:

```
[user]
  name = Your Name
  email = you@yourdomain.example.com
```

Of course you'll need to replace Your Name and you@yourdomain.example.com with your actual name and email address.

Aliases

You might well benefit from some aliases to common commands.

For example, you might well want to be able to shorten git checkout to git co. Or you may want to alias git diff --color-words (which gives a nicely formatted output of the diff) to git wdiff

The following git config --global commands:

```
git config --global alias.ci "commit -a"
git config --global alias.co checkout
git config --global alias.st "status -a"
git config --global alias.stat "status -a"
git config --global alias.br branch
git config --global alias.wdiff "diff --color-words"
```

will create an alias section in your .gitconfig file with contents like this:

```
[alias]
  ci = commit -a
  co = checkout
  st = status -a
  stat = status -a
  br = branch
  wdiff = diff --color-words
```

Editor

You may also want to make sure that your editor of choice is used

```
git config --global core.editor vim
```

Merging

To enforce summaries when doing merges (~/.gitconfig file again):

```
[merge]
  log = true
```

Or from the command line:

```
git config --global merge.log true
```

Fancy log output

This is a very nice alias to get a fancy log output; it should go in the alias section of your .gitconfig file:

```
lg = log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr)
→%C(bold blue)[%an]%Creset' --abbrev-commit --date=relative
```

You use the alias with:

```
git lg
```

and it gives graph / text output something like this (but with color!):

```
* 6d8e1ee - (HEAD, origin/my-fancy-feature, my-fancy-feature) NF - a fancy file (45 minutes
→ago) [Matthew Brett]
* d304a73 - (origin/placeholder, placeholder) Merge pull request #48 from hhuuggoo/master
→(2 weeks ago) [Jonathan Terhorst]
|\
| * 4aff2a8 - fixed bug 35, and added a test in test_bugfixes (2 weeks ago) [Hugo]
|/
* a7ff2e5 - Added notes on discussion/proposal made during Data Array Summit. (2 weeks ago)
→[Corran Webster]
* 68f6752 - Initial implimentation of AxisIndexer - uses 'index_by' which needs to be changed
→to a call on an Axes object - this is all very sketchy right now. (2 weeks ago) [Corr
* 376adbd - Merge pull request #46 from terhorst/master (2 weeks ago) [Jonathan Terhorst]
|\
| * b605216 - updated joshu example to current api (3 weeks ago) [Jonathan Terhorst]
| * 2e991e8 - add testing for outer ufunc (3 weeks ago) [Jonathan Terhorst]
| * 7beda5a - prevent axis from throwing an exception if testing equality with non-axis object (3
→weeks ago) [Jonathan Terhorst]
| * 65af65e - convert unit testing code to assertions (3 weeks ago) [Jonathan Terhorst]
| * 956fbab - Merge remote-tracking branch 'upstream/master' (3 weeks ago) [Jonathan
→Terhorst]
| |\
| |/\
```

Thanks to Yury V. Zaytsev for posting it.

83.5.4 Development workflow

You already have your own forked copy of the [Matplotlib](#) repository, by following [Making your own copy \(fork\) of Matplotlib](#). You have Set up your fork. You have configured git by following

Configure git. Now you are ready for some real work.

Workflow summary

In what follows we'll refer to the upstream Matplotlib master branch, as "trunk".

- Don't use your master branch for anything. Consider deleting it.
- When you are starting a new set of changes, fetch any changes from trunk, and start a new feature branch from that.
- Make a new branch for each separable set of changes — "one task, one branch" ([ipython git workflow](#)).
- Name your branch for the purpose of the changes - e.g. `bugfix-for-issue-14` or `refactor-database-code`.
- If you can possibly avoid it, avoid merging trunk or any other branches into your feature branch while you are working.
- If you do find yourself merging from trunk, consider Rebasing on trunk
- Ask on the [Matplotlib mailing list](#) if you get stuck.
- Ask for code review!

This way of working helps to keep work well organized, with readable history. This in turn makes it easier for project maintainers (that might be you) to see what you've done, and why you did it.

See [linux git workflow](#) and [ipython git workflow](#) for some explanation.

Consider deleting your master branch

It may sound strange, but deleting your own master branch can help reduce confusion about which branch you are on. See [deleting master on github](#) for details.

Update the mirror of trunk

First make sure you have done [Linking your repository to the upstream repo](#).

From time to time you should fetch the upstream (trunk) changes from github:

```
git fetch upstream
```

This will pull down any commits you don't have, and set the remote branches to point to the right commit. For example, 'trunk' is the branch referred to by (remote/branchname) upstream/master - and if there have been commits since you last checked, upstream/master will change after you do the fetch.

Make a new feature branch

When you are ready to make some changes to the code, you should start a new branch. Branches that are for a collection of related edits are often called 'feature branches'.

Making an new branch for each set of related changes will make it easier for someone reviewing your branch to see what you are doing.

Choose an informative name for the branch to remind yourself and the rest of us what the changes in the branch are for. For example `add-ability-to-fly`, or `buxfix-for-issue-42`.

```
# Update the mirror of trunk
git fetch upstream
# Make new feature branch starting at current trunk
git branch my-new-feature upstream/master
git checkout my-new-feature
```

Generally, you will want to keep your feature branches on your public [github](#) fork of [Matplotlib](#). To do this, you [git push](#) this new branch up to your github repo. Generally (if you followed the instructions in these pages, and by default), git will have a link to your github repo, called `origin`. You push up to your own repo on github with:

```
git push origin my-new-feature
```

In git `>= 1.7` you can ensure that the link is correctly set by using the `--set-upstream` option:

```
git push --set-upstream origin my-new-feature
```

From now on git will know that `my-new-feature` is related to the `my-new-feature` branch in the github repo.

The editing workflow

Overview

```
# hack hack
git add my_new_file
git commit -am 'NF - some message'
git push
```

In more detail

1. Make some changes
2. See which files have changed with `git status` (see [git status](#)). You'll see a listing like this one:

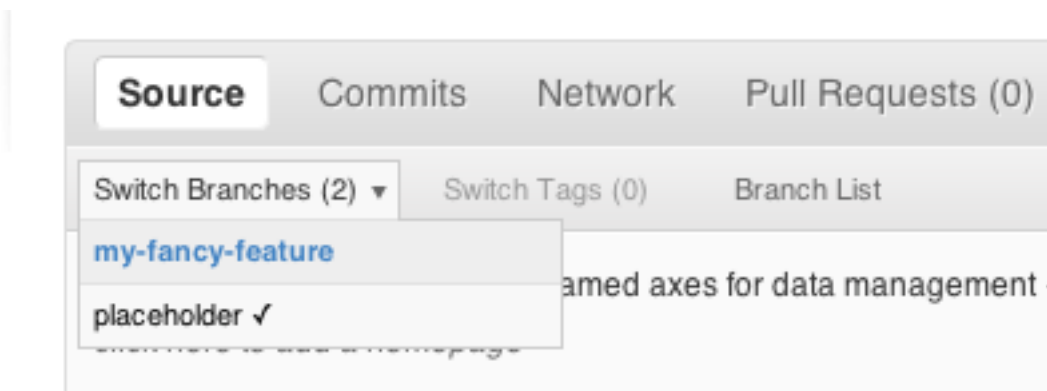
```
# On branch ny-new-feature
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   README
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   INSTALL
no changes added to commit (use "git add" and/or "git commit -a")
```

3. Check what the actual changes are with `git diff` ([git diff](#)).
4. Add any new files to version control `git add new_file_name` (see [git add](#)).
5. To commit all modified files into the local copy of your repo,, do `git commit -am 'A commit message'`. Note the `-am` options to commit. The `m` flag just signals that you're going to type a message on the command line. The `a` flag — you can just take on faith — or see [why the -a flag?](#) — and the helpful use-case description in the [tangled working copy problem](#). The [git commit](#) manual page might also be useful.
6. To push the changes up to your forked repo on github, do a `git push` (see [git push](#)).

Ask for your changes to be reviewed or merged

When you are ready to ask for someone to review your code and consider a merge:

1. Go to the URL of your forked repo, say <https://github.com/your-user-name/matplotlib>.
2. Use the 'Switch Branches' dropdown menu near the top left of the page to select the branch with your changes:



3. Click on the 'Pull request' button:



Enter a title for the set of changes, and some explanation of what you've done. Say if there is anything you'd like particular attention for - like a complicated change or some code you are not happy with.

If you don't think your request is ready to be merged, just say so in your pull request message. This is still a good way of getting some preliminary code review.

Some other things you might want to do

Delete a branch on github

```
git checkout master
# delete branch locally
git branch -D my-unwanted-branch
# delete branch on github
git push origin :my-unwanted-branch
```

Note the colon : before my-unwanted-branch. See also: <https://help.github.com/articles/pushing-to-a-remote/#deleting-a-remote-branch-or-tag>

Several people sharing a single repository

If you want to work on some stuff with other people, where you are all committing into the same repository, or even the same branch, then just share it via github.

First fork Matplotlib into your account, as from Making your own copy (fork) of Matplotlib.

Then, go to your forked repository github page, say <https://github.com/your-user-name/matplotlib>

Click on the 'Admin' button, and add anyone else to the repo as a collaborator:



Now all those people can do:


```
git clone https://github.com/your-user-name/matplotlib.git
```

Remember that links starting with https or git@ are read-write, and that git@ uses the ssh protocol; links starting with git:// are read-only.

Your collaborators can then commit directly into that repo with the usual:

```
git commit -am 'ENH - much better code'
git push origin master # pushes directly into your repo
```

Explore your repository

To see a graphical representation of the repository branches and commits:

```
gitk --all
```

To see a linear list of commits for this branch:

```
git log
```

You can also look at the [network graph visualizer](#) for your github repo.

Finally the Fancy log output lg alias will give you a reasonable text-based graph of the repository.

Rebasing on trunk

Let's say you thought of some work you'd like to do. You Update the mirror of trunk and Make a new feature branch called cool-feature. At this stage trunk is at some commit, let's call it E. Now you make some new commits on your cool-feature branch, let's call them A, B, C. Maybe your changes take a while, or you come back to them after a while. In the meantime, trunk has progressed from commit E to commit (say) G:

```
  A---B---C cool-feature
 /
D---E---F---G trunk
```

At this stage you consider merging trunk into your feature branch, and you remember that this here page sternly advises you not to do that, because the history will get messy. Most of the time you can just ask for a review, and not worry that trunk has got a little ahead. But sometimes, the changes in trunk might affect your changes, and you need to harmonize them. In this situation you may prefer to do a rebase.

rebase takes your changes (A, B, C) and replays them as if they had been made to the current state of trunk. In other words, in this case, it takes the changes represented by A, B, C and replays them on top of G. After the rebase, your history will look like this:

```

      A'--B'--C' cool-feature
      /
D---E---F---G trunk

```

See [rebase without tears](#) for more detail.

To do a rebase on trunk:

```

# Update the mirror of trunk
git fetch upstream
# go to the feature branch
git checkout cool-feature
# make a backup in case you mess up
git branch tmp cool-feature
# rebase cool-feature onto trunk
git rebase --onto upstream/master upstream/master cool-feature

```

In this situation, where you are already on branch cool-feature, the last command can be written more succinctly as:

```
git rebase upstream/master
```

When all looks good you can delete your backup branch:

```
git branch -D tmp
```

If it doesn't look good you may need to have a look at [Recovering from mess-ups](#).

If you have made changes to files that have also changed in trunk, this may generate merge conflicts that you need to resolve - see the [git rebase](#) man page for some instructions at the end of the "Description" section. There is some related help on merging in the git user manual - see [resolving a merge](#).

Recovering from mess-ups

Sometimes, you mess up merges or rebases. Luckily, in git it is relatively straightforward to recover from such mistakes.

If you mess up during a rebase:

```
git rebase --abort
```

If you notice you messed up after the rebase:

```

# reset branch back to the saved point
git reset --hard tmp

```

If you forgot to make a backup branch:

```
# look at the reflog of the branch
git reflog show cool-feature

8630830 cool-feature@{0}: commit: BUG: io: close file handles immediately
278dd2a cool-feature@{1}: rebase finished: refs/heads/my-feature-branch onto 11ee694744f2552d
26aa21a cool-feature@{2}: commit: BUG: lib: make seek_gzip_factory not leak gzip obj
...

# reset the branch to where it was before the botched rebase
git reset --hard cool-feature@{2}
```

Rewriting commit history

Note: Do this only for your own feature branches.

There's an embarrassing typo in a commit you made? Or perhaps the you made several false starts you would like the posterity not to see.

This can be done via interactive rebasing.

Suppose that the commit history looks like this:

```
git log --oneline
eadc391 Fix some remaining bugs
a815645 Modify it so that it works
2dec1ac Fix a few bugs + disable
13d7934 First implementation
6ad92e5 * masked is now an instance of a new object, MaskedConstant
29001ed Add pre-nep for a copule of structured_array_extensions.
...
```

and 6ad92e5 is the last commit in the cool-feature branch. Suppose we want to make the following changes:

- Rewrite the commit message for 13d7934 to something more sensible.
- Combine the commits 2dec1ac, a815645, eadc391 into a single one.

We do as follows:

```
# make a backup of the current state
git branch tmp HEAD
# interactive rebase
git rebase -i 6ad92e5
```

This will open an editor with the following text in it:

```

pick 13d7934 First implementation
pick 2dec1ac Fix a few bugs + disable
pick a815645 Modify it so that it works
pick eadc391 Fix some remaining bugs

# Rebase 6ad92e5..eadc391 onto 6ad92e5
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#

```

To achieve what we want, we will make the following changes to it:

```

r 13d7934 First implementation
pick 2dec1ac Fix a few bugs + disable
f a815645 Modify it so that it works
f eadc391 Fix some remaining bugs

```

This means that (i) we want to edit the commit message for 13d7934, and (ii) collapse the last three commits into one. Now we save and quit the editor.

Git will then immediately bring up an editor for editing the commit message. After revising it, we get the output:

```

[detached HEAD 721fc64] FOO: First implementation
 2 files changed, 199 insertions(+), 66 deletions(-)
[detached HEAD 0f22701] Fix a few bugs + disable
 1 files changed, 79 insertions(+), 61 deletions(-)
Successfully rebased and updated refs/heads/my-feature-branch.

```

and the history looks now like this:

```

0f22701 Fix a few bugs + disable
721fc64 ENH: Sophisticated feature
6ad92e5 * masked is now an instance of a new object, MaskedConstant

```

If it went wrong, recovery is again possible as explained above.

83.5.5 Maintainer workflow

This page is for maintainers — those of us who merge our own or other peoples' changes into the upstream repository.

Being as how you're a maintainer, you are completely on top of the basic stuff in Development workflow.

The instructions in Linking your repository to the upstream repo add a remote that has read-only access to the upstream repo. Being a maintainer, you've got read-write access.

It's good to have your upstream remote have a scary name, to remind you that it's a read-write remote:

```
git remote add upstream-rw git@github.com:matplotlib/matplotlib.git
git fetch upstream-rw
```

Integrating changes

Let's say you have some changes that need to go into trunk (upstream-rw/master).

The changes are in some branch that you are currently on. For example, you are looking at someone's changes like this:

```
git remote add someone git://github.com/someone/matplotlib.git
git fetch someone
git branch cool-feature --track someone/cool-feature
git checkout cool-feature
```

So now you are on the branch with the changes to be incorporated upstream. The rest of this section assumes you are on this branch.

A few commits

If there are only a few commits, consider rebasing to upstream:

```
# Fetch upstream changes
git fetch upstream-rw
# rebase
git rebase upstream-rw/master
```

Remember that, if you do a rebase, and push that, you'll have to close any github pull requests manually, because github will not be able to detect the changes have already been merged.

A long series of commits

If there are a longer series of related commits, consider a merge instead:

```
git fetch upstream-rw
git merge --no-ff upstream-rw/master
```

The merge will be detected by github, and should close any related pull requests automatically.

Note the `--no-ff` above. This forces git to make a merge commit, rather than doing a fast-forward, so that these set of commits branch off trunk then rejoin the main history with a merge, rather than appearing to have been made directly on top of trunk.

Check the history

Now, in either case, you should check that the history is sensible and you have the right commits:

```
git log --oneline --graph
git log -p upstream-rw/master..
```

The first line above just shows the history in a compact way, with a text representation of the history graph. The second line shows the log of commits excluding those that can be reached from trunk (`upstream-rw/master`), and including those that can be reached from current HEAD (implied with the `..` at the end). So, it shows the commits unique to this branch compared to trunk. The `-p` option shows the diff for these commits in patch form.

Push to trunk

```
git push upstream-rw my-new-feature:master
```

This pushes the `my-new-feature` branch in this repository to the `master` branch in the `upstream-rw` repository.

83.6 git resources

83.6.1 Tutorials and summaries

- [github help](#) has an excellent series of how-to guides.
- The [pro git book](#) is a good in-depth book on git.
- A [git cheat sheet](#) is a page giving summaries of common commands.
- The [git user manual](#)
- The [git tutorial](#)
- The [git community book](#)
- [git ready](#) — a nice series of tutorials
- [git magic](#) — extended introduction with intermediate detail

- The [git parable](#) is an easy read explaining the concepts behind git.
- [git foundation](#) expands on the [git parable](#).
- Fernando Perez' [git page](#) — [Fernando's git page](#) — many links and tips
- A good but technical page on [git concepts](#)
- [git svn crash course](#): git for those of us used to [subversion](#)

83.6.2 Advanced git workflow

There are many ways of working with git; here are some posts on the rules of thumb that other projects have come up with:

- Linus Torvalds on [git management](#)
- Linus Torvalds on [linux git workflow](#) . Summary; use the git tools to make the history of your edits as clean as possible; merge from upstream edits as little as possible in branches where you are doing active development.

