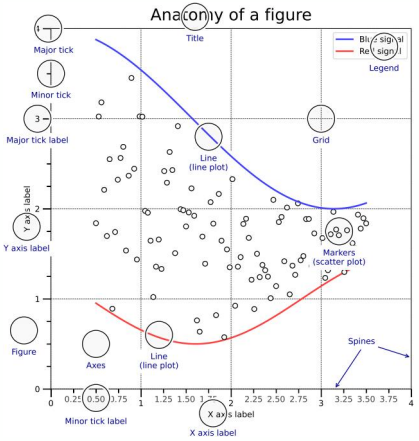
matplotlib Cheat sheet

Version 3.5.0

Quick start

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


X = np.linspace(0, 2*np.pi, 100)
Y = np.cos(X)

fig, ax = plt.subplots()
ax.plot(X, Y, color='green')

fig.savefig("figure.pdf")
fig.show()
```

Anatomy of a figure



Subplots layout

`subplot[s](rows,cols,…)`
`fig, axs = plt.subplots(3, 3)`

`G = gridspec(rows,cols,…)`
`ax = G[0,:]`

`ax.inset_axes(extent)`

`d=make_axes_locatable(ax)`
`ax = d.new_horizontal('10%')`

Getting help

- matplotlib.org
- github.com/matplotlib/matplotlib/issues
- discourse.matplotlib.org
- stackoverflow.com/questions/tagged/matplotlib
- gitter.im/matplotlib
- twitter.com/matplotlib
- Matplotlib users mailing list

Basic plots

`plot([X],Y,[fmt],…)`
X, Y, fmt, color, marker, linestyle

`scatter(X,Y,…)`
X, Y, [s]izes, [c]olors, marker, cmap

`bar[h](x,height,…)`
x, height, width, bottom, align, color

`imshow(Z,…)`
Z, cmap, interpolation, extent, origin

`contour[f]([X],[Y],Z,…)`
X, Y, Z, levels, colors, extent, origin

`pcolormesh([X],[Y],Z,…)`
X, Y, Z, vmin, vmax, cmap

`quiver([X],[Y],U,V,…)`
X, Y, U, V, C, units, angles

`pie(X,…)`
Z, explode, labels, colors, radius

`text(x,y,text,…)`
x, y, text, va, ha, size, weight, transform

`fill[_between][x](…)`
X, Y1, Y2, color, where

Advanced plots

`step(X,Y,[fmt],…)`
X, Y, fmt, color, marker, where

`boxplot(X,…)`
X, notch, sym, bootstrap, widths

`errorbar(X,Y,xerr,yerr,…)`
X, Y, xerr, yerr, fmt

`hist(X, bins, …)`
X, bins, range, density, weights

`violinplot(D,…)`
D, positions, widths, vert

`barbs([X],[Y], U, V, …)`
X, Y, U, V, C, length, pivot, sizes

`eventplot(positions,…)`
positions, orientation, lineoffsets

`hexbin(X,Y,C,…)`
X, Y, C, gridsize, bins

Scales

`ax.set_[xy]scale(scale,…)`

linear
any values

log
values > 0

symlog
any values

logit
0 < values < 1

Projections

`subplot(…,projection=p)`

p='polar'

p='3d'

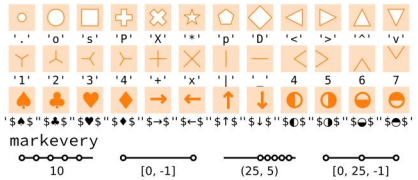
p=Orthographic()
from cartopy.crs import Cartographic

Lines

`linestyle or ls`

`capstyle or dash_capstyle`

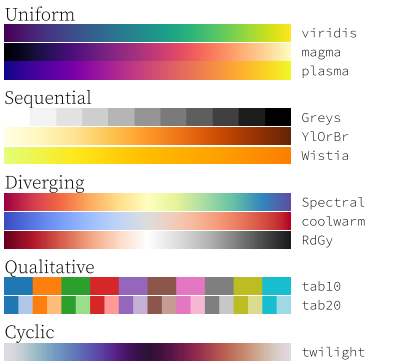
Markers



Colors

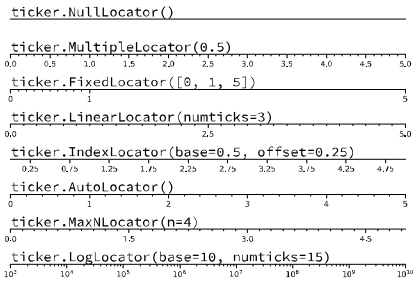
Colormaps

`plt.get_cmap(name)`



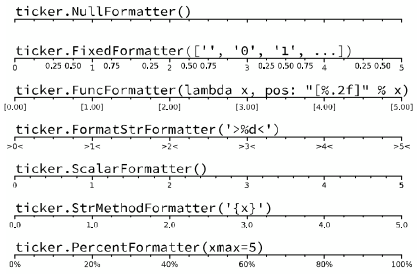
Tick locators

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_locator(locator)
```



