API Reference

matplotlib.pyplot

matplotlib.pyplot.plot

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 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')
  ```

- If *x* and/or *y* are 2D arrays a separate data set will be drawn for every column. If both *x* and *y* are 2D, they must have the same shape. If only one of them is 2D with shape (N, m) the other must have length N and will be used for every data set m.

Example:

```
>>> x = [1, 2, 3]
>>> y = np.array([[1, 2], [3, 4], [5, 6]])
>>> plot(x, y)
```

is equivalent to:

```
>>> for col in range(y.shape[1]):
...     plot(x, y[:, col])
```

- 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 `rcParams["axes.prop_cycle"]` (default: `cycler('color', ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#17becf'])`).

Parameters:

x, yarray-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.

fmtstr, 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.

dataindexable 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:

list of `Line2D`

A list of lines representing the plotted data.

Other Parameters:

scalex, scaleybool, default: True

These parameters determine 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 specify multiple lines with one plot call, the kwargs apply to all those lines. In case the label object is iterable, each element is used as labels for each set of data.

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) array and two offsets from the bottom left corner of the image
`alpha`	scalar or None
`animated`	bool
`antialiased` or aa	bool
`clip_box`	`BboxBase` or None
`clip_on`	bool
`clip_path`	Patch or (Path, Transform) or None
`color` or c	color
`dash_capstyle`	`CapStyle` or {'butt', 'projecting', 'round'}
`dash_joinstyle`	`JoinStyle` or {'miter', 'round', 'bevel'}
`dashes`	sequence of floats (on/off ink in points) or (None, None)

Property	Description
data	(2, N) array or two 1D arrays
drawstyle or ds	{'default', 'steps', 'steps-pre', 'steps-mid', 'steps-post'}, default: 'default'
figure	Figure
fillstyle	{'full', 'left', 'right', 'bottom', 'top', 'none'}
gapcolor	color or None
gid	str
in_layout	bool
label	object
linestyle or ls	{'-', '--', '-.', ':', '', (offset, on-off-seq), ...}
linewidth or lw	float
marker	marker style string, Path or MarkerStyle