83.6.3 Manual pages online

You can get these on your own machine with (e.g) `git help push` or (same thing) `git push --help`, but, for convenience, here are the online manual pages for some common commands:

- [git add](#)
- [git branch](#)
- [git checkout](#)
- [git clone](#)
- [git commit](#)
- [git config](#)
- [git diff](#)
- [git log](#)
- [git pull](#)
- [git push](#)
- [git remote](#)
- [git status](#)

83.7 Two and three dots in difference specs

Thanks to Yarik Halchenko for this explanation.

Imagine a series of commits A, B, C, D... Imagine that there are two branches, topic and master. You branched topic off master when master was at commit 'E'. The graph of the commits looks like this:

```
A---B---C topic
/
D---E---F---G master
```

Then:

```
git diff master..topic
```

will output the difference from G to C (i.e. with effects of F and G), while:

```
git diff master...topic
```

would output just differences in the topic branch (i.e. only A, B, and C).

REVIEWERS GUIDELINE

84.1 Pull request checklist

84.1.1 Branch selection

In general target the master branch for all new features and bug-fixes. PRs may target maintenance or doc branches on a case-by-case basis.

84.1.2 Documentation

- Every new feature should be documented. If it's a new module, don't forget to add a new rst file to the API docs.
- Each high-level plotting function should have a small example in the Example section of the docstring. This should be as simple as possible to demonstrate the method. More complex examples should go in the examples section of the documentation.
- Build the docs and make sure all formatting warnings are addressed.
- See Writing documentation for our documentation style guide.
- If your change is a major new feature, add an entry to doc/users/whats_new.rst.
- If you change the API in a backward-incompatible way, please document it in doc/api/api_changes.rst.

84.2 PR Review guidelines

- Be patient and [kind](#) with contributors.
- If you have commit rights, then you are trusted to use them. Please help review and merge PRs!
- Documentation and examples may be merged by the first reviewer. Use the threshold "is this better than it was?" as the review criteria.

- For code changes (anything in `src` or `lib`) at least two developers (those with commit rights) should review all pull requests. If you are the first to review a PR and approve of the changes use the github `'approve review'` tool to mark it as such. If you are a subsequent reviewer please approve the review and if you think no more review is needed, merge the PR.

Ensure that all API changes are documented in `doc/api/api_changes` and significant new features have an entry in `doc/user/whats_new`.

- Make sure the Travis, Appveyor, circle, and codecov tests are passing before merging.
 - Whenever a pull request is created or updated, Travis and Appveyor automatically runs the test suite on all versions of Python supported by Matplotlib. The `tox` support in Matplotlib may be useful for testing locally.
- Do not self merge, except for 'small' patches to un-break the CI or when another reviewer explicitly allows it (ex, "Approve modulo CI passing, may self merge when green")
- Squashing is case-by-case. The balance is between burden on the contributor, keeping a relatively clean history, and keeping a history usable for bisecting. The only time we are really strict about it is to eliminate binary files (ex multiple test image re-generations) and to remove upstream merges.
- Do not let perfect be the enemy of the good, particularly for documentation or example PRs. If you find yourself making many small suggestions, either open a PR against the original branch, push changes to the contributor branch, or merge the PR and then open a new PR against upstream.
- If you push to a contributor branch leave a comment explaining what you did, ex "I took the liberty of pushing a small clean-up PR to your branch, thanks for your work.". If you are going to make substantial changes to the code or intent of the PR please check with the contributor first.

84.3 Branches and Backports

The current active branches are

`master` This will be Matplotlib 3.0. Supports Python 3.5+.

`v2.2.x` Maintenance branch for Matplotlib 2.2 LTS. Supports Python 2.7, 3.4+

`v2.2.N-doc` Documentation for the current release. On a patch release, this will be replaced by a properly named branch for the new release.

We always will backport to 2.2.x

- critical bug fixes (segfault, failure to import, things that the user can not work around)
- fixes for regressions against 2.0 or 2.1

Everything else (regressions against 1.x versions, bugs/api inconsistencies the user can work around in their code) are on a case-by-case basis, should be low-risk, and need someone to advocate for and shepherd through the backport.

The only changes to be backported to 2.2.N-doc are changes to doc, examples, or tutorials. Any changes to lib or src should not be backported to this branch.

84.3.1 Automated backports

We use meeseeksdev bot to automatically backport merges to the correct maintenance branch base on the milestone. To work properly the milestone must be set before merging. If you have commit rights, the bot can also be manually triggered after a merge by leaving a message @meeseeksdev backport to BRANCH on the PR. If there are conflicts meeseekdevs will inform you that the backport needs to be done manually.

The target branch is configured by putting on-merge: backport to TARGETBRANCH in the milestone description on it's own line.

If the bot is not working as expected, please report issues to [Meeseeksdev](#).

84.3.2 Manual backports

When doing backports please copy the form used by meeseekdev, Backport PR #XXXX: TITLE OF PR. If you need to manually resolve conflicts make note of them and how you resolved them in the commit message.

We do a backport from master to v2.2.x assuming:

- matplotlib is a read-only remote branch of the matplotlib/matplotlib repo

The TARGET_SHA is the hash of the merge commit you would like to backport. This can be read off of the github PR page (in the UI with the merge notification) or through the git CLI tools.

Assuming that you already have a local branch v2.2.x (if not, then git checkout -b v2.2.x), and that your remote pointing to <https://github.com/matplotlib/matplotlib> is called upstream:

```
git fetch upstream
git checkout v2.2.x # or include -b if you don't already have this.
git reset --hard upstream/v2.2.x
git cherry-pick -m 1 TARGET_SHA
# resolve conflicts and commit if required
```

Files with conflicts can be listed by git status, and will have to be fixed by hand (search on >>>>). Once the conflict is resolved, you will have to re-add the file(s) to the branch and then continue the cherry pick:

```
git add lib/matplotlib/conflicted_file.py
git add lib/matplotlib/conflicted_file2.py
git cherry-pick --continue
```

Use your discretion to push directly to upstream or to open a PR; be sure to push or PR against the v2.2.x upstream branch, not master!

RELEASE GUIDE

A guide for developers who are doing a matplotlib release.

85.1 All Releases

85.1.1 Testing

We use [travis-ci](#) for continuous integration. When preparing for a release, the final tagged commit should be tested locally before it is uploaded:

```
python tests.py --processes=8 --process-timeout=300
```

In addition the following two tests should be run and manually inspected:

```
python unit/memleak_hawaii3.py
pushd examples/tests/
python backend_driver.py
popd
```

85.1.2 GitHub Stats

We automatically extract GitHub issue, PRs, and authors from GitHub via the API:

```
python tools/github_stats.py --since-tag $TAG --project 'matplotlib/matplotlib' --links > doc/
↪users/github_stats.rst
```

Review and commit changes. Some issue/PR titles may not be valid rst (the most common issue is * which is interpreted as unclosed markup).

85.1.3 Check Docs

Before tagging, update the what's new listing in doc/users/whats_new.rst by merging all files in doc/users/next_whats_new/ coherently. Also, temporarily comment out the include and toctree glob; re-instate these after a release. Finally, make sure that the docs build cleanly

```
make -C doc O=-n$(nproc) html latexpdf
```

After the docs are built, check that all of the links, internal and external, are still valid. We use linkchecker for this, which has not been ported to python3 yet. You will need to create a python2 environment with `requests==2.9.0` and linkchecker

```
conda create -p /tmp/lnkchk python=2 requests==2.9.0
source activate /tmp/lnkchk
pip install linkchecker
pushd doc/build/html
linkchecker index.html --check-extern
```

Address any issues which may arise. The internal links are checked on travis, this should only flag failed external links.

85.1.4 Create release commit and tag

To create the tag, first create an empty commit with a very terse set of the release notes in the commit message

```
git commit --allow-empty
```

and then create a signed, annotated tag with the same text in the body message

```
git tag -a -s v2.0.0
```

which will prompt you for your gpg key password and an annotation. For pre releases it is important to follow [PEP 440](#) so that the build artifacts will sort correctly in pypi. Finally, push the tag to GitHub

```
git push -t DANGER v2.0.0
```

Congratulations, the scariest part is done!

To prevent issues with any down-stream builders which download the tarball from GitHub it is important to move all branches away from the commit with the tag¹:

```
git commit --allow-empty
git push DANGER master
```

¹ The tarball that is provided by GitHub is produced using [git archive](#). We use [versioneer](#) which uses a format string in `lib/matplotlib/_version.py` to have git insert a list of references to exported commit (see `.gitattributes` for the configuration). This string is then used by versioneer to produce the correct version, based on the git tag, when users install from the tarball. However, if there is a branch pointed at the tagged commit, then the branch name will also be included in the tarball. When the branch eventually moves, anyone how checked the hash of the tarball before the branch moved will have an incorrect hash.

To generate the file that GitHub does use

```
git archive v2.0.0 -o matplotlib-2.0.0.tar.gz --prefix=matplotlib-2.0.0/
```

If this is a final release, also create a 'doc' branch (this is not done for pre-releases):

```
git branch v2.0.0-doc
git push DANGER v2.0.0-doc
```

and if this is a major or minor release, also create a bug-fix branch (a micro release will be cut off of this branch):

```
git branch v2.0.x
git push DANGER v2.0.x
```

85.1.5 Release Management / DOI

Via the GitHub UI (chase down link), turn the newly pushed tag into a release. If this is a pre-release remember to mark it as such.

For final releases also get a DOI from [zenodo](#) and edit `doc/_templates/citing.html` with DOI link and commit to the VER-doc branch and push to GitHub

```
git checkout v2.0.0-doc
emacs doc/_templates/citing.html
git push DANGER v2.0.0-doc:v2.0.0-doc
```

85.1.6 Building binaries

We distribute mac, windows, and many linux wheels as well as a source tarball via pypi. Before uploading anything, contact the various builders. Mac and manylinux wheels are built on travis . You need to edit the `.travis.yml` file and push to master of [the build project](#).

Update the master branch (for pre-releases the devel branch) of the [conda-forge feedstock](#) via pull request.

If this is a final release the following downstream packagers should be contacted:

- Debian
- Fedora
- Arch
- Gentoo
- Macports
- Homebrew
- Christoph Gohlke
- Continuum
- Enthought

This can be done ahead of collecting all of the binaries and uploading to pypi.

85.1.7 make distribution and upload to pypi / SF

Once you have collected all of the wheels, generate the tarball

```
git checkout v2.0.0
git clean -xfd
python setup.py sdist
```

and copy all of the wheels into dist directory. You should use twine to upload all of the files to pypi

```
twine upload -s dist/matplotlib*tar.gz
twine upload dist/*whl
```

Congratulations, you have now done the second scariest part!

Additionally, for a final release, upload all of the files to sourceforge.

85.1.8 Build and Deploy Documentation

To build the documentation you must have the tagged version installed, but build the docs from the ver-doc branch. An easy way to arrange this is:

```
pip install matplotlib
pip install -r doc-requirements.txt
git checkout v2.0.0-doc
git clean -xfd
cd doc
make O=-n$(nproc) html latexpdf
```

which will build both the html and pdf version of the documentation.

The built documentation exists in the matplotlib.github.com repository. Pushing changes to master automatically updates the website.

The documentation is organized by version. At the root of the tree is always the documentation for the latest stable release. Under that, there are directories containing the documentation for older versions. The documentation for current master are built on travis and push to the [devdocs](https://matplotlib.org/devdocs) repository. These are available at matplotlib.org/devdocs.

Assuming you have this repository checked out in the same directory as matplotlib

```
cd ../matplotlib.github.com
mkdir 2.0.0
rsync -a ../matplotlib/doc/build/html/* 2.0.0
cp ../matplotlib/doc/build/latex/Matplotlib.pdf 2.0.0
```

which will copy the built docs over. If this is a final release, also replace the top-level docs

```
rsync -a 2.0.0/* ./
```

You will need to manually edit versions.html to show the last 3 tagged versions. Now commit and push everything to GitHub

```
git add *  
git commit -a -m 'Updating docs for v2.0.0'  
git push DANGER master
```

Congratulations you have now done the third scariest part!

It typically takes about 5-10 minutes for GitHub to process the push and update the live web page (remember to clear your browser cache).

85.1.9 Announcing

The final step is to announce the release to the world. A short version of the release notes along with acknowledgments should be sent to

- matplotlib-user@python.org
- matplotlib-devel@python.org
- matplotlib-announce@python.org

For final releases announcements should also be sent to the numpy/scipy/jupyter mailing lists and python-announce.

In addition, announcements should be made on social networks (twitter, g+, FB). For major release, NumFOCUS should be contacted for inclusion in their newsletter and maybe to have something posted on their blog.

MINIMUM VERSION OF DEPENDENCIES POLICY

86.1 Python and numpy

- support minor versions of Python initially released in the previous 3 years
- support minor versions of numpy initially released in the previous 3 years or oldest that supports the minimum python version (which ever is higher)

We will bump the minimum python and numpy versions as we can every minor and major release, but never on a patch release.

86.2 Python Dependencies

For python dependencies we should support at least

with compiled extensions minor versions released in the last 3 years or the oldest that support our
 minimum python + numpy

without complied extensions minor versions released in the last 2 years or the oldest that supports
 our minimum python.

We will only bump these dependencies as we need new features or the old versions no longer support our minimum numpy or python.

86.3 System and C-dependencies

For system or c-dependencies (libpng, freetype, GUI frameworks, latex, gs, ffmpeg) support as old as practical. These can be difficult to install for end-users and we want to be usable on as many systems as possible. We will bump these on a case-by-case basis.

MATPLOTLIB ENHANCEMENT PROPOSALS

Matplotlib Enhancement Proposals (MEP), inspired by cpython's [PEP's](#) but less formal, are design documents for large or controversial changes to Matplotlib. These documents should provide a discussion of both why and how the changes should be made.

To create a new MEP open a pull request (PR) adding a file based on the template to this the MEP directory. For the initial PR only a rough description is required and it should be merged quickly. Further detailed discussion can happen in follow on PRs.

87.1 MEP Template

- Status
- Branches and Pull requests
- Abstract
- Detailed description
- Implementation
- Backward compatibility
- Alternatives

This MEP template is a guideline of the sections that a MEP should contain. Extra sections may be added if appropriate, and unnecessary sections may be noted as such.

87.1.1 Status

MEPs go through a number of phases in their lifetime:

- Discussion: The MEP is being actively discussed on the mailing list and it is being improved by its author. The mailing list discussion of the MEP should include the MEP number (MEPxxx) in the subject line so they can be easily related to the MEP.
- Progress: Consensus was reached and implementation work has begun.

- Completed: The implementation has been merged into master.
- Superseded: This MEP has been abandoned in favor of another approach.
- Rejected: There are currently no plans to implement the proposal.

87.1.2 Branches and Pull requests

All development branches containing work on this MEP should be linked to from here.

All pull requests submitted relating to this MEP should be linked to from here. (A MEP does not need to be implemented in a single pull request if it makes sense to implement it in discrete phases).

87.1.3 Abstract

The abstract should be a short description of what the MEP will achieve.

87.1.4 Detailed description

This section describes the need for the MEP. It should describe the existing problem that it is trying to solve and why this MEP makes the situation better. It should include examples of how the new functionality would be used and perhaps some use cases.

87.1.5 Implementation

This section lists the major steps required to implement the MEP. Where possible, it should be noted where one step is dependent on another, and which steps may be optionally omitted. Where it makes sense, each step should include a link related pull requests as the implementation progresses.

87.1.6 Backward compatibility

This section describes the ways in which the MEP breaks backward incompatibility.

87.1.7 Alternatives

If there were any alternative solutions to solving the same problem, they should be discussed here, along with a justification for the chosen approach.

87.2 MEP8: PEP8

- Status
- Branches and Pull requests
- Abstract
- Detailed description
- Implementation
- Backward compatibility
- Alternatives

87.2.1 Status

Completed

We are currently enforcing a sub-set of pep8 on new code contributions.

87.2.2 Branches and Pull requests

None so far.

87.2.3 Abstract

The matplotlib codebase predates PEP8, and therefore is less than consistent style-wise in some areas. Bringing the codebase into compliance with PEP8 would go a long way to improving its legibility.

87.2.4 Detailed description

Some files use four space indentation, some use three. Some use different levels in the same file.

For the most part, class/function/variable naming follows PEP8, but it wouldn't hurt to fix where necessary.

87.2.5 Implementation

The implementation should be fairly mechanical: running the pep8 tool over the code and fixing where appropriate.

This should be merged in after the 2.0 release, since the changes will likely make merging any pending pull requests more difficult.

Additionally, and optionally, PEP8 compliance could be tracked by an automated build system.

87.2.6 Backward compatibility

Public names of classes and functions that require change (there shouldn't be many of these) should first be deprecated and then removed in the next release cycle.

87.2.7 Alternatives

PEP8 is a popular standard for Python code style, blessed by the Python core developers, making any alternatives less desirable.

87.3 MEP9: Global interaction manager

- Status
- Branches and Pull requests
- Abstract
- Detailed description
- Implementation
- Current summary of the mixin
- Backward compatibility
- Alternatives

Add a global manager for all user interactivity with artists; make any artist resizable, moveable, highlightable, and selectable as desired by the user.

87.3.1 Status

Discussion

87.3.2 Branches and Pull requests

<https://github.com/dhyams/matplotlib/tree/MEP9>

87.3.3 Abstract

The goal is to be able to interact with matplotlib artists in a very similar way as drawing programs do. When appropriate, the user should be able to move, resize, or select an artist that is already on the canvas. Of course, the script writer is ultimately in control of whether an artist is able to be interacted with, or whether it is static.

This code to do this has already been privately implemented and tested, and would need to be migrated from its current "mixin" implementation, to a bona-fide part of matplotlib.

The end result would be to have four new keywords available to `matplotlib.artist.Artist`: `__moveable__`, `__resizeable__`, `__selectable__`, and `__highlightable__`. Setting any one of these keywords to `True` would activate interactivity for that artist.