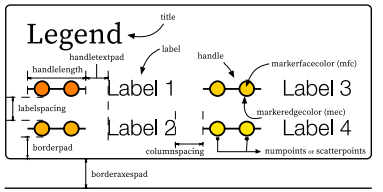
Tick formatters

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_formatter(formatter)
```

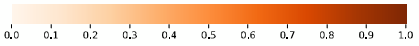


Ornaments

`ax.legend(…)`
handles, labels, loc, title, frameon



`ax.colorbar(…)`
mappable, ax, cax, orientation



`ax.annotate(…)`
text, xy, xytext, xycoords, textcoords, arrowprops

Event handling

```
fig, ax = plt.subplots()
def on_click(event):
  print(event)
fig.canvas.mpl_connect(
  'button_press_event', on_click)
```

Animation

```
import matplotlib.animation as mpla

T = np.linspace(0, 2*np.pi, 100)
S = np.sin(T)
line, = plt.plot(T, S)
def animate(i):
    line.set_ydata(np.sin(T+i/50))
anim = mpla.FuncAnimation(
    plt.gcf(), animate, interval=5)
plt.show()
```

Styles

`plt.style.use(style)`

Quick reminder

```
ax.grid()
ax.patch.set_alpha(0)
ax.set_[xy]lim(vmin, vmax)
ax.set_[xy]label(label)
ax.set_[xy]ticks(list)
ax.set_[xy]ticklabels(list)
ax.set_[sup]title(title)
ax.tick_params(width=10, …)
ax.set_axis_[on|off]()