markeredgecolor or mec	color
markeredgewidth or mew	float
markerfacecolor or mfc	color
markerfacecoloralt or mfcalt	color
markersize or ms	float
markevery	None or int or (int, int) or slice or list[int] or float or (float, float) or list[bool]
mouseover	bool
path_effects	list of AbstractPathEffect
picker	float or callable[[Artist, Event], tuple[bool, dict]]
pickradius	float

Property	Description
`rasterized`	bool
`sketch_params`	(scale: float, length: float, randomness: float)
`snap`	bool or None
`solid_capstyle`	`CapStyle` or {'butt', 'projecting', 'round'}
`solid_joinstyle`	`JoinStyle` or {'miter', 'round', 'bevel'}
`transform`	unknown
`url`	str
`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

character	description
`','`	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
`'8'`	octagon marker
`'s'`	square marker
`'p'`	pentagon marker
`'P'`	plus (filled) marker
`'*'`	star marker
`'h'`	hexagon1 marker
`'H'`	hexagon2 marker
`'+'`	plus marker
`'x'`	x marker
`'X'`	x (filled) marker

character	description
`'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

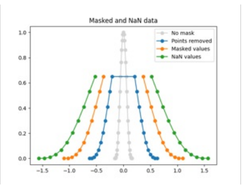
character	color
`'b'`	blue
`'g'`	green
`'r'`	red
`'c'`	cyan
`'m'`	magenta

character	color
`'y'`	yellow
`'k'`	black
`'w'`	white

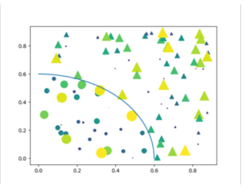
and the `'CN'` colors that index into the default property cycle.

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'`).

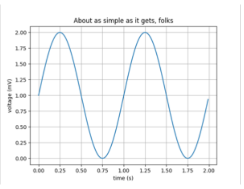
Examples using `matplotlib.pyplot.plot`



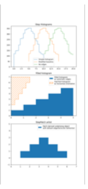
Plotting masked and NaN values



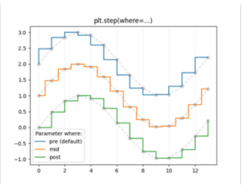
Scatter Masked



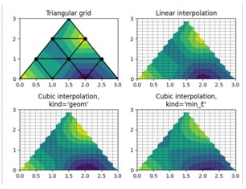
Simple Plot



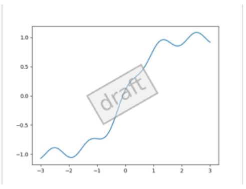
Stairs Demo



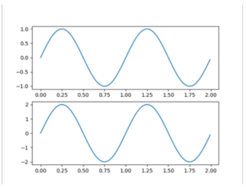
Step Demo



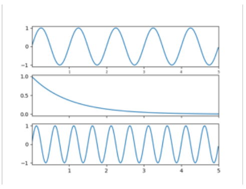
Triinterp Demo



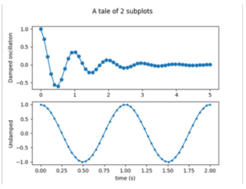
Custom Figure subclasses



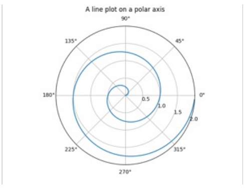
Managing multiple figures in pyplot



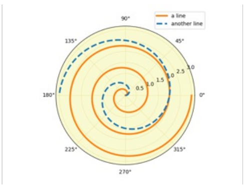
Shared axis



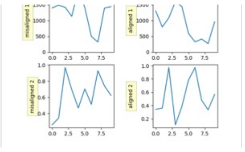
Multiple subplots



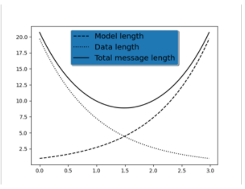
Polar plot



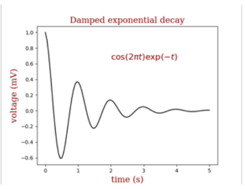
Polar legend