In effect, this MEP is a logical extension of event handling in matplotlib; matplotlib already supports "low level" interactions like left mouse presses, a key press, or similar. The MEP extends the support to the logical level, where callbacks are performed on the artists when certain interactive gestures from the user are detected.

87.3.4 Detailed description

This new functionality would be used to allow the end-user to better interact with the graph. Many times, a graph is almost what the user wants, but a small repositioning and/or resizing of components is necessary. Rather than force the user to go back to the script to trial-and-error the location, and simple drag and drop would be appropriate.

Also, this would better support applications that use matplotlib; here, the end-user has no reasonable access or desire to edit the underlying source in order to fine-tune a plot. Here, if matplotlib offered the capability, one could move or resize artists on the canvas to suit their needs. Also, the user should be able to highlight (with a mouse over) an artist, and select it with a double-click, if the application supports that sort of thing. In this MEP, we also want to support the highlighting and selection natively; it is up to application to handle what happens when the artist is selected. A typical handling would be to display a dialog to edit the properties of the artist.

In the future, as well (this is not part of this MEP), matplotlib could offer backend-specific property dialogs for each artist, which are raised on artist selection. This MEP would be a necessary stepping stone for that sort of capability.

There are currently a few interactive capabilities in matplotlib (e.g. `legend.draggable()`), but they tend to be scattered and are not available for all artists. This MEP seeks to unify the interactive interface and make it work for all artists.

The current MEP also includes grab handles for resizing artists, and appropriate boxes drawn when artists are moved or resized.

87.3.5 Implementation

- Add appropriate methods to the "tree" of artists so that the interactivity manager has a consistent interface for the interactivity manager to deal with. The proposed methods to add to the artists, if they are to support interactivity, are:
 - `get_pixel_position_ll(self)`: get the pixel position of the lower left corner of the artist's bounding box
 - `get_pixel_size(self)`: get the size of the artist's bounding box, in pixels
 - `set_pixel_position_and_size(self,x,y,dx,dy)`: set the new size of the artist, such that it fits within the specified bounding box.

- add capability to the backends to 1) provide cursors, since these are needed for visual indication of moving/resizing, and 2) provide a function that gets the current mouse position
- Implement the manager. This has already been done privately (by dhyams) as a mixin, and has been tested quite a bit. The goal would be to move the functionality of the manager into the artists so that it is in matplotlib properly, and not as a "monkey patch" as I currently have it coded.

87.3.6 Current summary of the mixin

(Note that this mixin is for now just private code, but can be added to a branch obviously)

InteractiveArtistMixin:

Mixin class to make any generic object that is drawn on a matplotlib canvas moveable and possibly resizable. The Powerpoint model is followed as closely as possible; not because I'm enamoured with Powerpoint, but because that's what most people understand. An artist can also be selectable, which means that the artist will receive the `on_activated()` callback when double clicked. Finally, an artist can be highlightable, which means that a highlight is drawn on the artist whenever the mouse passes over. Typically, highlightable artists will also be selectable, but that is left up to the user. So, basically there are four attributes that can be set by the user on a per-artist basis:

- highlightable
- selectable
- moveable
- resizable

To be moveable (draggable) or resizable, the object that is the target of the mixin must support the following protocols:

- `get_pixel_position_ll(self)`
- `get_pixel_size(self)`
- `set_pixel_position_and_size(self,x,y,sx,sy)`

Note that nonresizable objects are free to ignore the `sx` and `sy` parameters. To be highlightable, the object that is the target of the mixin must also support the following protocol:

- `get_highlight(self)`

Which returns a list of artists that will be used to draw the highlight.

If the object that is the target of the mixin is not an matplotlib artist, the following protocols must also be implemented. Doing so is usually fairly trivial, as there has to be an artist somewhere that is being drawn. Typically your object would just route these calls to that artist.

- `get_figure(self)`
- `get_axes(self)`
- `contains(self,event)`

- `set_animated(self,flag)`
- `draw(self,renderer)`
- `get_visible(self)`

The following notifications are called on the artist, and the artist can optionally implement these.

- `on_select_begin(self)`
- `on_select_end(self)`
- `on_drag_begin(self)`
- `on_drag_end(self)`
- `on_activated(self)`
- `on_highlight(self)`
- `on_right_click(self,event)`
- `on_left_click(self,event)`
- `on_middle_click(self,event)`
- `on_context_click(self,event)`
- `on_key_up(self,event)`
- `on_key_down(self,event)`

The following notifications are called on the canvas, if no interactive artist handles the event:

- `on_press(self,event)`
- `on_left_click(self,event)`
- `on_middle_click(self,event)`
- `on_right_click(self,event)`
- `on_context_click(self,event)`
- `on_key_up(self,event)`
- `on_key_down(self,event)`

The following functions, if present, can be used to modify the behavior of the interactive object:

- `press_filter(self,event)` # determines if the object wants to have the press event routed to it
- `handle_unpicked_cursor()` # can be used by the object to set a cursor as the cursor passes over the object when it is unpicked.

Supports multiple canvases, maintaining a drag lock, motion notifier, and a global "enabled" flag per canvas. Supports fixed aspect ratio resizings by holding the shift key during the resize.

Known problems:

- Zorder is not obeyed during the selection/drag operations. Because of the blit technique used, I do not believe this can be fixed. The only way I can think of is to search for all artists that have a zorder greater than me, set them all to animated, and then redraw them all on top during each drag refresh. This might be very slow; need to try.
- the mixin only works for wx backends because of two things: 1) the cursors are hardcoded, and 2) there is a call to `wx.GetMousePosition()`. Both of these shortcomings are reasonably fixed by having each backend supply these things.

87.3.7 Backward compatibility

No problems with backward compatibility, although once this is in place, it would be appropriate to obsolete some of the existing interactive functions (like `legend.draggable()`)

87.3.8 Alternatives

None that I know of.

87.4 MEP10: Docstring consistency

- Status
- Branches and Pull requests
- Abstract
- Detailed description
 - Numpy docstring format
 - Cross references
 - Overriding signatures
 - Linking rather than duplicating
 - autosummary extension
 - Examples linking to relevant documentation
 - Documentation using `help()` vs a browser
- Implementation
- Backward compatibility
- Alternatives

87.4.1 Status

Progress

This is still an on-going effort

87.4.2 Branches and Pull requests

87.4.3 Abstract

matplotlib has a great deal of inconsistency between docstrings. This not only makes the docs harder to read, but it is harder on contributors, because they don't know which specifications to follow. There should be a clear docstring convention that is followed consistently.

The organization of the API documentation is difficult to follow. Some pages, such as pyplot and axes, are enormous and hard to browse. There should instead be short summary tables that link to detailed documentation. In addition, some of the docstrings themselves are quite long and contain redundant information.

Building the documentation takes a long time and uses a make.py script rather than a Makefile.

87.4.4 Detailed description

There are number of new tools and conventions available since matplotlib started using Sphinx that make life easier. The following is a list of proposed changes to docstrings, most of which involve these new features.

Numpy docstring format

Numpy docstring format: This format divides the docstring into clear sections, each having different parsing rules that make the docstring easy to read both as raw text and as HTML. We could consider alternatives, or invent our own, but this is a strong choice, as it's well used and understood in the Numpy/Scipy community.

Cross references

Most of the docstrings in matplotlib use explicit "roles" when linking to other items, for example: `:func:`myfunction``. As of Sphinx 0.4, there is a "default_role" that can be set to "obj", which will polymorphically link to a Python object of any type. This allows one to write ``myfunction`` instead. This makes docstrings much easier to read and edit as raw text. Additionally, Sphinx allows for setting a current module, so links like ``~matplotlib.axes.Axes.set_xlim`` could be written as ``~axes.Axes.set_xlim``.

Overriding signatures

Many methods in matplotlib use the `*args` and `**kwargs` syntax to dynamically handle the keyword arguments that are accepted by the function, or to delegate on to another function. This, however, is often not useful as a signature in the documentation. For this reason, many matplotlib methods include something like:

```
def annotate(self, *args, **kwargs):
    """
    Create an annotation: a piece of text referring to a data
    point.

    Call signature::

        annotate(s, xy, xytext=None, xycoords='data',
                textcoords='data', arrowprops=None, **kwargs)
    """
```

This can't be parsed by Sphinx, and is rather verbose in raw text. As of Sphinx 1.1, if the `autodoc_docstring_signature` config value is set to `True`, Sphinx will extract a replacement signature from the first line of the docstring, allowing this:

```
def annotate(self, *args, **kwargs):
    """
    annotate(s, xy, xytext=None, xycoords='data',
            textcoords='data', arrowprops=None, **kwargs)

    Create an annotation: a piece of text referring to a data
    point.
    """
```

The explicit signature will replace the actual Python one in the generated documentation.

Linking rather than duplicating

Many of the docstrings include long lists of accepted keywords by interpolating things into the docstring at load time. This makes the docstrings very long. Also, since these tables are the same across many docstrings, it inserts a lot of redundant information in the docs – particularly a problem in the printed version.

These tables should be moved to docstrings on functions whose only purpose is for help. The docstrings that refer to these tables should link to them, rather than including them verbatim.

autosummary extension

The Sphinx autosummary extension should be used to generate summary tables, that link to separate pages of documentation. Some classes that have many methods (e.g. `Axes.axes`) should

be documented with one method per page, whereas smaller classes should have all of their methods together.

Examples linking to relevant documentation

The examples, while helpful at illustrating how to use a feature, do not link back to the relevant docstrings. This could be addressed by adding module-level docstrings to the examples, and then including that docstring in the parsed content on the example page. These docstrings could easily include references to any other part of the documentation.

Documentation using `help()` vs a browser

Using Sphinx markup in the source allows for good-looking docs in your browser, but the markup also makes the raw text returned using `help()` look terrible. One of the aims of improving the docstrings should be to make both methods of accessing the docs look good.

87.4.5 Implementation

1. The `numpydoc` extensions should be turned on for `matplotlib`. There is an important question as to whether these should be included in the `matplotlib` source tree, or used as a dependency. Installing Numpy is not sufficient to get the `numpydoc` extensions – it’s a separate install procedure. In any case, to the extent that they require customization for our needs, we should endeavor to submit those changes upstream and not fork them.
2. Manually go through all of the docstrings and update them to the new format and conventions. Updating the cross references (from ``:func:`myfunc`` to ``func``) may be able to be semi-automated. This is a lot of busy work, and perhaps this labor should be divided on a per-module basis so no single developer is over-burdened by it.
3. Reorganize the API docs using `autosummary` and `sphinx-autogen`. This should hopefully have minimal impact on the narrative documentation.
4. Modify the example page generator (`gen_rst.py`) so that it extracts the module docstring from the example and includes it in a non-literal part of the example page.
5. Use `sphinx-quickstart` to generate a new-style Sphinx Makefile. The following features in the current `make.py` will have to be addressed in some other way:
 - Copying of some static content
 - Specifying a "small" build (only low-resolution PNG files for examples)

Steps 1, 2, and 3 are interdependent. 4 and 5 may be done independently, though 5 has some dependency on 3.

87.4.6 Backward compatibility

As this mainly involves docstrings, there should be minimal impact on backward compatibility.

87.4.7 Alternatives

None yet discussed.

87.5 MEP11: Third-party dependencies

- Status
- Branches and Pull requests
- Abstract
- Detailed description
 - Current behavior
 - Desired behavior
- Implementation
- Backward compatibility
- Alternatives

This MEP attempts to improve the way in which third-party dependencies in matplotlib are handled.

87.5.1 Status

Completed – needs to be merged

87.5.2 Branches and Pull requests

#1157: Use automatic dependency resolution

#1290: Debundle pyparsing

#1261: Update six to 1.2

87.5.3 Abstract

One of the goals of matplotlib has been to keep it as easy to install as possible. To that end, some third-party dependencies are included in the source tree and, under certain circumstances, installed alongside matplotlib. This MEP aims to resolve some problems with that approach, bring some consistency, while continuing to make installation convenient.

At the time that was initially done, `setuptools`, `easy_install` and `PyPI` were not mature enough to be relied on. However, at present, we should be able to safely leverage the "modern" versions of those tools, distribute and `pip`.

While `matplotlib` has dependencies on both Python libraries and C/C++ libraries, this MEP addresses only the Python libraries so as to not confuse the issue. C libraries represent a larger and mostly orthogonal set of problems.

87.5.4 Detailed description

`matplotlib` depends on the following third-party Python libraries:

- `Numpy`
- `dateutil` (pure Python)
- `pytz` (pure Python)
- `six` – required by `dateutil` (pure Python)
- `pyparsing` (pure Python)
- `PIL` (optional)
- GUI frameworks: `pygtk`, `gobject`, `tkinter`, `PySide`, `PyQt4`, `wx` (all optional, but one is required for an interactive GUI)

Current behavior

When installing from source, a git checkout or `pip`:

- `setup.py` attempts to import `numpy`. If this fails, the installation fails.
- For each of `dateutil`, `pytz` and `six`, `setup.py` attempts to import them (from the top-level namespace). If that fails, `matplotlib` installs its local copy of the library into the top-level namespace.
- `pyparsing` is always installed inside of the `matplotlib` namespace.

This behavior is most surprising when used with `pip`, because no `pip` dependency resolution is performed, even though it is likely to work for all of these packages.

The fact that `pyparsing` is installed in the `matplotlib` namespace has reportedly (#1290) confused some users into thinking it is a `matplotlib`-related module and import it from there rather than the top-level.

When installing using the Windows installer, `dateutil`, `pytz` and `six` are installed at the top-level always, potentially overwriting already installed copies of those libraries.

TODO: Describe behavior with the OS-X installer.

When installing using a package manager (Debian, RedHat, MacPorts etc.), this behavior actually does the right thing, and there are no special patches in the `matplotlib` packages to deal with the

fact that we handle `dateutil`, `pytz` and `six` in this way. However, care should be taken that whatever approach we move to continues to work in that context.

Maintaining these packages in the matplotlib tree and making sure they are up-to-date is a maintenance burden. Advanced new features that may require a third-party pure Python library have a higher barrier to inclusion because of this burden.

Desired behavior

Third-party dependencies are downloaded and installed from their canonical locations by leveraging `pip`, `distribute` and `PyPI`.

`dateutil`, `pytz`, and `pyparsing` should be made into optional dependencies – though obviously some features would fail if they aren't installed. This will allow the user to decide whether they want to bother installing a particular feature.

87.5.5 Implementation

For installing from source, and assuming the user has all of the C-level compilers and dependencies, this can be accomplished fairly easily using `distribute` and following the instructions [here](#). The only anticipated change to the matplotlib library code will be to import `pyparsing` from the top-level namespace rather than from within matplotlib. Note that `distribute` will also allow us to remove the direct dependency on `six`, since it is, strictly speaking, only a direct dependency of `dateutil`.

For binary installations, there are a number of alternatives (here ordered from best/hardest to worst/easiest):

1. The `distutils` `wininst` installer allows a post-install script to run. It might be possible to get this script to run `pip` to install the other dependencies. (See [this thread](#) for someone who has trod that ground before).
2. Continue to ship `dateutil`, `pytz`, `six` and `pyparsing` in our installer, but use the post-install-script to install them only if they can not already be found.
3. Move all of these packages inside a (new) `matplotlib.extern` namespace so it is clear for outside users that these are external packages. Add some conditional imports in the core matplotlib codebase so `dateutil` (at the top-level) is tried first, and failing that `matplotlib.extern.dateutil` is used.

2 and 3 are undesirable as they still require maintaining copies of these packages in our tree – and this is exacerbated by the fact that they are used less – only in the binary installers. None of these 3 approaches address Numpy, which will still have to be manually installed using an installer.

TODO: How does this relate to the Mac OS-X installer?

87.5.6 Backward compatibility

At present, matplotlib can be installed from source on a machine without the third party dependencies and without an internet connection. After this change, an internet connection (and a working

PyPI) will be required to install matplotlib for the first time. (Subsequent matplotlib updates or development work will run without accessing the network).

87.5.7 Alternatives

Distributing binary eggs doesn't feel like a usable solution. That requires getting easy_install installed first, and Windows users generally prefer the well known exe or msi installer that works out of the box.

87.6 MEP12: Improve Gallery and Examples

- Status
- Branches and Pull requests
- Abstract
- Detailed description
- Implementation
 - Gallery sections
 - Clean up guidelines
 - * Additional suggestions
- Backward compatibility
- Alternatives
 - Tags

87.6.1 Status

Progress

Initial changes added in 1.3. Conversion of the gallery is on-going. 29 September 2015 - The last pylab_examples where pylab is imported has been converted over to use matplotlib pyplot and numpy.

87.6.2 Branches and Pull requests

#1623, #1924, #2181

PR #2474 <<https://github.com/matplotlib/matplotlib/pull/2474>>_ demonstrates a single example being cleaned up and moved to the appropriate section.

87.6.3 Abstract

Reorganizing the matplotlib plot gallery would greatly simplify navigation of the gallery. In addition, examples should be cleaned-up and simplified for clarity.

87.6.4 Detailed description

The matplotlib gallery was recently set up to split examples up into sections. As discussed in that PR¹, the current example sections (`api`, `pylab_examples`) aren't terribly useful to users: New sections in the gallery would help users find relevant examples.

These sections would also guide a cleanup of the examples: Initially, all the current examples would remain and be listed under their current directories. Over time, these examples could be cleaned up and moved into one of the new sections.

This process allows users to easily identify examples that need to be cleaned up; i.e. anything in the `api` and `pylab_examples` directories.

87.6.5 Implementation

1. Create new gallery sections. [Done]
2. Clean up examples and move them to the new gallery sections (over the course of many PRs and with the help of many users/developers). [In progress]

Gallery sections

The naming of sections is critical and will guide the clean-up effort. The current sections are:

- Lines, bars, and markers (more-or-less 1D data)
- Shapes and collections
- Statistical plots
- Images, contours, and fields
- Pie and polar charts: Round things
- Color
- Text, labels, and annotations
- Ticks and spines
- Subplots, axes, and figures
- Specialty plots (e.g., sankey, radar, tornado)
- Showcase (plots with tweaks to make them publication-quality)
- separate sections for toolboxes (already exists: `'mplot3d'`, `'axes_grid'`, `'units'`, `'widgets'`)

¹ <https://github.com/matplotlib/matplotlib/pull/714>

These names are certainly up for debate. As these sections grow, we should reevaluate them and split them up as necessary.