fig.tight_layout()
plt.gcf(), plt.gca()
mpl.rc('axes', linewidth=1, …)
fig.patch.set_alpha(0)
text=r'$\frac{-e^{i\pi}}{2^n}$'
```

Keyboard shortcuts

ctrl + s	Save	ctrl + w	Close plot
r	Reset view	f	Fullscreen 0/1
f	View forward	b	View back
p	Pan view	o	Zoom to rect
x	X pan/zoom	y	Y pan/zoom
g	Minor grid 0/1	G	Major grid 0/1
l	X axis log/linear	L	Y axis log/linear

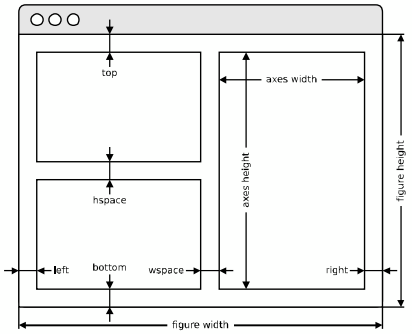
Ten simple rules

1. Know Your Audience
2. Identify Your Message
3. Adapt the Figure
4. Captions Are Not Optional
5. Do Not Trust the Defaults
6. Use Color Effectively
7. Do Not Mislead the Reader
8. Avoid "Chartjunk"
9. Message Trumps Beauty
10. Get the Right Tool

Axes adjustments

API

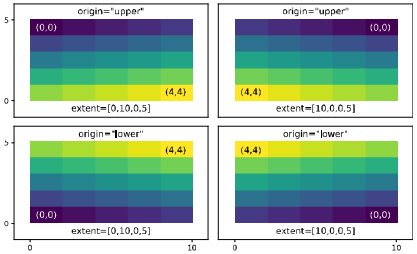
plt.**subplots_adjust**(...)



Extent & origin

API

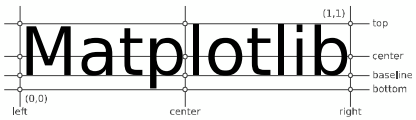
ax.**imshow**(extent=..., origin=...)



Text alignments

API

ax.**text**(..., ha=... , va=..., ...)



Text parameters

API

ax.**text**(..., family=..., size=..., weight=...)
ax.**text**(..., fontproperties=...)

The quick brown fox — xx-large (1.73)
The quick brown fox — x-large (1.44)
The quick brown fox — large (1.20)
The quick brown fox — medium (1.00)
The quick brown fox — small (0.83)
The quick brown fox — x-small (0.69)
The quick brown fox — xx-small (0.58)

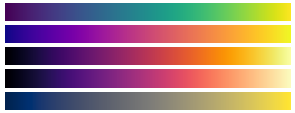
The quick brown fox jumps over the lazy dog — black (900)
The quick brown fox jumps over the lazy dog — bold (700)
The quick brown fox jumps over the lazy dog — semibold (600)
The quick brown fox jumps over the lazy dog — normal (400)
The quick brown fox jumps over the lazy dog — ultralight (100)

The quick brown fox jumps over the lazy dog — monospace
The quick brown fox jumps over the lazy dog — serif
The quick brown fox jumps over the lazy dog — sans
The quick brown fox jumps over the lazy dog — cursive

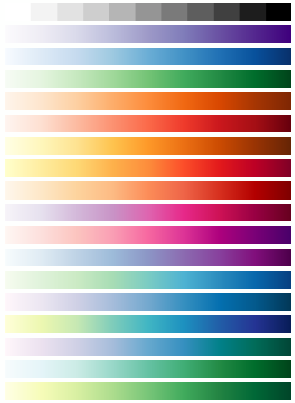
The quick brown fox jumps over the lazy dog — italic
The quick brown fox jumps over the lazy dog — normal

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG — small-caps
The quick brown fox jumps over the lazy dog — normal

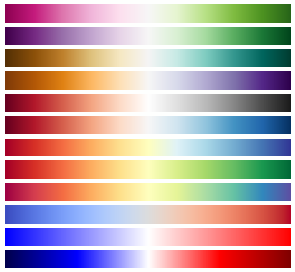
Uniform colormaps



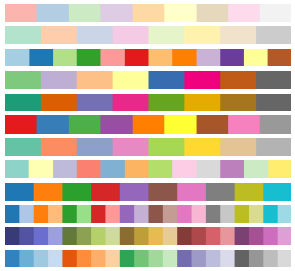
