


[Fork me on GitHub](#)

Version 3.1.1

[home](#) | [examples](#) | [tutorials](#) | [API](#) | [contents](#) » [API Overview](#) »

[previous](#) | [next](#) | [modules](#) | [index](#)
[matplotlib.pyplot](#) » [matplotlib.pyplot](#) »

matplotlib.pyplot.plot

```
matplotlib.pyplot.plot(*args, scalex=True, scaley=True,
data=None, **kwargs)
```

[\[source\]](#)

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
>>> plot(x, y, 'bo')     # plot x and y using blue circle markers
>>> plot(y)              # plot y using x as index array 0..len(y)-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:

Quick search

Go

Table of Contents

[matplotlib.pyplot.plot](#)

- [Examples using matplotlib.pyplot.plot](#)

Related Topics

[Documentation overview](#)

- [API Overview](#)
 - [matplotlib.pyplot](#)
 - [matplotlib.pyplot](#)
 - Previous: [matplotlib.pyplot.plasma](#)
 - Next: [matplotlib.pyplot.plot_date](#)

[Show Page Source](#)

```
>>> 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 and default to `range(len(y))`.

Commonly, these parameters are 1D arrays.

They can also be scalars, or two-dimensional (in that case, the columns represent separate data sets).

These arguments cannot be passed as keywords.

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.

This argument cannot be passed as keyword.

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
<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

Property	Description
<code>animated</code>	bool
<code>antialiased</code> or <code>aa</code>	bool
<code>clip_box</code>	<code>Bbox</code>
<code>clip_on</code>	bool
<code>clip_path</code>	<code>[(Path, Transform) Patch None]</code>
<code>color</code> or <code>c</code>	color
<code>contains</code>	callable
<code>dash_capstyle</code>	<code>{'butt', 'round', 'projecting'}</code>
<code>dash_joinstyle</code>	<code>{'miter', 'round', 'bevel'}</code>
<code>dashes</code>	sequence of floats (on/off ink in points) or (None, None)
<code>drawstyle</code> or <code>ds</code>	<code>{'default', 'steps', 'steps-pre', 'steps-mid', 'steps-post'}</code> , default: 'default'
<code>figure</code>	<code>Figure</code>
<code>fillstyle</code>	<code>{'full', 'left', 'right', 'bottom', 'top', 'none'}</code>
<code>gid</code>	str
<code>in_layout</code>	bool
<code>label</code>	object
<code>linestyle</code> or <code>ls</code>	<code>{'-', '--', '-.', ':', '', (offset, on-off-seq), ...}</code>
<code>linewidth</code> or <code>lw</code>	float
<code>marker</code>	marker style
<code>markeredgecolor</code> or <code>mec</code>	color
<code>markeredgewidth</code> or <code>mew</code>	float
<code>markerfacecolor</code> or <code>mfc</code>	color
<code>markerfacecoloralt</code> or <code>mfcalt</code>	color
<code>markersize</code> or <code>ms</code>	float
<code>markevery</code>	None or int or (int, int) or slice or List[int] or float or (float, float)
<code>path_effects</code>	<code>AbstractPathEffect</code>
<code>picker</code>	float or callable[[Artist, Event], Tuple[bool, dict]]
<code>pickradius</code>	float
<code>rasterized</code>	bool or None
<code>sketch_params</code>	(scale: float, length: float, randomness: float)
<code>snap</code>	bool or None
<code>solid_capstyle</code>	<code>{'butt', 'round', 'projecting'}</code>
<code>solid_joinstyle</code>	<code>{'miter', 'round', 'bevel'}</code>
<code>transform</code>	<code>matplotlib.transforms.Transform</code>
<code>url</code>	str

Property	Description	
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 = '[marker][line][color]'
```

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.

Other combinations such as [color][marker][line] are also supported, but note that their parsing may be ambiguous.