Clean up guidelines

The current examples in the `api` and `pylab_examples` sections of the gallery would remain in those directories until they are cleaned up. After clean-up, they would be moved to one of the new gallery sections described above. "Clean-up" should involve:

- `sphinx-gallery docstrings`: a title and a description of the example formatted as follows, at the top of the example:

```
"""
=====
Colormaps alter your perception
=====

Here I plot the function

.. math:: f(x, y) = \sin(x) + \cos(y)

with different colormaps. Look at how colormaps alter your perception!
"""
```

- PEP8 clean-ups (running `flake8`, or a similar checker, is highly recommended)
- Commented-out code should be removed.
- Replace uses of `pylab` interface with `pyplot` (+ `numpy`, etc.). See [c25ef1e](#)
- Remove shebang line, e.g.:

```
#!/usr/bin/env python
```

- Use consistent imports. In particular:

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

Avoid importing specific functions from these modules (e.g. `from numpy import sin`)

- Each example should focus on a specific feature (excluding showcase examples, which will show more "polished" plots). Tweaking unrelated to that feature should be removed. See [f7b2217](#), [e57b5fc](#), and [1458aa8](#)

Use of `pylab` should be demonstrated/discussed on a dedicated help page instead of the gallery examples.

Note: When moving an existing example, you should search for references to that example. For example, the API documentation for `axes.py` and `pyplot.py` may use these examples to generate plots. Use your favorite search tool (e.g., `grep`, `ack`, [grin](#), [pss](#)) to search the `matplotlib` package. See [2dc9a46](#) and [aa6b410](#)

Additional suggestions

- Provide links (both ways) between examples and API docs for the methods/objects used. (issue [#2222](#))
- Use `plt.subplots` (note trailing "s") in preference over `plt.subplot`.
- Rename the example to clarify its purpose. For example, the most basic demo of `imshow` might be `imshow_demo.py`, and one demonstrating different interpolation settings would be `imshow_demo_interpolation.py` (not `imshow_demo2.py`).
- Split up examples that try to do too much. See [5099675](#) and [fc2ab07](#)
- Delete examples that don't show anything new.
- Some examples exercise esoteric features for unit testing. These tweaks should be moved out of the gallery to an example in the unit directory located in the root directory of the package.
- Add plot titles to clarify intent of the example. See [bd2b13c](#)

87.6.6 Backward compatibility

The website for each Matplotlib version is readily accessible, so users who want to refer to old examples can still do so.

87.6.7 Alternatives

Tags

Tagging examples will also help users search the example gallery. Although tags would be a big win for users with specific goals, the plot gallery will remain the entry point to these examples, and sections could really help users navigate the gallery. Thus, tags are complementary to this reorganization.

87.7 MEP13: Use properties for Artists

- Status
- Branches and Pull requests
- Abstract
- Detailed description
- Implementation
- Backward compatibility

- Examples
 - `axes.Axes.set_axis_off/set_axis_on`
 - `axes.Axes.get_xlim/set_xlim` and `get_autoscalex_on/set_autoscalex_on`
 - `axes.Axes.get_title/set_title`
 - `axes.Axes.get_xticklabels/set_xticklabels`
- Alternatives

87.7.1 Status

- Discussion

87.7.2 Branches and Pull requests

None

87.7.3 Abstract

Wrap all of the matplotlib getter and setter methods with python [properties](#), allowing them to be read and written like class attributes.

87.7.4 Detailed description

Currently matplotlib uses getter and setter functions (usually prefixed with `get_` and `set_`, respectively) for reading and writing data related to classes. However, since 2.6 python supports properties, which allow such setter and getter functions to be accessed as though they were attributes. This proposal would implement all existing setter and getter methods as properties.

87.7.5 Implementation

1. All existing getter and setter methods will need to have two aliases, one with the `get_` or `set_` prefix and one without. Getter methods that currently lack prefixes should be recording in a text file.
2. Classes should be reorganized so setter and getter methods are sequential in the code, with getter methods first.
3. Getter and setter methods the provide additional optional arguments should have those arguments accessible in another manner, either as additional getter or setter methods or attributes of other classes. If those classes are not accessible, getters for them should be added.

4. Property decorators will be added to the setter and getter methods without the prefix. Those with the prefix will be marked as deprecated.
5. Docstrings will need to be rewritten so the getter with the prefix has the current docstring and the getter without the prefix has a generic docstring appropriate for an attribute.
6. Automatic alias generation will need to be modified so it will also create aliases for the properties.
7. All instances of getter and setter method calls will need to be changed to attribute access.
8. All setter and getter aliases with prefixes will be removed

The following steps can be done simultaneously: 1, 2, and 3; 4 and 5; 6 and 7.

Only the following steps must be done in the same release: 4, 5, and 6. All other changes can be done in separate releases. 8 should be done several major releases after everything else.

87.7.6 Backward compatibility

All existing getter methods that do not have a prefix (such as `get_`) will need to be changed from function calls to attribute access. In most cases this will only require removing the parenthesis.

setter and getter methods that have additional optional arguments will need to have those arguments implemented in another way, either as a separate property in the same class or as attributes or properties of another class.

Cases where the setter returns a value will need to be changed to using the setter followed by the getter.

Cases where there are `set_ATTR_on()` and `set_ATTR_off()` methods will be changed to `ATTR_on` properties.

87.7.7 Examples

`axes.Axes.set_axis_off/set_axis_on`

Current implementation:

```
axes.Axes.set_axis_off()
axes.Axes.set_axis_on()
```

New implementation:

```
True = axes.Axes.axis_on
False = axes.Axes.axis_on
axes.Axes.axis_on = True
axes.Axes.axis_on = False
```

`axes.Axes.get_xlim/set_xlim` and `get_autoscalex_on/set_autoscalex_on`

Current implementation:

```
[left, right] = axes.Axes.get_xlim()
auto = axes.Axes.get_autoscalex_on()

[left, right] = axes.Axes.set_xlim(left=left, right=right, emit=emit, auto=auto)
[left, right] = axes.Axes.set_xlim(left=left, right=None, emit=emit, auto=auto)
[left, right] = axes.Axes.set_xlim(left=None, right=right, emit=emit, auto=auto)
[left, right] = axes.Axes.set_xlim(left=left, emit=emit, auto=auto)
[left, right] = axes.Axes.set_xlim(right=right, emit=emit, auto=auto)

axes.Axes.set_autoscalex_on(auto)
```

New implementation:

```
[left, right] = axes.Axes.axes_xlim
auto = axes.Axes.autoscalex_on

axes.Axes.axes_xlim = [left, right]
axes.Axes.axes_xlim = [left, None]
axes.Axes.axes_xlim = [None, right]
axes.Axes.axes_xlim[0] = left
axes.Axes.axes_xlim[1] = right

axes.Axes.autoscalex_on = auto

axes.Axes.emit_xlim = emit
```

`axes.Axes.get_title/set_title`

Current implementation:

```
string = axes.Axes.get_title()
axes.Axes.set_title(string, fontdict=fontdict, **kwargs)
```

New implementation:

```
string = axes.Axes.title
string = axes.Axes.title_text.text

text.Text = axes.Axes.title_text
text.Text.<attribute> = attribute
text.Text.fontdict = fontdict

axes.Axes.title = string
```

(continues on next page)

(continued from previous page)

```
axes.Axes.title = text.Text
axes.Axes.title__text = string
axes.Axes.title__text = text.Text
```

`axes.Axes.get_xticklabels/set_xticklabels`

Current implementation:

```
[text.Text] = axes.Axes.get_xticklabels()
[text.Text] = axes.Axes.get_xticklabels(minor=False)
[text.Text] = axes.Axes.get_xticklabels(minor=True)
[text.Text] = axes.Axes.([string], fontdict=None, **kwargs)
[text.Text] = axes.Axes.([string], fontdict=None, minor=False, **kwargs)
[text.Text] = axes.Axes.([string], fontdict=None, minor=True, **kwargs)
```

New implementation:

```
[text.Text] = axes.Axes.xticklabels
[text.Text] = axes.Axes.xminorticklabels
axes.Axes.xticklabels = [string]
axes.Axes.xminorticklabels = [string]
axes.Axes.xticklabels = [text.Text]
axes.Axes.xminorticklabels = [text.Text]
```

87.7.8 Alternatives

Instead of using decorators, it is also possible to use the property function. This would change the procedure so that all getter methods that lack a prefix will need to be renamed or removed. This makes handling docstrings more difficult and harder to read.

It is not necessary to deprecate the setter and getter methods, but leaving them in will complicate the code.

This could also serve as an opportunity to rewrite or even remove automatic alias generation.

Another alternate proposal:

Convert `set_xlim`, `set_xlabel`, `set_title`, etc. to `xlim`, `xlabel`, `title`,... to make the transition from plt functions to axes methods significantly simpler. These would still be methods, not properties, but it's still a great usability enhancement while retaining the interface.

87.8 MEP14: Text handling

- Status
- Branches and Pull requests
- Abstract
- Detailed description
- Implementation
- Backward compatibility
- Alternatives

87.8.1 Status

- Discussion

87.8.2 Branches and Pull requests

Issue #253 demonstrates a bug where using the bounding box rather than the advance width of text results in misaligned text. This is a minor point in the grand scheme of things, but it should be addressed as part of this MEP.

87.8.3 Abstract

By reorganizing how text is handled, this MEP aims to:

- improve support for Unicode and non-ltr languages
- improve text layout (especially multi-line text)
- allow support for more fonts, especially non-Apple-format TrueType fonts and OpenType fonts.
- make the font configuration easier and more transparent

87.8.4 Detailed description

Text layout

At present, matplotlib has two different ways to render text: "built-in" (based on FreeType and our own Python code), and "usetex" (based on calling out to a TeX installation). Adjunct to the "built-in" renderer there is also the Python-based "mathtext" system for rendering mathematical equations using a subset of the TeX language without having a TeX installation available. Support for these two engines is strewn about many source files, including every backend, where one finds clauses like

```
if rcParams['text.usetex']: # do one thing else: # do another
```

Adding a third text rendering approach (more on that later) would require editing all of these places as well, and therefore doesn't scale.

Instead, this MEP proposes adding a concept of "text engines", where the user could select one of many different approaches for rendering text. The implementations of each of these would be localized to their own set of modules, and not have little pieces around the whole source tree.

Why add more text rendering engines? The "built-in" text rendering has a number of shortcomings.

- It only handles right-to-left languages, and doesn't handle many special features of Unicode, such as combining diacriticals.
- The multiline support is imperfect and only supports manual line-breaking – it can not break up a paragraph into lines of a certain length.
- It also does not handle inline formatting changes in order to support something like Markdown, reStructuredText or HTML. (Though rich-text formatting is contemplated in this MEP, since we want to make sure this design allows it, the specifics of a rich-text formatting implementation is outside of the scope of this MEP.)

Supporting these things is difficult, and is the "full-time job" of a number of other projects:

- [pango/harfbuzz](#)
- [QtTextLayout](#)
- [Microsoft DirectWrite](#)
- [Apple Core Text](#)

Of the above options, it should be noted that harfbuzz is designed from the start as a cross platform option with minimal dependencies, so therefore is a good candidate for a single option to support.

Additionally, for supporting rich text, we could consider using [WebKit](#), and possibly whether than represents a good single cross-platform option. Again, however, rich text formatting is outside of the scope of this project.

Rather than trying to reinvent the wheel and add these features to matplotlib's "built-in" text renderer, we should provide a way to leverage these projects to get more powerful text layout. The "built-in" renderer will still need to exist for reasons of ease of installation, but its feature set will be more limited compared to the others. [TODO: This MEP should clearly decide what those limited features are, and fix any bugs to bring the implementation into a state of working correctly in all cases that we want it to work. I know @leejjoon has some thoughts on this.]

Font selection

Going from an abstract description of a font to a file on disk is the task of the font selection algorithm – it turns out to be much more complicated than it seems at first.

The "built-in" and "usetex" renderers have very different ways of handling font selection, given their different technologies. TeX requires the installation of TeX-specific font packages, for example, and can not use TrueType fonts directly. Unfortunately, despite the different semantics for font selection, the same set of font properties are used for each. This is true of both the FontProperties

class and the font-related rcParams (which basically share the same code underneath). Instead, we should define a core set of font selection parameters that will work across all text engines, and have engine-specific configuration to allow the user to do engine-specific things when required. For example, it is possible to directly select a font by name in the "built-in" using `font.family`, but the same is not possible with "usetex". It may be possible to make it easier to use TrueType fonts by using XeTeX, but users will still want to use the traditional metafonts through TeX font packages. So the issue still stands that different text engines will need engine-specific configuration, and it should be more obvious to the user which configuration will work across text engines and which are engine-specific.

Note that even excluding "usetex", there are different ways to find fonts. The default is to use the font list cache in `font_manager.py` which matches fonts using our own algorithm based on the [CSS font matching algorithm](#). It doesn't always do the same thing as the native font selection algorithms on Linux (`fontconfig`), Mac and Windows, and it doesn't always find all of the fonts on the system that the OS would normally pick up. However, it is cross-platform, and always finds the fonts that ship with matplotlib. The Cairo and MacOSX backends (and presumably a future HTML5-based backend) currently bypass this mechanism and use the OS-native ones. The same is true when not embedding fonts in SVG, PS or PDF files and opening them in a third-party viewer. A downside there is that (at least with Cairo, need to confirm with MacOSX) they don't always find the fonts we ship with matplotlib. (It may be possible to add the fonts to their search path, though, or we may need to find a way to install our fonts to a location the OS expects to find them).

There are also special modes in the PS and PDF to only use the core fonts that are always available to those formats. There, the font lookup mechanism must only match against those fonts. It is unclear whether the OS-native font lookup systems can handle this case.

There is also experimental support for using `fontconfig` for font selection in matplotlib, turned off by default. `fontconfig` is the native font selection algorithm on Linux, but is also cross platform and works well on the other platforms (though obviously is an additional dependency there).

Many of the text layout libraries proposed above (pango, QtTextLayout, DirectWrite and CoreText etc.) insist on using the font selection library from their own ecosystem.

All of the above seems to suggest that we should move away from our self-written font selection algorithm and use the native APIs where possible. That's what Cairo and MacOSX backends already want to use, and it will be a requirement of any complex text layout library. On Linux, we already have the bones of a `fontconfig` implementation (which could also be accessed through pango). On Windows and Mac we may need to write custom wrappers. The nice thing is that the API for font lookup is relatively small, and essentially consist of "given a dictionary of font properties, give me a matching font file".

Font subsetting

Font subsetting is currently handled using `ttconv`. `ttconv` was a standalone commandline utility for converting TrueType fonts to subsetted Type 3 fonts (among other features) written in 1995, which matplotlib (well, I) forked in order to make it work as a library. It only handles Apple-style TrueType fonts, not ones with the Microsoft (or other vendor) encodings. It doesn't handle OpenType fonts at all. This means that even though the STIX fonts come as .otf files, we have to convert them to .ttf files to ship them with matplotlib. The Linux packagers hate this – they'd rather just depend on the upstream STIX fonts. `ttconv` has also been shown to have a few bugs that have been difficult to fix over time.

Instead, we should be able to use FreeType to get the font outlines and write our own code (probably in Python) to output subsetting fonts (Type 3 on PS and PDF and SVGFonts or paths on SVG). FreeType, as a popular and well-maintained project, handles a wide variety of fonts in the wild. This would remove a lot of custom C code, and remove some code duplication between backends.

Note that subsetting fonts this way, while the easiest route, does lose the hinting in the font, so we will need to continue, as we do now, provide a way to embed the entire font in the file where possible.

Alternative font subsetting options include using the subsetting built-in to Cairo (not clear if it can be used without the rest of Cairo), or using fontforge (which is a heavy and not terribly cross-platform dependency).

FreeType wrappers

Our FreeType wrapper could really use a reworking. It defines its own image buffer class (when a Numpy array would be easier). While FreeType can handle a huge diversity of font files, there are limitations to our wrapper that make it much harder to support non-Apple-vendor TrueType files, and certain features of OpenType files. (See #2088 for a terrible result of this, just to support the fonts that ship with Windows 7 and 8). I think a fresh rewrite of this wrapper would go a long way.

Text anchoring and alignment and rotation

The handling of baselines was changed in 1.3.0 such that the backends are now given the location of the baseline of the text, not the bottom of the text. This is probably the correct behavior, and the MEP refactoring should also follow this convention.

In order to support alignment on multi-line text, it should be the responsibility of the (proposed) text engine to handle text alignment. For a given chunk of text, each engine calculates a bounding box for that text and the offset of the anchor point within that box. Therefore, if the va of a block was "top", the anchor point would be at the top of the box.

Rotating of text should always be around the anchor point. I'm not sure that lines up with current behavior in matplotlib, but it seems like the sanest/least surprising choice. [This could be revisited once we have something working]. Rotation of text should not be handled by the text engine – that should be handled by a layer between the text engine and the rendering backend so it can be handled in a uniform way. [I don't see any advantage to rotation being handled by the text engines individually...]

There are other problems with text alignment and anchoring that should be resolved as part of this work. [TODO: enumerate these].

Other minor problems to fix

The mathtext code has backend-specific code – it should instead provide its output as just another text engine. However, it's still desirable to have mathtext layout inserted as part of a larger layout performed by another text engine, so it should be possible to do this. It's an open question whether embedding the text layout of an arbitrary text engine in another should be possible.