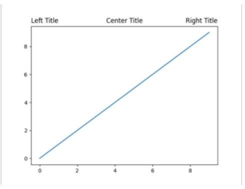
Align y-labels



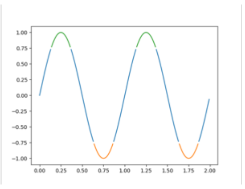
Legend using pre-defined labels



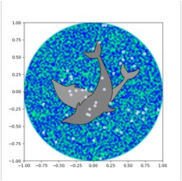
Controlling style of text and labels using a dictionary



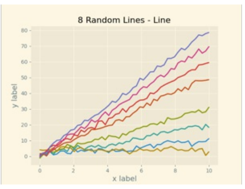
Title positioning



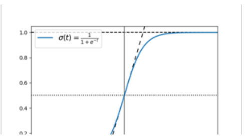
Color by y-value

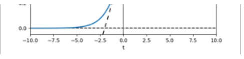


Dolphins

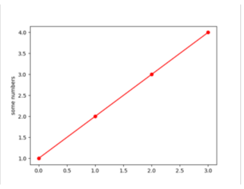


Solarized Light stylesheet

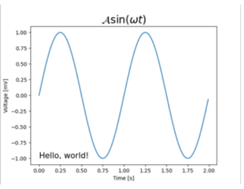




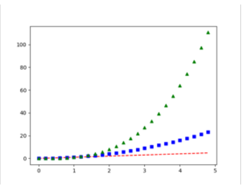
Infinite lines



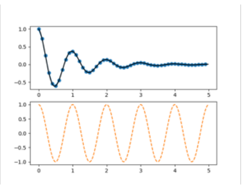
Simple plot



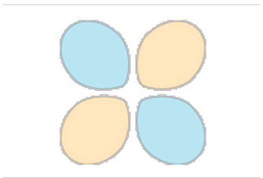
Text and mathtext using pyplot



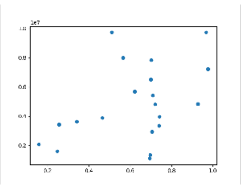
Multiple lines using pyplot



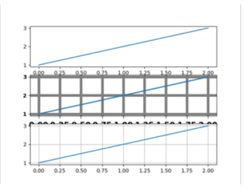
Two subplots using pyplot



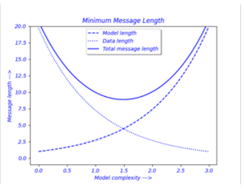
Frame grabbing



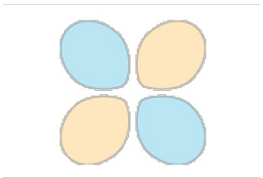
Coords Report



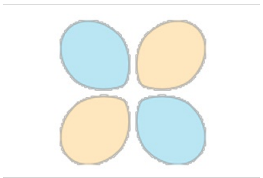
Customize Rc



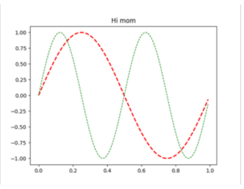
Findobj Demo



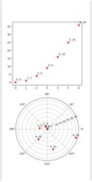
Multipage PDF



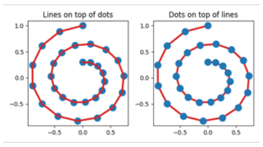
Print Stdout



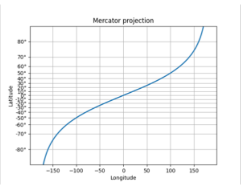
Set and get properties



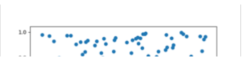
transforms.offset_copy

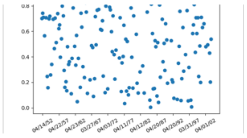


Zorder Demo

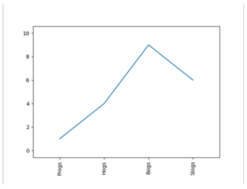


Custom scale

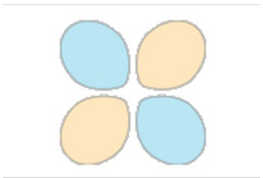




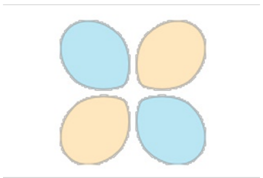
Placing date ticks using recurrence rules



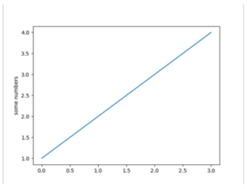
Rotating custom tick labels



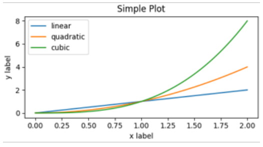
CanvasAgg demo



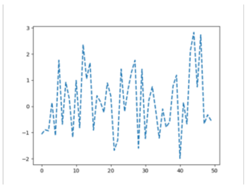
Tool Manager



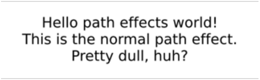
Pyplot tutorial



Quick start guide



Customizing Matplotlib with style sheets and rcParams



Path effects guide

API Reference

`matplotlib.pyplot`

matplotlib.pyplot.subplot

matplotlib.pyplot.subplot

matplotlib.pyplot.subplot(args*, ***kwargs*)[source] #**

Add an Axes to the current figure or retrieve an existing Axes.

This is a wrapper of `Figure.add_subplot` which provides additional behavior when working with the implicit API (see the notes section).

Call signatures:

```
subplot(nrows, ncols, index, **kwargs)
subplot(pos, **kwargs)
subplot(**kwargs)
subplot(ax)
```

Parameters:

***argsint, (int, int, *index*), or `SubplotSpec` , default: (1, 1, 1)**

The position of the subplot described by one of

- Three integers (*nrows*, *ncols*, *index*). The subplot will take the *index* position on a grid with *nrows* rows and *ncols* columns. *index* starts at 1 in the upper left corner and increases to the right. *index* can also be a two-tuple specifying the (*first*, *last*) indices (1-based, and including *last*) of the subplot, e.g., `fig.add_subplot(3, 1, (1, 2))` makes a subplot that spans the upper 2/3 of the figure.
- A 3-digit integer. The digits are interpreted as if given separately as three single-digit integers, i.e. `fig.add_subplot(235)` is the same as `fig.add_subplot(2, 3, 5)` . Note that this can only be used if there are no more than 9 subplots.
- A `SubplotSpec` .

projection{None, 'aitoff', 'hammer', 'lambert', 'mollweide', 'polar', 'rectilinear', str}, optional

The projection type of the subplot (`Axes`). *str* is the name of a custom projection, see `projections` . The default None results in a 'rectilinear' projection.

polarbool, default: False

If True, equivalent to projection='polar'.

sharex, sharey `Axes` , optional

Share the x or y `axis` with sharex and/or sharey. The axis will have the same limits, ticks, and scale as the axis of the shared axes.

labelstr

A label for the returned axes.

Returns:

`Axes`

The Axes of the subplot. The returned Axes can actually be an instance of a subclass, such as projections.polar.PolarAxes for polar projections.

Other Parameters:

****kwargs**

This method also takes the keyword arguments for the returned axes base class; except for the *figure* argument. The keyword arguments for the rectilinear base class Axes can be found in the following table but there might also be other keyword arguments if another projection is used.

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 and two offsets from the bottom left corner of the image
alpha	scalar or None
anchor	(float, float) or {'C', 'SW', 'S', 'SE', 'E', 'NE', ...}
animated	bool
aspect	{'auto', 'equal'} or float
autoscale_on	bool
autoscalex_on	unknown
autoscaley_on	unknown
axes_locator	Callable[[Axes, Renderer], Bbox]
axisbelow	bool or 'line'
box_aspect	float or None
clip_box	BboxBase or None
clip_on	bool
clip_path	Patch or (Path, Transform) or None
facecolor or fc	color
figure	Figure
frame_on	bool

Property	Description
gid	str
in_layout	bool
label	object
mouseover	bool
navigate	bool
navigate_mode	unknown
path_effects	list of AbstractPathEffect
picker	None or bool or float or callable
position	[left, bottom, width, height] or Bbox
prop_cycle	Cycler
rasterization_zorder	float or None
rasterized	bool
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
subplotspec	unknown
title	str
transform	Transform
url	str
visible	bool
xbound	(lower: float, upper: float)
xlabel	str
xlim	(left: float, right: float)
xmargin	float greater than -0.5

Property	Description
xscale	unknown
xticklabels	unknown
xticks	unknown
ybound	(lower: float, upper: float)
ylabel	str
ylim	(bottom: float, top: float)
ymargin	float greater than -0.5
yscale	unknown
yticklabels	unknown
yticks	unknown
zorder	float

See also

Figure.add_subplot

pyplot.subplots

pyplot.axes

Figure.subplots

Notes

Creating a new Axes will delete any preexisting Axes 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)
```