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

character	description
'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
'or'     # red circles
'-g'     # green solid line
'--'     # dashed line with default color
'^k:'    # black triangle_up markers connected by a dotted line
```

Colors

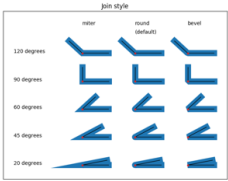
The supported color abbreviations are the single letter codes

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

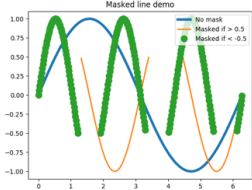
and the 'CN' colors that index into the default property cycle.

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').

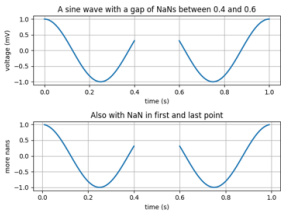
Examples using matplotlib.pyplot.plot



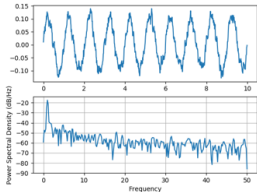
Join styles and cap styles



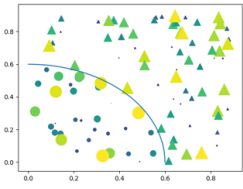
Masked Demo



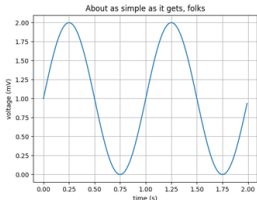
Nan Test



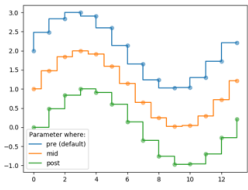
Psd Demo



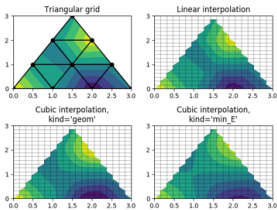
Scatter Masked



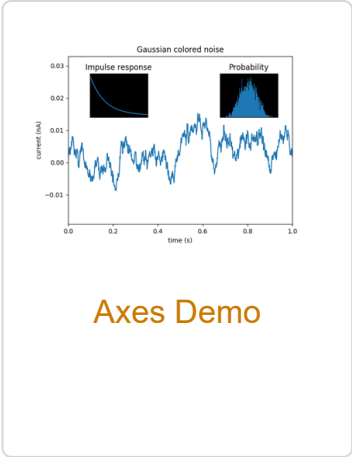
Simple Plot



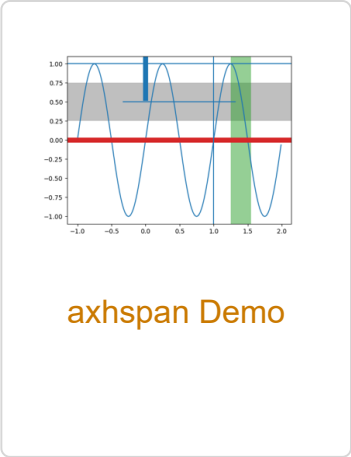
Step Demo



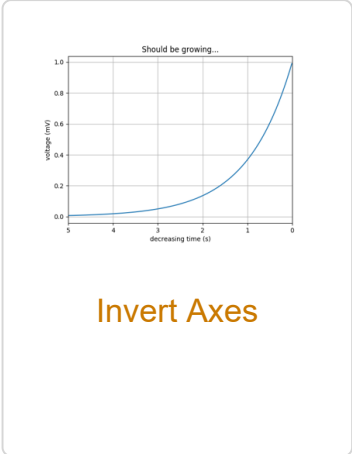
Triinterp Demo



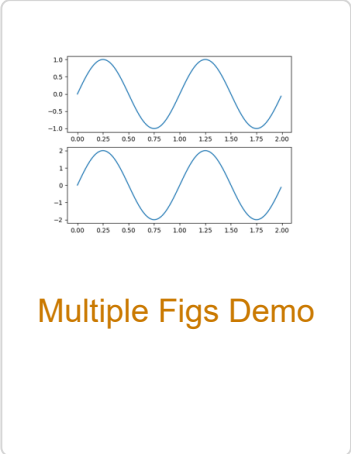
Axes Demo



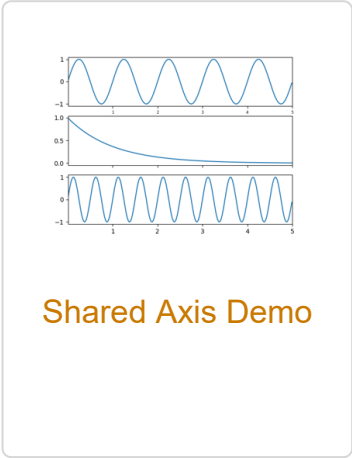
axhspan Demo



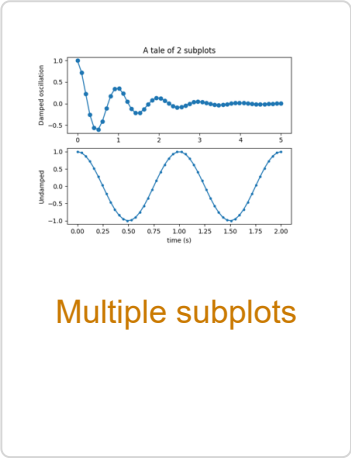
Invert Axes



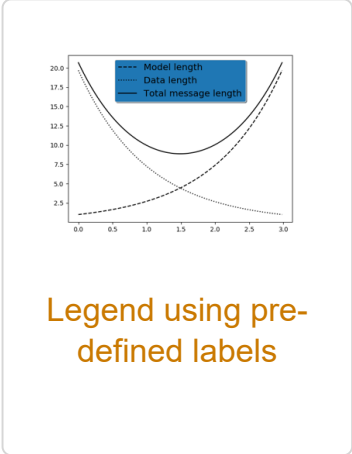
Multiple Figs Demo



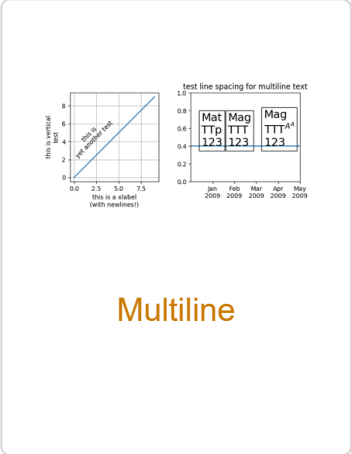