The text mode is currently set by a global rcParam ("text.usetex") so it's either all on or all off. We should continue to have a global rcParam to choose the text engine ("text.layout_engine"), but it should under the hood be an overridable property on the Text object, so the same figure can combine the results of multiple text layout engines if necessary.

87.8.5 Implementation

A concept of a "text engine" will be introduced. Each text engine will implement a number of abstract classes. The `TextFont` interface will represent text for a given set of font properties. It isn't necessarily limited to a single font file – if the layout engine supports rich text, it may handle a number of font files in a family. Given a `TextFont` instance, the user can get a `TextLayout` instance, which represents the layout for a given string of text in a given font. From a `TextLayout`, an iterator over `TextSpans` is returned so the engine can output raw editable text using as few spans as possible. If the engine would rather get individual characters, they can be obtained from the `TextSpan` instance:

```
class TextFont(TextFontBase):
    def __init__(self, font_properties):
        """
        Create a new object for rendering text using the given font properties.
        """
        pass

    def get_layout(self, s, ha, va):
        """
        Get the TextLayout for the given string in the given font and
        the horizontal (left, center, right) and verticalalignment (top,
        center, baseline, bottom)
        """
        pass

class TextLayout(TextLayoutBase):
    def get_metrics(self):
        """
        Return the bounding box of the layout, anchored at (0, 0).
        """
        pass

    def get_spans(self):
        """
        Returns an iterator over the spans of different in the layout.
        This is useful for backends that want to editable raw text as
        individual lines. For rich text where the font may change,
        each span of different font type will have its own span.
        """
        pass

    def get_image(self):
        """
        Returns a rasterized image of the text. Useful for raster backends,
        like Agg.
```

(continues on next page)

(continued from previous page)

```

    In all likelihood, this will be overridden in the backend, as it can
    be created from get_layout(), but certain backends may want to
    override it if their library provides it (as freetype does).
    """

    pass

    def get_rectangles(self):
        """
        Returns an iterator over the filled black rectangles in the layout.
        Used by TeX and mathtext for drawing, for example, fraction lines.
        """

        pass

    def get_path(self):
        """
        Returns a single Path object of the entire laid out text.

        [Not strictly necessary, but might be useful for textpath
        functionality]
        """

        pass

class TextSpan(TextSpanBase):
    x, y      # Position of the span -- relative to the text layout as a whole
              # where (0, 0) is the anchor. y is the baseline of the span.
    fontfile  # The font file to use for the span
    text      # The text content of the span

    def get_path(self):
        pass # See TextLayout.get_path

    def get_chars(self):
        """
        Returns an iterator over the characters in the span.
        """

        pass

class TextChar(TextCharBase):
    x, y      # Position of the character -- relative to the text layout as
              # a whole, where (0, 0) is the anchor. y is in the baseline
              # of the character.
    codepoint # The unicode code point of the character -- only for informational
              # purposes, since the mapping of codepoint to glyph_id may have been
              # handled in a complex way by the layout engine. This is an int
              # to avoid problems on narrow Unicode builds.

```

(continues on next page)

(continued from previous page)

```

glyph_id # The index of the glyph within the font
fontfile # The font file to use for the char

def get_path(self):
    """
    Get the path for the character.
    """
pass

```

Graphic backends that want to output subset of fonts would likely build up a file-global dictionary of characters where the keys are (fontname, glyph_id) and the values are the paths so that only one copy of the path for each character will be stored in the file.

Special casing: The "usetex" functionality currently is able to get Postscript directly from TeX to insert directly in a Postscript file, but for other backends, parses a DVI file and generates something more abstract. For a case like this, TextLayout would implement get_spans for most backends, but add get_ps for the Postscript backend, which would look for the presence of this method and use it if available, or fall back to get_spans. This kind of special casing may also be necessary, for example, when the graphics backend and text engine belong to the same ecosystem, e.g. Cairo and Pango, or MacOSX and CoreText.

There are three main pieces to the implementation:

- 1) Rewriting the freetype wrapper, and removing ttconv.
 - a) Once (1) is done, as a proof of concept, we can move to the upstream STIX .otf fonts
 - b) Add support for web fonts loaded from a remote URL. (Enabled by using freetype for font subsetting).
- 2) Refactoring the existing "builtin" and "usetex" code into separate text engines and to follow the API outlined above.
- 3) Implementing support for advanced text layout libraries.

(1) and (2) are fairly independent, though having (1) done first will allow (2) to be simpler. (3) is dependent on (1) and (2), but even if it doesn't get done (or is postponed), completing (1) and (2) will make it easier to move forward with improving the "builtin" text engine.

87.8.6 Backward compatibility

The layout of text with respect to its anchor and rotation will change in hopefully small, but improved, ways. The layout of multiline text will be much better, as it will respect horizontal alignment. The layout of bidirectional text or other advanced Unicode features will now work inherently, which may break some things if users are currently using their own workarounds.

Fonts will be selected differently. Hacks that used to sort of work between the "builtin" and "usetex" text rendering engines may no longer work. Fonts found by the OS that weren't previously found by matplotlib may be selected.

87.8.7 Alternatives

TBD

87.9 MEP15 - Fix axis autoscaling when limits are specified for one axis only

- Status
- Branches and Pull requests
- Abstract
- Detailed description
- Implementation
- Backward compatibility
- Alternatives

87.9.1 Status

Discussion

87.9.2 Branches and Pull requests

None so far.

87.9.3 Abstract

When one axis of a 2-dimensional plot is overridden via `xlim` or `ylim`, automatic scaling of the remaining axis should be based on the data that falls within the specified limits of the first axis.

87.9.4 Detailed description

When axis limits for a 2-D plot are specified for one axis only (via `xlim` or `ylim`), matplotlib currently does not currently rescale the other axis. The result is that the displayed curves or symbols may be compressed into a tiny portion of the available area, so that the final plot conveys much less information than it would with appropriate axis scaling.

The proposed change of behavior would make matplotlib choose the scale for the remaining axis using only the data that falls within the limits for the axis where limits were specified.

87.9.5 Implementation

I don't know enough about the internals of matplotlib to be able to suggest an implementation.

87.9.6 Backward compatibility

From the standpoint of software interfaces, there would be no break in backward compatibility. Some outputs would be different, but if the user truly desires the previous behavior, he/she can achieve this by overriding the axis scaling for both axes.

87.9.7 Alternatives

The only alternative that I can see is to maintain the status quo.

87.10 MEP19: Continuous Integration

87.10.1 Status

Completed

87.10.2 Branches and Pull requests

87.10.3 Abstract

matplotlib could benefit from better and more reliable continuous integration, both for testing and building installers and documentation.

87.10.4 Detailed description

Current state-of-the-art

Testing

matplotlib currently uses Travis-CI for automated tests. While Travis-CI should be praised for how much it does as a free service, it has a number of shortcomings:

- It often fails due to network timeouts when installing dependencies.
- It often fails for inexplicable reasons.
- build or test products can only be saved from build off of branches on the main repo, not pull requests, so it is often difficult to "post mortem" analyse what went wrong. This is particularly frustrating when the failure can not be subsequently reproduced locally.
- It is not extremely fast. matplotlib's cpu and memory requirements for testing are much higher than the average Python project.

- It only tests on Ubuntu Linux, and we have only minimal control over the specifics of the platform. It can be upgraded at any time outside of our control, causing unexpected delays at times that may not be convenient in our release schedule.

On the plus side, Travis-CI's integration with github – automatically testing all pending pull requests – is exceptional.

Builds

There is no centralized effort for automated binary builds for matplotlib. However, the following disparate things are being done [If the authors mentioned here could fill in detail, that would be great!]:

- @sandrotsi: builds Debian packages
- @takluyver: Has automated Ubuntu builds on Launchpad
- @cgohlke: Makes Windows builds (don't know how automated that is)
- @r-owen: Makes OS-X builds (don't know how automated that is)

Documentation

Documentation of master is now built by travis and uploaded to <http://matplotlib.org/devdocs/index.html>

@NelleV, I believe, generates the docs automatically and posts them on the web to chart MEP10 progress.

Peculiarities of matplotlib

matplotlib has complex requirements that make testing and building more taxing than many other Python projects.

- The CPU time to run the tests is quite high. It puts us beyond the free accounts of many CI services (e.g. ShiningPanda)
- It has a large number of dependencies, and testing the full matrix of all combinations is impractical. We need to be clever about what space we test and guarantee to support.

Requirements

This section outlines the requirements that we would like to have.

1. Testing all pull requests by hooking into the Github API, as Travis-CI does
2. Testing on all major platforms: Linux, Mac OS-X, MS Windows (in that order of priority, based on user survey)
3. Retain the last n days worth of build and test products, to aid in post-mortem debugging.
4. Automated nightly binary builds, so that users can test the bleeding edge without installing a complete compilation environment.

5. Automated benchmarking. It would be nice to have a standard benchmark suite (separate from the tests) whose performance could be tracked over time, in different backends and platforms. While this is separate from building and testing, ideally it would run on the same infrastructure.
6. Automated nightly building and publishing of documentation (or as part of testing, to ensure PRs don't introduce documentation bugs). (This would not replace the static documentation for stable releases as a default).
7. The test systems should be manageable by multiple developers, so that no single person becomes a bottleneck. (Travis-CI's design does this well – storing build configuration in the git repository, rather than elsewhere, is a very good design.)
8. Make it easy to test a large but sparse matrix of different versions of matplotlib's dependencies. The matplotlib user survey provides some good data as to where to focus our efforts: <https://docs.google.com/spreadsheet/ccc?key=0AjrPjITMRTwTdHpQS25pcTZIRWdqX0pNckNSU01sMHc#gid=0>
9. Nice to have: A decentralized design so that those with more obscure platforms can publish build results to a central dashboard.

87.10.5 Implementation

This part is yet-to-be-written.

However, ideally, the implementation would be a third-party service, to avoid adding system administration to our already stretched time. As we have some donated funds, this service may be a paid one if it offers significant time-saving advantages over free offerings.

87.10.6 Backward compatibility

Backward compatibility is not a major concern for this MEP. We will replace current tools and procedures with something better and throw out the old.

87.10.7 Alternatives

87.10.8 Hangout Notes

CI Infrastructure

- We like Travis and it will probably remain part of our arsenal in any event. The reliability issues are being looked into.
- Enable Amazon S3 uploads of testing products on Travis. This will help with post-mortem of failures (@mdboom is looking into this now).
- We want Mac coverage. The best bet is probably to push Travis to enable it for our project by paying them for a Pro account (since they don't otherwise allow testing on both Linux and Mac).

- We want Windows coverage. Shining Panda is an option there.
- Investigate finding or building a tool that would collect and synthesize test results from a number of sources and post it to Github using the Github API. This may be of general use to the Scipy community.
- For both Windows and Mac, we should document (or better yet, script) the process of setting up the machine for a build, and how to build binaries and installers. This may require getting information from Russel Owen and Christoph Gohlke. This is a necessary step for doing automated builds, but would also be valuable for a number of other reasons.

The test framework itself

- We should investigate ways to make it take less time
 - Eliminating redundant tests, if possible
 - General performance improvements to matplotlib will help
- We should be covering more things, particularly more backends
- We should have more unit tests, fewer integration tests, if possible

87.11 MEP21: color and cm refactor

- Status
- Branches and Pull requests
- Abstract
- Detailed description
- Implementation
- Backward compatibility
- Alternatives

87.11.1 Status

- Discussion: This MEP has not commenced yet, but here are some ongoing ideas which may become a part of this MEP:

87.11.2 Branches and Pull requests

87.11.3 Abstract

- color
 - tidy up the namespace
 - Define a "Color" class
 - make it easy to convert from one color type to another ``hex -> RGB``, ``RGB -> hex``, ``HSV -> RGB`` etc.
 - improve the construction of a colormap - the dictionary approach is archaic and overly complex (though incredibly powerful)
 - make it possible to interpolate between two or more color types in different modes, especially useful for construction of colormaps in HSV space for instance
- cm
 - rename the module to something more descriptive - mappables?

Overall, there are a lot of improvements that can be made with matplotlib color handling - managing backwards compatibility will be difficult as there are some badly named variables/modules which really shouldn't exist - but a clear path and message for migration should be available, with a large amount of focus on this in the API changes documentation.

87.11.4 Detailed description

87.11.5 Implementation

87.11.6 Backward compatibility

87.11.7 Alternatives

87.12 MEP22: Toolbar rewrite

- Status
- Branches and Pull requests
- Abstract
- Detailed description
- Implementation
 - `ToolBase(object)`
 - `ToolToggleBase(ToolBase)`

- NavigationBase
- ToolbarBase
- Backward compatibility

87.12.1 Status

Progress

87.12.2 Branches and Pull requests

Previous work

- <https://github.com/matplotlib/matplotlib/pull/1849>
- <https://github.com/matplotlib/matplotlib/pull/2557>
- <https://github.com/matplotlib/matplotlib/pull/2465>

Pull Requests:

- Removing the NavigationToolbar classes <https://github.com/matplotlib/matplotlib/pull/2740> CLOSED
- Keeping the NavigationToolbar classes <https://github.com/matplotlib/matplotlib/pull/2759> CLOSED
- Navigation by events: <https://github.com/matplotlib/matplotlib/pull/3652>

87.12.3 Abstract

The main goal of this MEP is to make it easier to modify (add, change, remove) the way the user interacts with the figures.

The user interaction with the figure is deeply integrated within the Canvas and Toolbar. Making extremely difficult to do any modification.

This MEP proposes the separation of this interaction into Toolbar, Navigation and Tools to provide independent access and reconfiguration.

This approach will make easier to create and share tools among users. In the far future, we can even foresee a kind of Marketplace for Tools where the most popular can be added into the main distribution.

87.12.4 Detailed description

The reconfiguration of the Toolbar is complex, most of the time it requires a custom backend.

The creation of custom Tools sometimes interferes with the Toolbar, as example see <https://github.com/matplotlib/matplotlib/issues/2694> also the shortcuts are hardcoded and again not easily modifiable <https://github.com/matplotlib/matplotlib/issues/2699>

The proposed solution is to take the actions out of the Toolbar and the shortcuts out of the Canvas. This actions and shortcuts will be in the form of Tools.

A new class Navigation will be the bridge between the events from the Canvas and Toolbar and redirect them to the appropriate Tool.

At the end the user interaction will be divided into three classes:

- NavigationBase: This class is instantiated for each FigureManager and connect the all user interactions with the Tools
- ToolbarBase: This existing class is relegated only as a GUI access to Tools.
- ToolBase: Is the basic definition of Tools.

87.12.5 Implementation

ToolBase(object)

Tools can have a graphical representation as the SubplotTool or not even be present in the Toolbar as Quit

The ToolBase has the following class attributes for configuration at definition time

- keymap = None: Key(s) to be used to trigger the tool
- description = "": Small description of the tool
- image = None: Image that is used in the toolbar

The following instance attributes are set at instantiation:

- name
- navigation

Methods

- trigger(self, event): This is the main method of the Tool, it is called when the Tool is triggered by: * Toolbar button click * keypress associated with the Tool Keymap * Call to navigation.trigger_tool(name)
- set_figure(self, figure): Set the figure and navigation attributes
- destroy(self, *args): Destroy the Tool graphical interface (if exists)

Available Tools

- ToolQuit
- ToolEnableAllNavigation
- ToolEnableNavigation

- ToolToggleGrid
- ToolToggleFullScreen
- ToolToggleYScale
- ToolToggleXScale
- ToolHome
- ToolBack
- ToolForward
- SaveFigureBase
- ConfigureSubplotsBase

ToolToggleBase(ToolBase)

The ToolToggleBase has the following class attributes for configuration at definition time

- `radio_group = None`: Attribute to group 'radio' like tools (mutually exclusive)
- `cursor = None`: Cursor to use when the tool is active

The Toggleable Tools, can capture keypress, mouse moves, and mouse button press

It defines the following methods

- `enable(self, event)`: Called by ToolToggleBase.trigger method
- `disable(self, event)`: Called when the tool is untoggled
- `toggled` : Property True or False

Available Tools

- ToolZoom
- ToolPan

NavigationBase

Defines the following attributes

- `canvas`:
- `keypresslock`: Lock to know if the canvas `key_press_event` is available and process it
- `messagelock`: Lock to know if the message is available to write

Public methods for User use:

- `nav_connect(self, s, func)`: Connect to navigation for events
- `nav_disconnect(self, cid)`: Disconnect from navigation event
- `message_event(self, message, sender=None)`: Emit a `tool_message_event` event

- `active_toggle(self)`: Property The currently toggled tools or None
- `get_tool_keymap(self, name)`: Return a list of keys that are associated with the tool
- `set_tool_keymap(self, name, *keys)`: Set the keys for the given tool
- `remove_tool(self, name)`: Removes tool from the navigation control.
- `add_tools(self, tools)`: Add multiple tools to Navigation
- `add_tool(self, name, tool, group=None, position=None)`: Add a tool to the Navigation
- `tool_trigger_event(self, name, sender=None, canvasevent=None, data=None)`: Trigger a tool and fire the event
- `tools(self)` Property: Return a dict with available tools with corresponding keymaps, descriptions and objects
- `get_tool(self, name)`: Return the tool object

ToolbarBase

Methods for Backend implementation

- `add_toolitem(self, name, group, position, image, description, toggle)`: Add a toolitem to the toolbar. This method is a callback from `tool_added_event` (emitted by navigation)
- `set_message(self, s)`: Display a message on toolbar or in status bar
- `toggle_toolitem(self, name)`: Toggle the toolitem without firing event.
- `remove_toolitem(self, name)`: Remove a toolitem from the Toolbar

87.12.6 Backward compatibility

For backward compatibility added a 'navigation' key to `rcsetup.validate_toolbar`, that is used for Navigation classes instantiation instead of the `NavigationToolbar` classes

With this parameter, it makes it transparent to anyone using the existing backends.

[@pelson comment: This also gives us an opportunity to avoid needing to implement all of this in the same PR - some backends can potentially exist without the new functionality for a short while (but it must be done at some point).]