If you do not want this behavior, use the `Figure.add_subplot` method or the `pyplot.axes` function instead.

If no *kwargs* are passed and there exists an Axes in the location specified by *args* then that Axes will be returned rather than a new Axes being created.

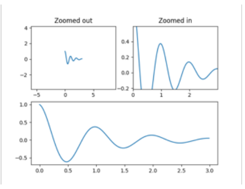
If *kwargs* are passed and there exists an Axes in the location specified by *args*, the projection type is the same, and the *kwargs* match with the existing Axes, then the existing Axes is returned. Otherwise a new Axes is created with the specified parameters. We save a reference to the *kwargs* which we use for this comparison. If any of the values in *kwargs* are mutable we will not

We save a reference to the *kwargs* which we use for this comparison. If any of the values in *kwargs* are mutable we will not detect the case where they are mutated. In these cases we suggest using `Figure.add_subplot` and the explicit Axes API rather than the implicit pyplot API.

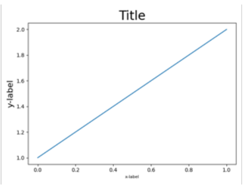
Examples

```
plt.subplot(221)

# equivalent but more general
ax1 = plt.subplot(2, 2, 1)

# add a subplot with no frame
ax2 = plt.subplot(222, frameon=False)

# add a polar subplot
plt.subplot(223, projection='polar')

# add a red subplot that shares the x-axis with ax1
plt.subplot(224, sharex=ax1, facecolor='red')

# delete ax2 from the figure
plt.delaxes(ax2)

# add ax2 to the figure again
plt.subplot(ax2)

# make the first axes "current" again
plt.subplot(221)
```

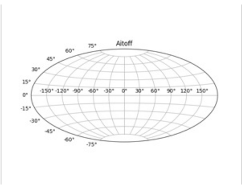
Examples using `matplotlib.pyplot.subplot`



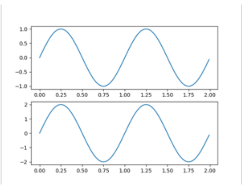
Controlling view limits using margins and sticky_edges



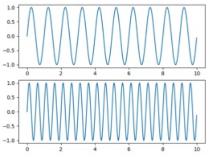
Resizing axes with tight layout



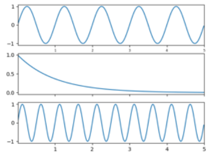
Geographic Projections



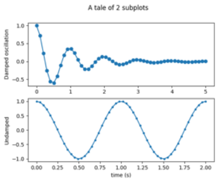
Managing multiple figures in pyplot



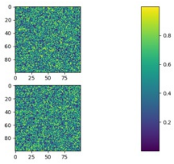
Sharing axis limits and views



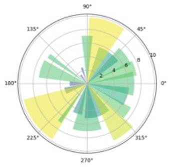
Shared axis



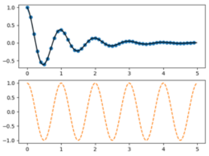
Multiple subplots



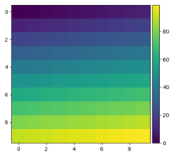
Subplots spacings and margins



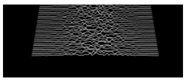
Bar chart on polar axis



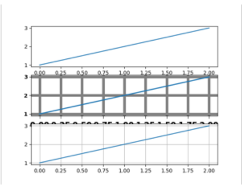
Two subplots using pyplot



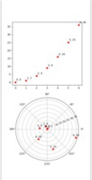
Simple Colorbar



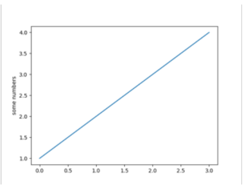
MATPLOTLIB UNCHAINED



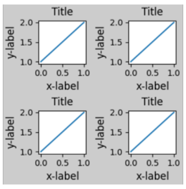
Customize Rc



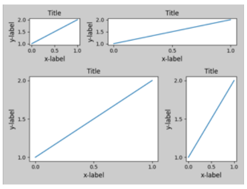
transforms.offset_copy



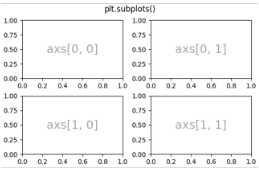
Pyplot tutorial



Constrained Layout Guide



Tight Layout guide



Arranging multiple Axes in a Figure