Shared Axis Demo



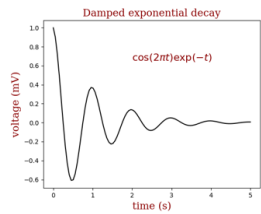
Multiple subplots



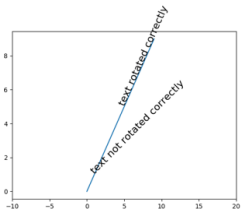
Legend using pre-defined labels



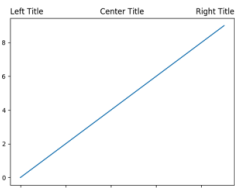
Multiline



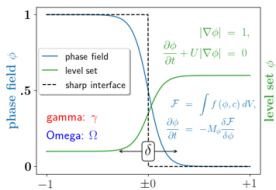
Controlling style of text and labels using a dictionary



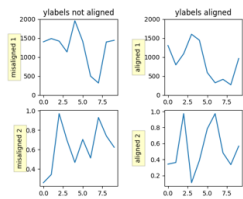
Text Rotation Relative To Line



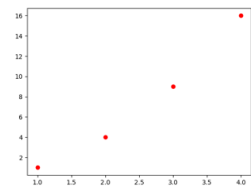
Titles Demo



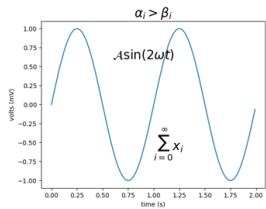
Usetex Demo



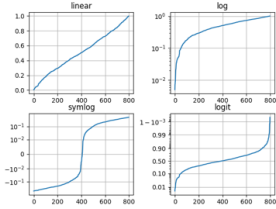
Align y-labels



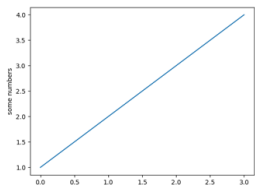
plot() format string



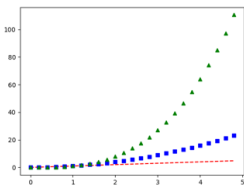
Pyplot Mathtext



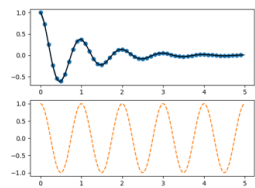
Pyplot Scales



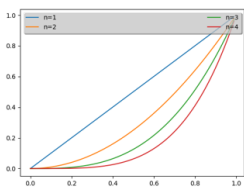
Pyplot Simple



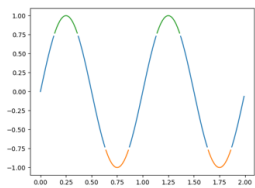
Pyplot Three



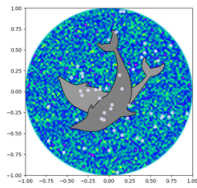
Pyplot Two Subplots



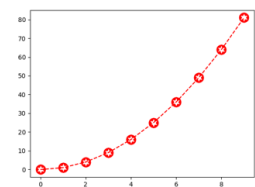
Whats New 0.98.4
Legend



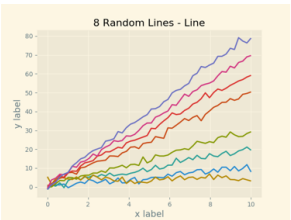
Color by y-value



Dolphins



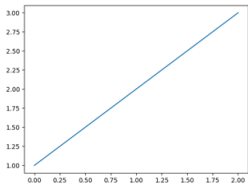
Marker Path



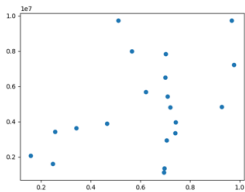
Solarized Light
stylesheet



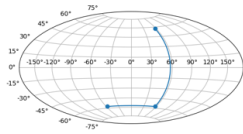
Frame grabbing



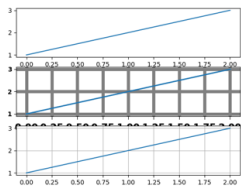
Agg Buffer



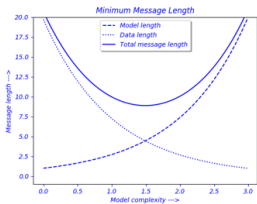
Coords Report



Custom projection



Customize Rc



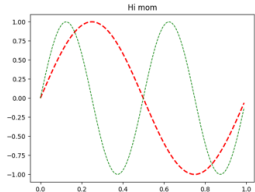