87.13 MEP23: Multiple Figures per GUI window

- Status
- Branches and Pull requests
- Abstract

- Detailed description
- Implementation
 - FigureManagerBase
 - new_figure_manager
 - new_figure_manager_given_figure
 - NavigationBase
- Backward compatibility
- Alternatives

87.13.1 Status

Discussion

87.13.2 Branches and Pull requests

Previous work - <https://github.com/matplotlib/matplotlib/pull/2465> To-delete

87.13.3 Abstract

Add the possibility to have multiple figures grouped under the same FigureManager

87.13.4 Detailed description

Under the current structure, every canvas has its own window.

This is and may continue to be the desired method of operation for most use cases.

Sometimes when there are too many figures open at the same time, it is desirable to be able to group these under the same window [see](<https://github.com/matplotlib/matplotlib/issues/2194>).

The proposed solution modifies FigureManagerBase to contain and manage more than one canvas. The settings parameter rcParams["backend.multifigure"] control when the MultiFigure behaviour is desired.

Note

It is important to note, that the proposed solution, assumes that the [MEP22](<https://github.com/matplotlib/matplotlib/wiki/MEP22>) is already in place. This is simply because the actual implementation of the Toolbar makes it pretty hard to switch between canvases.

87.13.5 Implementation

The first implementation will be done in GTK3 using a Notebook as canvas container.

FigureManagerBase

will add the following new methods

- `add_canvas`: To add a canvas to an existing `FigureManager` object
- `remove_canvas`: To remove a canvas from a `FigureManager` object, if it is the last one, it will be destroyed
- `move_canvas`: To move a canvas from one `FigureManager` to another.
- `set_canvas_title`: To change the title associated with a specific canvas container
- `get_canvas_title`: To get the title associated with a specific canvas container
- `get_active_canvas`: To get the canvas that is in the foreground and is subject to the gui events. There is no `set_active_canvas` because the active canvas, is defined when `show` is called on a `Canvas` object.

`new_figure_manager`

To control which `FigureManager` will contain the new figures, an extra optional parameter `figure_manager` will be added, this parameter value will be passed to `new_figure_manager_given_figure`

`new_figure_manager_given_figure`

- If `figuremanager` parameter is give, this `FigureManager` object will be used instead of creating a new one.
- If `rcParams['backend.multifigure'] == True`: The last `FigureManager` object will be used instead of creating a new one.

NavigationBase

Modifies the `NavigationBase` to keep a list of canvases, directing the actions to the active one

87.13.6 Backward compatibility

For the `MultiFigure` properties to be visible, the user has to activate them directly setting `rcParams['backend.multifigure'] = True`

It should be backwards compatible for backends that adhere to the current `FigureManagerBase` structure even if they have not implemented the `MultiFigure` magic yet.

87.13.7 Alternatives

Insted of modifying the `FigureManagerBase` it could be possible to add a parallel class, that handles the cases where `rcParams['backend.multifigure'] = True`. This will warranty that there won't be

any problems with custom made backends, but also makes bigger the code, and more things to maintain.

87.14 MEP24: negative radius in polar plots

- Status
- Branches and Pull requests
- Abstract
- Detailed description
- Implementation
- Related Issues
- Backward compatibility
- Alternatives

87.14.1 Status

Discussion

87.14.2 Branches and Pull requests

None

87.14.3 Abstract

It is clear that polar plots need to be able to gracefully handle negative r values (not by clipping or reflection).

87.14.4 Detailed description

One obvious application that we should support is bB plots (see <https://github.com/matplotlib/matplotlib/issues/1730#issuecomment-40815837>), but this seems more generally useful (for example growth rate as a function of angle). The assumption in the current code (as I understand it) is that the center of the graph is $r=0$, however it would be good to be able to set the center to be at any r (with any value less than the offset clipped).

87.14.5 Implementation

87.14.6 Related Issues

#1730, #1603, #2203, #2133

87.14.7 Backward compatibility

87.14.8 Alternatives

87.15 MEP25: Serialization

- Status
- Branches and Pull requests
- Abstract
- Detailed description
- Examples
- Implementation
- Backward compatibility
- Alternatives

87.15.1 Status

Discussion

87.15.2 Branches and Pull requests

- development branches:
- related pull requests:

87.15.3 Abstract

This MEP aims at adding a serializable Controller objects to act as an Artist managers. Users would then communicate changes to an Artist via a Controller. In this way, functionality of the Controller objects may be added incrementally since each Artist is still responsible for drawing everything. The goal is to create an API that is usable both by graphing libraries requiring high-level descriptions of figures and libraries requiring low-level interpretations.

87.15.4 Detailed description

Matplotlib is a core plotting engine with an API that many users already understand. It's difficult/impossible for other graphing libraries to (1) get a complete figure description, (2) output raw data from the figure object as the user has provided it, (3) understand the semantics of the figure objects without heuristics, and (4) give matplotlib a complete figure description to visualize. In addition, because an Artist has no conception of its own semantics within the figure, it's difficult to interact with them in a natural way.

In this sense, matplotlib will adopt a standard Model-View-Controller (MVC) framework. The Model will be the user defined data, style, and semantics. The Views are the ensemble of each individual Artist, which are responsible for producing the final image based on the model. The Controller will be the Controller object managing its set of Artist objects.

The Controller must be able to export the information that it's carrying about the figure on command, perhaps via a `to_json` method or similar. Because it would be extremely extraneous to duplicate all of the information in the model with the controller, only user-specified information (data + style) are explicitly kept. If a user wants more information (defaults) from the view/model, it should be able to query for it.

- This might be annoying to do, non-specified kwargs are pulled from the `rcParams` object which is in turn created from reading a user specified file and can be dynamically changed at run time. I suppose we could keep a dict of default defaults and compare against that. Not clear how this will interact with the style sheet [\[\[MEP26\]\]](#) - @tacaswell

Additional Notes:

- The raw data does not necessarily need to be a list, ndarray, etc. Rather, it can more abstractly just have a method to yield data when needed.
- Because the Controller will contain extra information that users may not want to keep around, it should not be created by default. You should be able to both (a) instantiate a Controller with a figure and (b) build a figure with a Controller.

Use Cases:

- Export all necessary informat
- Serializing a matplotlib figure, saving it, and being able to rerun later.
- Any other source sending an appropriately formatted representation to matplotlib to open

87.15.5 Examples

Here are some examples of what the controllers should be able to do.

1. Instantiate a matplotlib figure from a serialized representation (e.g., JSON):

```
import json
from matplotlib.controllers import Controller
with open('my_figure') as f:
    o = json.load(f)
```

(continues on next page)

(continued from previous page)

```
c = Controller(o)
fig = c.figure
```

2. Manage artists from the controller (e.g., Line2D):

```
# not really sure how this should look
c.axes[0].lines[0].color = 'b'
# ?
```

3. Export serializable figure representation:

```
o = c.to_json()
# or... we should be able to throw a figure object in there too
o = Controller.to_json(mpl_fig)
```

87.15.6 Implementation

1. Create base Controller objects that are able to manage Artist objects (e.g., Hist)

Comments:

- initialization should happen via unpacking **, so we need a copy of call signature parameter for the Artist we're ultimately trying to control. Unfortunate hard-coded repetition...
- should the additional **kwargs accepted by each Artist be tracked at the Controller
- how does a Controller know which artist belongs where? E.g., do we need to pass axes references?

Progress:

- A simple NB demonstrating some functionality for Line2DController objects: <https://nbviewer.jupyter.org/gist/theengineear/f0aa8d79f64325e767c0>

2. Write in protocols for the Controller to update the model.

Comments:

- how should containers be dealt with? E.g., what happens to old patches when we re-bin a histogram?
- in the link from (1), the old line is completely destroyed and redrawn, what if something is referencing it?

3. Create method by which a json object can be assembled from the Controllers
4. Deal with serializing the unserializable aspects of a figure (e.g., non-affine transforms?)
5. Be able to instantiate from a serialized representation

6. Reimplement the existing pyplot and Axes method, e.g. `pyplot.hist` and `Axes.hist` in terms of the new controller class.

> @theengineer: in #2 above, what do you mean by get updates from each Artist?

^ Yup. The Controller shouldn't need to get updated. This just happens in #3. Delete comments when you see this.

87.15.7 Backward compatibility

- pickling will change
- non-affine transformations will require a defined pickling method

87.15.8 Alternatives

PR #3150 suggested adding semantics by parasitically attaching extra containers to axes objects. This is a more complete solution with what should be a more developed/flexible/powerful framework.

87.16 MEP26: Artist styling

- Status
- Branches and Pull requests
- Abstract
- Detailed description
- Implementation
 - BNF Grammar
 - Syntax
 - * Selectors
 - * GID selector
 - * Attributes and values
 - Parsing
 - Visitor pattern for matplotlib figure
- Backward compatibility
- Alternatives
- Appendix

87.16.1 Status

Rejected

87.16.2 Branches and Pull requests

87.16.3 Abstract

This MEP proposes a new stylesheet implementation to allow more comprehensive and dynamic styling of artists.

The current version of matplotlib (1.4.0) allows stylesheets based on the rcParams syntax to be applied before creation of a plot. The methodology below proposes a new syntax, based on CSS, which would allow styling of individual artists and properties, which can be applied dynamically to existing objects.

This is related to (and makes steps toward) the overall goal of moving to a DOM/tree-like architecture.

87.16.4 Detailed description

Currently, the look and appearance of existing artist objects (figure, axes, Line2D etc...) can only be updated via `set_` and `get_` methods on the artist object, which is quite laborious, especially if no reference to the artist(s) has been stored. The new style sheets introduced in 1.4 allow styling before a plot is created, but do not offer any means to dynamically update plots or distinguish between artists of the same type (i.e. to specify the line color and line style separately for differing Line2D objects).

The initial development should concentrate on allowing styling of artist primitives (those artists that do not contain other artists), and further development could expand the CSS syntax rules and parser to allow more complex styling. See the appendix for a list of primitives.

The new methodology would require development of a number of steps:

- A new stylesheet syntax (likely based on CSS) to allow selection of artists by type, class, id etc...
- A mechanism by which to parse a stylesheet into a tree
- A mechanism by which to translate the parse-tree into something which can be used to update the properties of relevant artists. Ideally this would implement a method by which to traverse the artists in a tree-like structure.
- A mechanism by which to generate a stylesheet from existing artist properties. This would be useful to allow a user to export a stylesheet from an existing figure (where the appearance may have been set using the matplotlib API)...

87.16.5 Implementation

It will be easiest to allow a '3rd party' to modify/set the style of an artist if the 'style' is created as a separate class and store against the artist as a property. The GraphicsContext class already provides a the basis of a Style class and an artists draw method can be refactored to use the Style class rather than setting up it's own GraphicsContext and transferring it's style-related properties to it. A minimal example of how this could be implemented is shown here: https://github.com/JamesRamm/mpl_experiment

IMO, this will also make the API and code base much neater as individual get/set methods for artist style properties are now redundant... Indirectly related would be a general drive to replace get/set methods with properties. Implementing the style class with properties would be a big stride toward this...

For initial development, I suggest developing a syntax based on a much (much much) simplified version of CSS. I am in favour of dubbing this Artist Style Sheets :+1: :

BNF Grammar

I propose a very simple syntax to implement initially (like a proof of concept), which can be expanded upon in the future. The BNF form of the syntax is given below and then explained

```
RuleSet ::= SelectorSequence "{ "Declaration" }"  
  
SelectorSequence ::= Selector { "," Selector }  
  
Declaration ::= propName ":" propValue ";"  
  
Selector ::= ArtistIdent { "#" Ident }  
  
propName ::= Ident  
  
propValue ::= Ident | Number | Colour | "None"
```

ArtistIdent, Ident, Number and Colour are tokens (the basic building blocks of the expression) which are defined by regular expressions.

Syntax

A CSS stylesheet consists of a series of rule sets in hierarchical order (rules are applied from top to bottom). Each rule follows the syntax

```
selector {attribute: value;}
```

Each rule can have any number of attribute: value pairs, and a stylesheet can have any number of rules.

The initial syntax is designed only for artist primitives. It does not address the question of how to set properties on container types (whose properties may themselves be artists with settable properties), however, a future solution to this could simply be nested RuleSet s

Selectors

Selectors define the object to which the attribute updates should be applied. As a starting point, I propose just 2 selectors to use in initial development:

Artist Type Selector

Select an artist by it's type. E.g Line2D or Text:

```
Line2D {attribute: value}
```

The regex for matching the artist type selector (ArtistIdent in the BNF grammar) would be:

```
ArtistIdent = r'(?P<ArtistIdent>
→\bLine2D\b|\bText\b|\bAxesImage\b|\bFigureImage\b|\bPatch\b)'
```

GID selector

Select an artist by its gid:

```
Line2D#myGID {attribute: value}
```

A gid can be any string, so the regex could be as follows:

```
Ident = r'(?P<Ident>[a-zA-Z_][a-zA-Z_0-9]*)'
```

The above selectors roughly correspond to their CSS counterparts (<http://www.w3.org/TR/CSS21/selectors.html>)

Attributes and values

- Attributes are any valid (settable) property for the artist in question.
- Values are any valid value for the property (Usually a string, or number).

Parsing

Parsing would consist of breaking the stylesheet into tokens (the python cookbook gives a nice tokenizing recipe on page 66), applying the syntax rules and constructing a Tree. This requires defining the grammar of the stylesheet (again, we can borrow from CSS) and writing a parser. Happily, there is a recipe for this in the python cookbook aswell.

Visitor pattern for matplotlib figure

In order to apply the stylesheet rules to the relevant artists, we need to 'visit' each artist in a figure and apply the relevant rule. Here is a visitor class (again, thanks to python cookbook), where each node would be an artist in the figure. A `visit__` method would need to be implemented for each mpl artist, to handle the different properties for each

```
class Visitor:
    def visit(self, node):
        name = 'visit_' + type(node).__name__
        meth = getattr(self, name, None)
        if meth is None:
            raise NotImplementedError
        return meth(node)
```

An evaluator class would then take the stylesheet rules and implement the visitor on each one of them.

87.16.6 Backward compatibility

Implementing a separate Style class would break backward compatibility as many get/set methods on an artist would become redundant. While it would be possible to alter these methods to hook into the Style class (stored as a property against the artist), I would be in favor of simply removing them to both neaten/simplify the codebase and to provide a simple, uncluttered API...

87.16.7 Alternatives

No alternatives, but some of the ground covered here overlaps with MEP25, which may assist in this development

87.16.8 Appendix

Matplotlib primitives

This will form the initial selectors which stylesheets can use.

- Line2D
- Text
- AxesImage
- FigureImage
- Patch

87.17 MEP27: decouple pyplot from backends

- Status
- Branches and Pull requests
- Abstract
- Detailed description
- Implementation
- Future compatibility
- Backward compatibility
- Alternatives
- Questions

87.17.1 Status

Progress

87.17.2 Branches and Pull requests

Main PR (including GTK3): + <https://github.com/matplotlib/matplotlib/pull/4143>

Backend specific branch diffs: + <https://github.com/OceanWolf/matplotlib/compare/backend-refactor...OceanWolf:backend-refactor-tkagg> + <https://github.com/OceanWolf/matplotlib/compare/backend-refactor...OceanWolf:backend-refactor-qt> + <https://github.com/OceanWolf/matplotlib/compare/backend-refactor...backend-refactor-wx>

87.17.3 Abstract

This MEP refactors the backends to give a more structured and consistent API, removing generic code and consolidate existing code. To do this we propose splitting:

1. FigureManagerBase and its derived classes into the core functionality class FigureManager and a backend specific class WindowBase and
2. ShowBase and its derived classes into Gcf.show_all and MainLoopBase.

87.17.4 Detailed description

This MEP aims to consolidate the backends API into one single uniform API, removing generic code out of the backend (which includes `__pylab__` helpers and Gcf), and push code to a more appropriate level in matplotlib. With this we automatically remove inconsistencies that appear in the backends,

such as `FigureManagerBase.resize(w, h)` which sometimes sets the canvas, and other times set the entire window to the dimensions given, depending on the backend.

Two main places for generic code appear in the classes derived from `FigureManagerBase` and `ShowBase`.

1. `FigureManagerBase` has three jobs at the moment:
 1. The documentation describes it as a “Helper class for pyplot mode, wraps everything up into a neat bundle”
 2. But it doesn’t just wrap the canvas and toolbar, it also does all of the windowing tasks itself. The conflation of these two tasks gets seen the best in the following line: ``python self.set_window_title("Figure %d" % num)`` This combines backend specific code `self.set_window_title(title)` with matplotlib generic code `title = "Figure %d" % num`.
 3. Currently the backend specific subclass of `FigureManager` decides when to end the mainloop. This also seems very wrong as the figure should have no control over the other figures.
2. `ShowBase` has two jobs:
 1. It has the job of going through all figure managers registered in `__pylab_helpers.Gcf` and telling them to show themselves.
 2. And secondly it has the job of performing the backend specific mainloop to block the main programme and thus keep the figures from dying.

87.17.5 Implementation

The description of this MEP gives us most of the solution:

1. To remove the windowing aspect out of `FigureManagerBase` letting it simply wrap this new class along with the other backend classes. Create a new `WindowBase` class that can handle this functionality, with pass-through methods (`:arrow_right:`) to `WindowBase`. Classes that subclass `WindowBase` should also subclass the GUI specific window class to ensure backward compatibility (`manager.window == manager.window`).
2. Refactor the mainloop of `ShowBase` into `MainLoopBase`, which encapsulates the end of the loop as well. We give an instance of `MainLoop` to `FigureManager` as a key unlock the exit method (requiring all keys returned before the loop can die). Note this opens the possibility for multiple backends to run concurrently.
3. Now that `FigureManagerBase` has no backend specifics in it, to rename it to `FigureManager`, and move to a new file `backend_managers.py` noting that:
 1. This allows us to break up the conversion of backends into separate PRs as we can keep the existing `FigureManagerBase` class and its dependencies intact.
 2. and this also anticipates MEP22 where the new `NavigationBase` has morphed into a backend independent `ToolManager`.