Findobj Demo



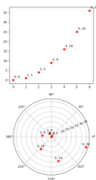
Multipage PDF



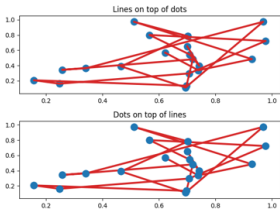
Print Stdout



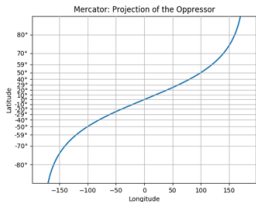
Set And Get



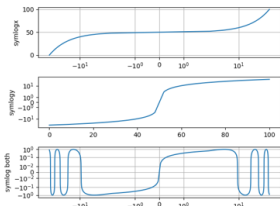
Transoffset



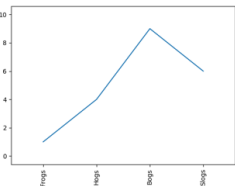
Zorder Demo



Custom scale



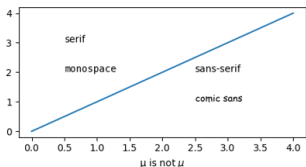
Symlog Demo



Rotating custom tick labels



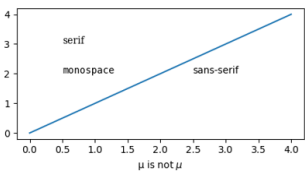
Tool Manager



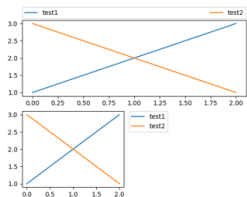
Pgf Fonts



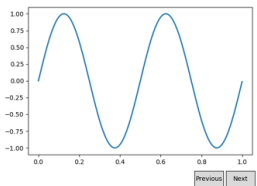
Pgf Preamble



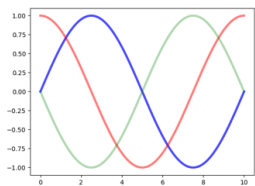
Pgf Texsystem



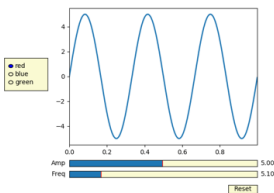
Simple Legend01



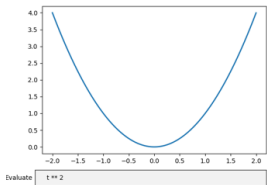
Buttons



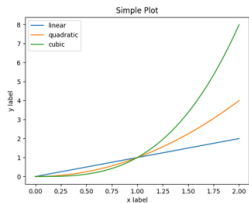
Rectangle Selector



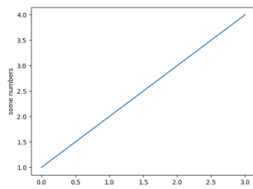
Slider Demo



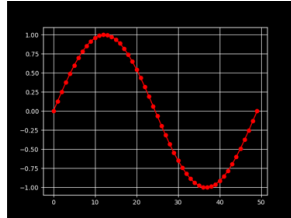
Textbox



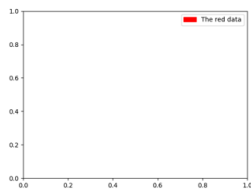
Usage Guide



[Pyplot tutorial](#)



[Customizing
Matplotlib with style
sheets and rcParams](#)



[Legend guide](#)

Hello path effects world!
This is the normal path effect.
Pretty dull, huh?

[Path effects guide](#)

© Copyright 2002 - 2012 John Hunter, Darren Dale, Eric Firing, Michael Droettboom and the Matplotlib development team; 2012 - 2018 The Matplotlib development team.

Last updated on Jul 09, 2019. Created using [Sphinx](#) 1.8.5. Doc version v3.1.1-9-g823365338.