FigureManager-Base(canvas, num)	FigureManager(figure, num)	Window-Base(title)	Notes
show		show	
destroy	calls destroy on all components	destroy	
full_screen_toggle	handles logic	set_fullscreen	
resize		resize	
key_press	key_press		
get_window_title		get_window_title	
set_window_title		set_window_title	
	_get_toolbar		A common method to all subclasses of FigureManagerBase
		set_default_size	
		add_element_to_window	

Show-Base	MainLoop-Base	Notes
mainloop	begin	
	end	Gets called automatically when no more instances of the subclass exist
__call__		Method moved to Gcf.show_all

87.17.6 Future compatibility

As eluded to above when discussing MEP 22, this refactor makes it easy to add in new generic features. At the moment, MEP 22 has to make ugly hacks to each class extending from FigureManagerBase. With this code, this only needs to get made in the single FigureManager class. This also makes the later deprecation of NavigationToolbar2 very straightforward, only needing to touch the single FigureManager class

MEP 23 makes for another use case where this refactored code will come in very handy.

87.17.7 Backward compatibility

As we leave all backend code intact, only adding missing methods to existing classes, this should work seamlessly for all use cases. The only difference will lie for backends that used FigureManager.resize to resize the canvas and not the window, due to the standardisation of the API.

I would envision that the classes made obsolete by this refactor get deprecated and removed on the same timetable as NavigationToolbar2, also note that the change in call signature to the FigureCanvasWx constructor, while backward compatible, I think the old (imho ugly style) signature should get deprecated and removed in the same manner as everything else.

back-end	manager.resize(w,h)	Extra
gtk3	window	
Tk	canvas	
Qt	window	
Wx	canvas	FigureManagerWx had frame as an alias to window, so this also breaks BC.

87.17.8 Alternatives

If there were any alternative solutions to solving the same problem, they should be discussed here, along with a justification for the chosen approach.

87.17.9 Questions

Mdehoon: Can you elaborate on how to run multiple backends concurrently?

OceanWolf: @mdehoon, as I say, not for this MEP, but I see this MEP opens it up as a future possibility. Basically the MainLoopBase class acts a per backend Gcf, in this MEP it tracks the number of figures open per backend, and manages the mainloops for those backends. It closes the backend specific mainloop when it detects that no figures remain open for that backend. Because of this I imagine that with only a small amount of tweaking that we can do full-multi-backend matplotlib. No idea yet why one would want to, but I leave the possibility there in MainLoopBase. With all the backend-code specifics refactored out of FigureManager also aids in this, one manager to rule them (the backends) all.

Mdehoon: @OceanWolf, OK, thanks for the explanation. Having a uniform API for the backends is very important for the maintainability of matplotlib. I think this MEP is a step in the right direction.

87.18 MEP28: Remove Complexity from Axes.boxplot

- Status
- Branches and Pull requests
- Abstract
- Detailed description
 - Importance
- Implementation
 - Passing transform functions to `cbook.boxplots_stats`

- Simplifications to the Axes.boxplot API and other functions
- Backward compatibility
 - Schedule
 - Anticipated Impacts to Users
 - Anticipated Impacts to Downstream Libraries
- Alternatives
 - Variations on the theme
 - Doing less
 - Doing nothing

87.18.1 Status

Discussion

87.18.2 Branches and Pull requests

The following lists any open PRs or branches related to this MEP:

1. Deprecate redundant statistical kwargs in Axes.boxplot: <https://github.com/phobson/matplotlib/tree/MEP28-initial-deprecations>
2. Deprecate redundant style options in Axes.boxplot: <https://github.com/phobson/matplotlib/tree/MEP28-initial-deprecations>
3. Deprecate passings 2D numpy arrays as input: None
4. Add pre- & post-processing options to cbook.boxplot_stats: <https://github.com/phobson/matplotlib/tree/boxplot-stat-transforms>
5. Exposing cbook.boxplot_stats through Axes.boxplot kwargs: None
6. Remove redundant statistical kwargs in Axes.boxplot: None
7. Remove redundant style options in Axes.boxplot: None
8. Remaining items that arise through discussion: None

87.18.3 Abstract

Over the past few releases, the Axes.boxplot method has grown in complexity to support fully customizable artist styling and statistical computation. This lead to Axes.boxplot being split off into multiple parts. The statistics needed to draw a boxplot are computed in cbook.boxplot_stats, while the actual artists are drawn by Axes.bxp. The original method, Axes.boxplot remains as the most public API that handles passing the user-supplied data to cbook.boxplot_stats, feeding the results to Axes.bxp, and pre-processing style information for each facet of the boxplot plots.

This MEP will outline a path forward to rollback the added complexity and simplify the API while maintaining reasonable backwards compatibility.

87.18.4 Detailed description

Currently, the `Axes.boxplot` method accepts parameters that allow the users to specify medians and confidence intervals for each box that will be drawn in the plot. These were provided so that advanced users could provide statistics computed in a different fashion than the simple method provided by matplotlib. However, handling this input requires complex logic to make sure that the forms of the data structure match what needs to be drawn. At the moment, that logic contains 9 separate if/else statements nested up to 5 levels deep with a for loop, and may raise up to 2 errors. These parameters were added prior to the creation of the `Axes.bxp` method, which draws boxplots from a list of dictionaries containing the relevant statistics. Matplotlib also provides a function that computes these statistics via `cbook.boxplot_stats`. Note that advanced users can now either a) write their own function to compute the stats required by `Axes.bxp`, or b) modify the output returned by `cbook.boxplots_stats` to fully customize the position of the artists of the plots. With this flexibility, the parameters to manually specify only the medians and their confidence intervals remain for backwards compatibility.

Around the same time that the two roles of `Axes.boxplot` were split into `cbook.boxplot_stats` for computation and `Axes.bxp` for drawing, both `Axes.boxplot` and `Axes.bxp` were written to accept parameters that individually toggle the drawing of all components of the boxplots, and parameters that individually configure the style of those artists. However, to maintain backwards compatibility, the `sym` parameter (previously used to specify the symbol of the fliers) was retained. This parameter itself requires fairly complex logic to reconcile the `sym` parameters with the newer `flierprops` parameter at the default style specified by matplotlibrc.

This MEP seeks to dramatically simplify the creation of boxplots for novice and advanced users alike. Importantly, the changes proposed here will also be available to downstream packages like seaborn, as seaborn smartly allows users to pass arbitrary dictionaries of parameters through the seaborn API to the underlying matplotlib functions.

This will be achieved in the following way:

1. `cbook.boxplot_stats` will be modified to allow pre- and post- computation transformation functions to be passed in (e.g., `np.log` and `np.exp` for lognormally distributed data)
2. `Axes.boxplot` will be modified to also accept and naïvely pass them to `cbook.boxplots_stats` (Alt: pass the stat function and a dict of its optional parameters).
3. Outdated parameters from `Axes.boxplot` will be deprecated and later removed.

Importance

Since the limits of the whiskers are computed arithmetically, there is an implicit assumption of normality in box and whisker plots. This primarily affects which data points are classified as outliers.

Allowing transformations to the data and the results used to draw boxplots will allow users to opt-out of that assumption if the data are known to not fit a normal distribution.

Below is an example of how `Axes.boxplot` classifies outliers of lognormal data differently depending on these types of transforms.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cbook
np.random.seed(0)

fig, ax = plt.subplots(figsize=(4, 6))
ax.set_yscale('log')
data = np.random.lognormal(-1.75, 2.75, size=37)

stats = cbook.boxplot_stats(data, labels=['arithmetic'])
logstats = cbook.boxplot_stats(np.log(data), labels=['log-transformed'])

for lsdict in logstats:
    for key, value in lsdict.items():
        if key != 'label':
            lsdict[key] = np.exp(value)

stats.extend(logstats)
ax.bxp(stats)
fig.show()
```

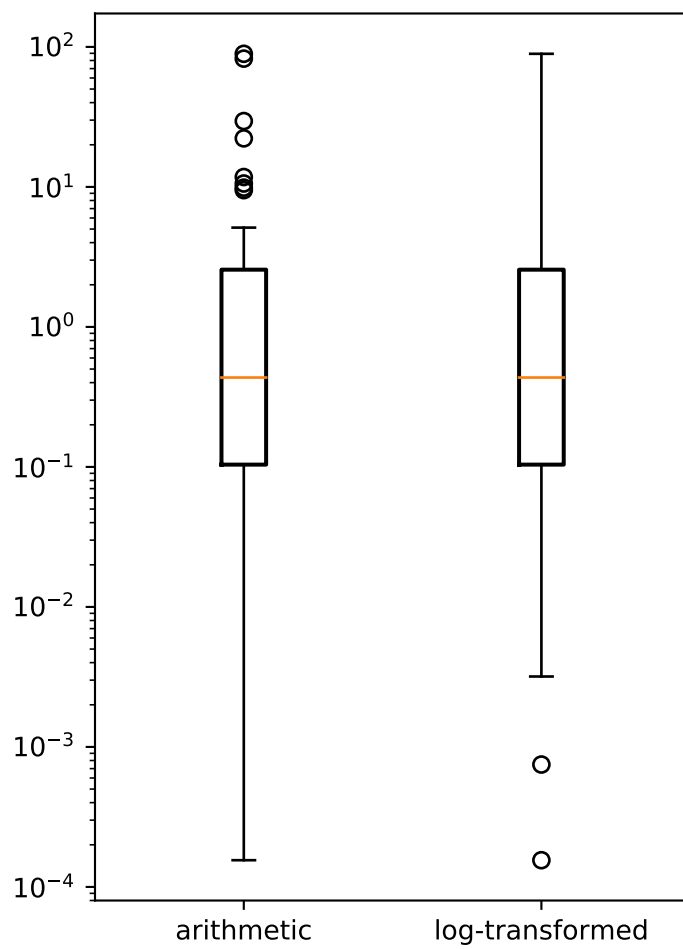
87.18.5 Implementation

Passing transform functions to `cbook.boxplots_stats`

This MEP proposes that two parameters (e.g., `transform_in` and `transform_out`) be added to the `cookbook` function that computes the statistics for the `boxplot` function. These will be optional keyword-only arguments and can easily be set to `lambda x: x` as a no-op when omitted by the user. The `transform_in` function will be applied to the data as the `boxplot_stats` function loops through each subset of the data passed to it. After the list of statistics dictionaries are computed the `transform_out` function is applied to each value in the dictionaries.

These transformations can then be added to the call signature of `Axes.boxplot` with little impact to that method's complexity. This is because they can be directly passed to `cbook.boxplot_stats`. Alternatively, `Axes.boxplot` could be modified to accept an optional statistical function `kwarg` and a dictionary of parameters to be directly passed to it.

At this point in the implementation users and external libraries like `seaborn` would have complete control via the `Axes.boxplot` method. More importantly, at the very least, `seaborn` would require no changes to its API to allow users to take advantage of these new options.



Simplifications to the Axes.boxplot API and other functions

Simplifying the boxplot method consists primarily of deprecating and then removing the redundant parameters. Optionally, a next step would include rectifying minor terminological inconsistencies between Axes.boxplot and Axes.bxp.

The parameters to be deprecated and removed include:

1. usermedians - processed by 10 SLOC, 3 if blocks, a for loop
2. conf_intervals - handled by 15 SLOC, 6 if blocks, a for loop
3. sym - processed by 12 SLOC, 4 if blocks

Removing the sym option allows all code in handling the remaining styling parameters to be moved to Axes.bxp. This doesn't remove any complexity, but does reinforce the single responsibility principle among Axes.bxp, cbook.boxplot_stats, and Axes.boxplot.

Additionally, the notch parameter could be renamed shownotches to be consistent with Axes.bxp. This kind of cleanup could be taken a step further and the whis, bootstrap, autorange could be rolled into the kwargs passed to the new statfxn parameter.

87.18.6 Backward compatibility

Implementation of this MEP would eventually result in the backwards incompatible deprecation and then removal of the keyword parameters usermedians, conf_intervals, and sym. Cursory searches on GitHub indicated that usermedians, conf_intervals are used by few users, who all seem to have a very strong knowledge of matplotlib. A robust deprecation cycle should provide sufficient time for these users to migrate to a new API.

Deprecation of sym however, may have a much broader reach into the matplotlib userbase.

Schedule

An accelerated timeline could look like the following:

1. v2.0.1 add transforms to cbook.boxplots_stats, expose in Axes.boxplot
2. v2.1.0 Initial Deprecations , and using 2D numpy arrays as input
 - a. Using 2D numpy arrays as input. The semantics around 2D arrays are generally confusing.
 - b. usermedians, conf_intervals, sym parameters
3. v2.2.0
 - a. remove usermedians, conf_intervals, sym parameters
 - b. deprecate notch in favor of shownotches to be consistent with other parameters and Axes.bxp
4. v2.3.0

- a. remove notch parameter
- b. move all style and artist toggling logic to `Axes.bxp` such `Axes.boxplot` is little more than a broker between `Axes.bxp` and `cbook.boxplots_stats`

Anticipated Impacts to Users

As described above deprecating `usermedians` and `conf_intervals` will likely impact few users. Those who will be impacted are almost certainly advanced users who will be able to adapt to the change.

Deprecating the `sym` option may import more users and effort should be taken to collect community feedback on this.

Anticipated Impacts to Downstream Libraries

The source code (GitHub master as of 2016-10-17) was inspected for `seaborn` and `python-ggplot` to see if these changes would impact their use. None of the parameters nominated for removal in this MEP are used by `seaborn`. The `seaborn` APIs that use `matplotlib`'s `boxplot` function allow user's to pass arbitrary `**kwargs` through to `matplotlib`'s API. Thus `seaborn` users with modern `matplotlib` installations will be able to take full advantage of any new features added as a result of this MEP.

`Python-ggplot` has implemented its own function to draw boxplots. Therefore, no impact can come to it as a result of implementing this MEP.

87.18.7 Alternatives

Variations on the theme

This MEP can be divided into a few loosely coupled components:

1. Allowing pre- and post-computation transformation function in `cbook.boxplot_stats`
2. Exposing that transformation in the `Axes.boxplot` API
3. Removing redundant statistical options in `Axes.boxplot`
4. Shifting all styling parameter processing from `Axes.boxplot` to `Axes.bxp`.

With this approach, #2 depends on #1, and #4 depends on #3.

There are two possible approaches to #2. The first and most direct would be to mirror the new `transform_in` and `transform_out` parameters of `cbook.boxplot_stats` in `Axes.boxplot` and pass them directly.

The second approach would be to add `statfxn` and `statfxn_args` parameters to `Axes.boxplot`. Under this implementation, the default value of `statfxn` would be `cbook.boxplot_stats`, but users could pass their own function. Then `transform_in` and `transform_out` would then be passed as elements of the `statfxn_args` parameter.

```

def boxplot_stats(data, ..., transform_in=None, transform_out=None):
    if transform_in is None:
        transform_in = lambda x: x

    if transform_out is None:
        transform_out = lambda x: x

    output = []
    for _d in data:
        d = transform_in(_d)
        stat_dict = do_stats(d)
        for key, value in stat_dict.item():
            if key != 'label':
                stat_dict[key] = transform_out(value)
        output.append(d)
    return output

class Axes(...):
    def boxplot_option1(data, ..., transform_in=None, transform_out=None):
        stats = cbook.boxplot_stats(data, ...,
                                     transform_in=transform_in,
                                     transform_out=transform_out)
        return self.bxp(stats, ...)

    def boxplot_option2(data, ..., statfxn=None, **statopts):
        if statfxn is None:
            statfxn = boxplot_stats
        stats = statfxn(data, **statopts)
        return self.bxp(stats, ...)

```

Both cases would allow users to do the following:

```

fig, ax1 = plt.subplots()
artists1 = ax1.boxplot_optionX(data, transform_in=np.log,
                               transform_out=np.exp)

```

But Option Two lets a user write a completely custom stat function (e.g., `my_box_stats`) with fancy BCA confidence intervals and the whiskers set differently depending on some attribute of the data.

This is available under the current API:

```

fig, ax1 = plt.subplots()
my_stats = my_box_stats(data, bootstrap_method='BCA',
                        whisker_method='dynamic')
ax1.bxp(my_stats)

```

And would be more concise with Option Two

```
fig, ax = plt.subplots()
statopts = dict(transform_in=np.log, transform_out=np.exp)
ax.boxplot(data, ..., **statopts)
```

Users could also pass their own function to compute the stats:

```
fig, ax1 = plt.subplots()
ax1.boxplot(data, statfxn=my_box_stats, bootstrap_method='BCA',
            whisker_method='dynamic')
```

From the examples above, Option Two seems to have only marginal benefit, but in the context of downstream libraries like seaborn, its advantage is more apparent as the following would be possible without any patches to seaborn:

```
import seaborn
tips = seaborn.load_data('tips')
g = seaborn.factorplot(x="day", y="total_bill", hue="sex", data=tips,
                      kind='box', palette="PRGn", shownotches=True,
                      statfxn=my_box_stats, bootstrap_method='BCA',
                      whisker_method='dynamic')
```

This type of flexibility was the intention behind splitting the overall boxplot API in the current three functions. In practice however, downstream libraries like seaborn support versions of matplotlib dating back well before the split. Thus, adding just a bit more flexibility to the Axes.boxplot could expose all the functionality to users of the downstream libraries with modern matplotlib installation without intervention from the downstream library maintainers.

Doing less

Another obvious alternative would be to omit the added pre- and post- computation transform functionality in `cbook.boxplot_stats` and `Axes.boxplot`, and simply remove the redundant statistical and style parameters as described above.

Doing nothing

As with many things in life, doing nothing is an option here. This means we simply advocate for users and downstream libraries to take advantage of the split between `cbook.boxplot_stats` and `Axes.bxp` and let them decide how to provide an interface to that.

87.19 MEP29: Text light markup

- Status
- Branches and Pull requests
- Abstract
- Detailed description
- Implementation
 - Improvements
 - Problems
- Backward compatibility
- Alternatives

87.19.1 Status

Discussion

87.19.2 Branches and Pull requests

None at the moment, proof of concept only.

87.19.3 Abstract

This MEP proposes to add lightweight markup to the text artist.

87.19.4 Detailed description

Using different size/color/family in a text annotation is difficult because the text method accepts argument for size/color/family/weight/etc. that are used for the whole text. But, if one wants, for example, to have different colors, one has to look at the gallery where one such example is provided: http://matplotlib.org/examples/text_labels_and_annotations/rainbow_text.html

This example takes a list of strings as well as a list of colors which makes it cumbersome to use. An alternative would be to use a restricted set of pango-like markup (see <https://developer.gnome.org/pango/stable/PangoMarkupFormat.html>) and to interpret this markup.

Some markup examples:

```
Hello <b>world!</b>`
Hello <span color="blue">world!</span>
```

87.19.5 Implementation

A proof of concept is provided in `markup_example.py` but it currently only handles the horizontal direction.

Improvements

- This proof of concept uses regex to parse the text but it may be better to use the `html.parser` from the standard library.
- Computation of text fragment positions could benefit from the `OffsetFrom` class. See for example item 5 in [Using Complex Coordinates with Annotations](#)

Problems

- One serious problem is how to deal with text having both latex and html-like tags. For example, consider the following:

`$Bold$`

Recommendation would be to have mutual exclusion.

87.19.6 Backward compatibility

None at the moment since it is only a proof of concept

87.19.7 Alternatives

As proposed by @anntzer, this could be also implemented as improvements to `mathtext`. For example:

```
r"$\text{Hello \textbf{world}}$"
r"$\text{Hello \textcolor{blue}{world}}$"
r"$\text{Hello \textsf{\small world}}$"
```

LICENSES

Matplotlib only uses BSD compatible code. If you bring in code from another project make sure it has a PSF, BSD, MIT or compatible license (see the Open Source Initiative [licenses page](#) for details on individual licenses). If it doesn't, you may consider contacting the author and asking them to relicense it. GPL and LGPL code are not acceptable in the main code base, though we are considering an alternative way of distributing L/GPL code through an separate channel, possibly a toolkit. If you include code, make sure you include a copy of that code's license in the license directory if the code's license requires you to distribute the license with it. Non-BSD compatible licenses are acceptable in matplotlib toolkits (e.g., basemap), but make sure you clearly state the licenses you are using.

88.1 Why BSD compatible?

The two dominant license variants in the wild are GPL-style and BSD-style. There are countless other licenses that place specific restrictions on code reuse, but there is an important difference to be considered in the GPL and BSD variants. The best known and perhaps most widely used license is the GPL, which in addition to granting you full rights to the source code including redistribution, carries with it an extra obligation. If you use GPL code in your own code, or link with it, your product must be released under a GPL compatible license. i.e., you are required to give the source code to other people and give them the right to redistribute it as well. Many of the most famous and widely used open source projects are released under the GPL, including linux, gcc, emacs and sage.

The second major class are the BSD-style licenses (which includes MIT and the python PSF license). These basically allow you to do whatever you want with the code: ignore it, include it in your own open source project, include it in your proprietary product, sell it, whatever. python itself is released under a BSD compatible license, in the sense that, quoting from the PSF license page:

There **is** no GPL-like "copyleft" restriction. Distributing binary-only versions of Python, modified **or not**, **is** allowed. There **is** no requirement to release **any** of your source code. You can also write extension modules **for** Python **and** provide them only **in** binary form.

Famous projects released under a BSD-style license in the permissive sense of the last paragraph are the BSD operating system, python and TeX.

There are several reasons why early matplotlib developers selected a BSD compatible license. matplotlib is a python extension, and we choose a license that was based on the python license (BSD compatible). Also, we wanted to attract as many users and developers as possible, and many software companies will not use GPL code in software they plan to distribute, even those that are highly committed to open source development, such as [enthought](#), out of legitimate concern that use of the GPL will "infect" their code base by its viral nature. In effect, they want to retain the right to release some proprietary code. Companies and institutions who use matplotlib often make significant contributions, because they have the resources to get a job done, even a boring one. Two of the matplotlib backends (FLTK and WX) were contributed by private companies. The final reason behind the licensing choice is compatibility with the other python extensions for scientific computing: ipython, numpy, scipy, the enthought tool suite and python itself are all distributed under BSD compatible licenses.

DEFAULT COLOR CHANGES

As discussed at length elsewhere [insert links], jet is an empirically bad color map and should not be the default color map. Due to the position that changing the appearance of the plot breaks backward compatibility, this change has been put off for far longer than it should have been. In addition to changing the default color map we plan to take the chance to change the default color-cycle on plots and to adopt a different color map for filled plots (imshow, pcolor, contourf, etc) and for scatter like plots.

89.1 Default Heat Map Colormap

The choice of a new color map is fertile ground to bike-shedding ("No, it should be `_this_` color") so we have a proposed set criteria (via Nathaniel Smith) to evaluate proposed color maps.

- it should be a sequential colormap, because diverging colormaps are really misleading unless you know where the "center" of the data is, and for a default colormap we generally won't.
- it should be perceptually uniform, i.e., human subjective judgments of how far apart nearby colors are should correspond as linearly as possible to the difference between the numerical values they represent, at least locally.
- it should have a perceptually uniform luminance ramp, i.e. if you convert to greyscale it should still be uniform. This is useful both in practical terms (greyscale printers are still a thing!) and because luminance is a very strong and natural cue to magnitude.
- it should also have some kind of variation in hue, because hue variation is a really helpful additional cue to perception, having two cues is better than one, and there's no reason not to do it.
- the hue variation should be chosen to produce reasonable results even for viewers with the more common types of colorblindness. (Which rules out things like red-to-green.)
- For bonus points, it would be nice to choose a hue ramp that still works if you throw away the luminance variation, because then we could use the version with varying luminance for 2d plots, and the version with just hue variation for 3d plots. (In 3d plots you really want to reserve the luminance channel for lighting/shading, because your brain is really good at extracting 3d shape from luminance variation. If the 3d surface itself has massively varying luminance then this screws up the ability to see shape.)
- Not infringe any existing IP

89.1.1 Example script

89.1.2 Proposed Colormaps

89.2 Default Scatter Colormap

For heat-map like applications it can be desirable to cover as much of the luminence scale as possible, however when color mapping markers, having markers too close to white can be a problem. For that reason we propose using a different (but maybe related) color map to the heat map for marker-based. The design parameters are the same as above, only with a more limited luminence variation.

89.2.1 Example script

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(1234)

fig, (ax1, ax2) = plt.subplots(1, 2)

N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = np.pi * (15 * np.random.rand(N))**2 # 0 to 15 point radiuses

ax1.scatter(x, y, s=area, c=colors, alpha=0.5)


X,Y = np.meshgrid(np.arange(0, 2*np.pi, .2),
                  np.arange(0, 2*np.pi, .2))
U = np.cos(X)
V = np.sin(Y)
Q = ax2.quiver(X, Y, U, V, units='width')
qd = np.random.rand(np.prod(X.shape))
Q.set_array(qd)
```

89.2.2 Proposed Colormaps

89.3 Color Cycle / Qualitative color map

When plotting lines it is frequently desirable to plot multiple lines or artists which need to be distinguishable, but there is no inherent ordering.

89.3.1 Example script

```
import numpy as np
import matplotlib.pyplot as plt

fig, (ax1, ax2) = plt.subplots(1, 2)

x = np.linspace(0, 1, 10)

for j in range(10):
    ax1.plot(x, x * j)

th = np.linspace(0, 2*np.pi, 1024)
for j in np.linspace(0, np.pi, 10):
    ax2.plot(th, np.sin(th + j))

ax2.set__xlim(0, 2*np.pi)
```

89.3.2 Proposed Color cycle

Part VIII

Glossary

AGG The Anti-Grain Geometry ([Agg](#)) rendering engine, capable of rendering high-quality images

Cairo The [Cairo](#) graphics engine

dateutil The [dateutil](#) library provides extensions to the standard datetime module

EPS Encapsulated Postscript ([EPS](#))

FreeType [FreeType](#) is a font rasterization library used by matplotlib which supports TrueType, Type 1, and OpenType fonts.

GDK The Gimp Drawing Kit for GTK+

GTK The GIMP Toolkit ([GTK](#)) graphical user interface library

JPG The Joint Photographic Experts Group ([JPEG](#)) compression method and file format for photographic images

numpy [numpy](#) is the standard numerical array library for python, the successor to Numeric and numarray. numpy provides fast operations for homogeneous data sets and common mathematical operations like correlations, standard deviation, fourier transforms, and convolutions.

PDF Adobe's Portable Document Format ([PDF](#))

PNG Portable Network Graphics ([PNG](#)), a raster graphics format that employs lossless data compression which is more suitable for line art than the lossy jpg format. Unlike the gif format, png is not encumbered by requirements for a patent license.

PS Postscript ([PS](#)) is a vector graphics ASCII text language widely used in printers and publishing. Postscript was developed by adobe systems and is starting to show its age: for example it does not have an alpha channel. PDF was designed in part as a next-generation document format to replace postscript

pgi pgi exists as a relatively new Python wrapper to GTK3 and acts as a pure python alternative to PyGObject. pgi still exists in its infancy, currently missing many features of PyGObject. However Matplotlib does not use any of these missing features.

pygtk [pygtk](#) provides python wrappers for the GTK widgets library for use with the GTK or GTKAgg backend. Widely used on linux, and is often packaged as 'python-gtk2'

PyGObject Like pygtk, PyGObject provides python wrappers for the GTK widgets library; unlike pygtk, PyGObject wraps GTK3 instead of the now obsolete GTK2.

pyqt [pyqt](#) provides python wrappers for the Qt widgets library and is required by the matplotlib Qt5Agg and Qt4Agg backends. Widely used on linux and windows; many linux distributions package this as 'python-qt5' or 'python-qt4'.

python [python](#) is an object oriented interpreted language widely used for scripting, application development, web application servers, scientific computing and more.

pytz [pytz](#) provides the Olson tz database in Python. it allows accurate and cross platform timezone calculations and solves the issue of ambiguous times at the end of daylight savings

Qt [Qt](#) is a cross-platform application framework for desktop and embedded development.

Qt4 [Qt4](#) is the previous, but most widely used, version of Qt cross-platform application framework for desktop and embedded development.

Qt5 **Qt5** is the current version of Qt cross-platform application framework for desktop and embedded development.

raster graphics **Raster graphics**, or bitmaps, represent an image as an array of pixels which is resolution dependent. Raster graphics are generally most practical for photo-realistic images, but do not scale easily without loss of quality.

SVG The Scalable Vector Graphics format (**SVG**). An XML based vector graphics format supported by many web browsers.

TIFF Tagged Image File Format (**TIFF**) is a file format for storing images, including photographs and line art.

Tk **Tk** is a graphical user interface for Tcl and many other dynamic languages. It can produce rich, native applications that run unchanged across Windows, Mac OS X, Linux and more.

vector graphics **vector graphics** use geometrical primitives based upon mathematical equations to represent images in computer graphics. Primitives can include points, lines, curves, and shapes or polygons. Vector graphics are scalable, which means that they can be resized without suffering from issues related to inherent resolution like are seen in raster graphics. Vector graphics are generally most practical for typesetting and graphic design applications.

wxpython **wxpython** provides python wrappers for the wxWidgets library for use with the WX and WXAgg backends. Widely used on linux, OS-X and windows, it is often packaged by linux distributions as 'python-wxgtk'

wxWidgets **WX** is cross-platform GUI and tools library for GTK, MS Windows, and MacOS. It uses native widgets for each operating system, so applications will have the look-and-feel that users on that operating system expect.

BIBLIOGRAPHY

- [colorcet] <https://github.com/bokeh/colorcet>
- [Ware] http://ccom.unh.edu/sites/default/files/publications/Ware_1988_CGA_Color_sequences_univariate_maps.pdf
- [Moreland] <http://www.kennethmoreland.com/color-maps/ColorMapsExpanded.pdf>
- [list-colormaps] <https://gist.github.com/endolith/2719900#id7>
- [mycarta-banding] <https://mycarta.wordpress.com/2012/10/14/the-rainbow-is-deadlong-live-the-rainbow-part-3/>
- [mycarta-jet] <https://mycarta.wordpress.com/2012/10/06/the-rainbow-is-deadlong-live-the-rainbow-part-3/>
- [bw] <http://www.tannerhelland.com/3643/gray-scale-image-algorithm-vb6/>
- [colorblindness] <http://www.color-blindness.com/>
- [vischeck] <http://www.vischeck.com/vischeck/>
- [IBM] <https://dx.doi.org/10.1109/VISUAL.1995.480803>
- [1] Kpathsea documentation The library that kpsewhich is part of.
- [1] Michel Bernadou, Kamal Hassan, "Basis functions for general Hsieh-Clough-Tocher triangles, complete or reduced.", International Journal for Numerical Methods in Engineering, 17(5):784 - 789. 2.01.
- [2] C.T. Kelley, "Iterative Methods for Optimization".

PYTHON MODULE INDEX

m

matplotlib.afm, ??
matplotlib.animation, ??
matplotlib.artist, ??
matplotlib.axis, ??
matplotlib.backend_bases, ??
matplotlib.backend_managers, ??
matplotlib.backend_tools, ??
matplotlib.backends.backend_agg, ??
matplotlib.backends.backend_cairo, ??
matplotlib.backends.backend_mixed, ??
matplotlib.backends.backend_nbagg, ??
matplotlib.backends.backend_pdf, ??
matplotlib.backends.backend_pgf, ??
matplotlib.backends.backend_ps, ??
matplotlib.backends.backend_svg, ??
matplotlib.backends.backend_tkagg, ??
matplotlib.backends.backend_wxagg, ??
matplotlib.cbook, ??
matplotlib.cm, ??
matplotlib.collections, ??
matplotlib.colorbar, ??
matplotlib.colors, ??
matplotlib.container, ??
matplotlib.contour, ??
matplotlib.dates, ??
matplotlib.dviread, ??
matplotlib.figure, ??
matplotlib.font_manager, ??
matplotlib.fontconfig_pattern, ??
matplotlib.gridspec, ??
matplotlib.image, ??
matplotlib.legend, ??
matplotlib.legend_handler, ??
matplotlib.lines, ??
matplotlib.markers, ??
matplotlib.mathtext, ??
matplotlib.mlab, ??
matplotlib.offsetbox, ??
matplotlib.patches, ??
matplotlib.path, ??
matplotlib.patheffects, 199
matplotlib.projections, ??
matplotlib.projections.polar, ??
matplotlib.pyplot, ??
matplotlib.rcsetup, ??
matplotlib.sankey, ??
matplotlib.scale, ??
matplotlib.sphinxext.plot_directive, ??
matplotlib.spines, ??
matplotlib.style, ??
matplotlib.table, ??
matplotlib.text, ??
matplotlib.ticker, ??
matplotlib.tight_layout, ??
matplotlib.transforms, ??
matplotlib.tri, ??
matplotlib.type1font, ??
matplotlib.units, ??
matplotlib.widgets, ??
mpl_toolkits.axes_grid1.anchored_artists, ??
mpl_toolkits.axes_grid1.axes_divider, ??
mpl_toolkits.axes_grid1.axes_grid, ??
mpl_toolkits.axes_grid1.axes_rgb, ??
mpl_toolkits.axes_grid1.axes_size, ??
mpl_toolkits.axes_grid1.colorbar, ??
mpl_toolkits.axes_grid1.inset_locator, ??
mpl_toolkits.axes_grid1.mpl_axes, ??
mpl_toolkits.axes_grid1.parasite_axes, ??
mpl_toolkits.axisartist.angle_helper, ??
mpl_toolkits.axisartist.axes_divider, ??
mpl_toolkits.axisartist.axes_grid, ??
mpl_toolkits.axisartist.axes_rgb, ??
mpl_toolkits.axisartist.axis_artist, ??

`mpl_toolkits.axisartist.axisline_style`, ??
`mpl_toolkits.axisartist.axislines`, ??
`mpl_toolkits.axisartist.clip_path`, ??
`mpl_toolkits.axisartist.floating_axes`, ??
`mpl_toolkits.axisartist.grid_finder`, ??
`mpl_toolkits.axisartist.grid_helper_curvilinear`,
??
`mpl_toolkits.axisartist.parasite_axes`, ??

A

`add_collection3d()`
(`mpl_toolkits.mplot3d.Axes3D`
method), 319

B

`bar()` (`mpl_toolkits.mplot3d.Axes3D` method),
319

C

`contour()` (`mpl_toolkits.mplot3d.Axes3D`
method), 317
`contourf()` (`mpl_toolkits.mplot3d.Axes3D`
method), 317

E

environment variable
MPLBACKEND, 69, 495
PATH, 273, 275, 276, 279

L

legend entry,
105
legend handle,
105
legend key,
105
legend label,
105

M

`matplotlib.patheffects` (module), 199
MPLBACKEND, 69, 495

P

PATH, 273, 275, 276, 279
`plot()` (`mpl_toolkits.mplot3d.Axes3D` method),
311

`plot_surface()` (`mpl_toolkits.mplot3d.Axes3D`
method), 313
`plot_trisurf()` (`mpl_toolkits.mplot3d.Axes3D`
method), 314
`plot_wireframe()` (`mpl_toolkits.mplot3d.Axes3D`
method), 312

Q

`quiver()` (`mpl_toolkits.mplot3d.Axes3D`
method), 320

S

`scatter()` (`mpl_toolkits.mplot3d.Axes3D`
method), 312

T

`text()` (`mpl_toolkits.mplot3d.Axes3D` method),
321