Sequential colormaps



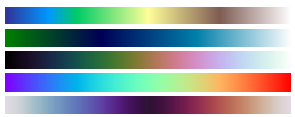
Diverging colormaps



Qualitative colormaps



Miscellaneous colormaps

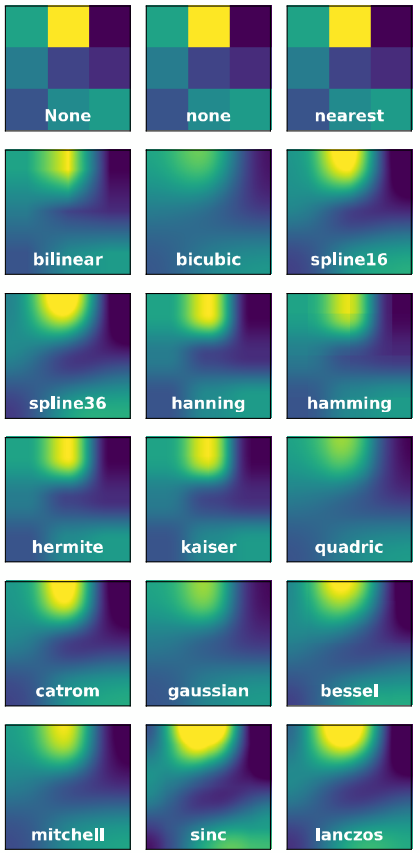


Color names

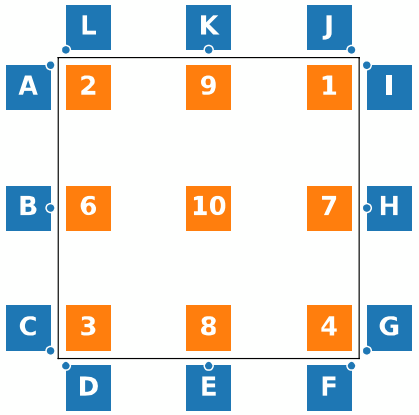
API

Image interpolation

API



Legend placement



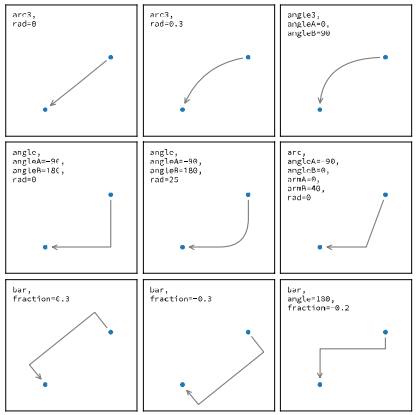
ax.**legend**(loc="string", bbox_to_anchor=(x,y))

2: upper left — 9: upper center — 1: upper right
6: center left — 10: center — 7: center right
3: lower left — 8: lower center — 4: lower right

A: upper right / `(-0.1,0.9)` — B: center right / `(-0.1,0.5)`
C: lower right / `(-0.1,0.1)` — D: upper left / `(0.1,-0.1)`
E: upper center / `(0.5,-0.1)` — F: upper right / `(0.9,-0.1)`
G: lower left / `(1.1,0.1)` — H: center left / `(1.1,0.5)`
I: upper left / `(1.1,0.9)` — J: lower right / `(0.9,1.1)`
K: lower center / `(0.5,1.1)` — L: lower left / `(0.1,1.1)`

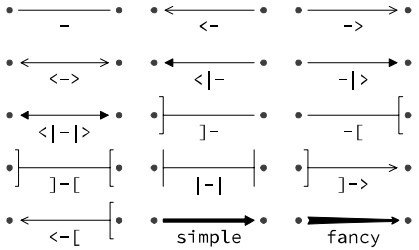
Annotation connection styles

API



Annotation arrow styles

API



How do I ...

... resize a figure?
→ fig.set_size_inches(w, h)
... save a figure?
→ fig.savefig("figure.pdf")
... save a transparent figure?
→ fig.savefig("figure.pdf", transparent=True)
... clear a figure/an axes?
→ fig.clear() → ax.clear()
... close all figures?
→ plt.close("all")
... remove ticks?
→ ax.set_[xy]ticks([])
... remove tick labels ?
→ ax.set_[xy]ticklabels([])
... rotate tick labels ?
→ ax.set_[xy]ticks(rotation=90)
... hide top spine?
→ ax.spines['top'].set_visible(False)
... hide legend border?
→ ax.legend(frameon=False)
... show error as shaded region?
→ ax.fill_between(X, Y+error, Y-error)
... draw a rectangle?
→ ax.add_patch(plt.Rectangle((0, 0), 1, 1)
... draw a vertical line?
→ ax.axvline(x=0.5)
... draw outside frame?
→ ax.plot(..., clip_on=False)
... use transparency?
→ ax.plot(..., alpha=0.25)
... convert an RGB image into a gray image?
→ gray = 0.2989*R + 0.5870*G + 0.1140*B
... set figure background color?
→ fig.patch.set_facecolor("grey")
... get a reversed colormap?
→ plt.get_cmap("viridis_r")
... get a discrete colormap?
→ plt.get_cmap("viridis", 10)
... show a figure for one second?
→ fig.show(block=False), time.sleep(1)

Performance tips

```
scatter(X, Y)                          slow
plot(X, Y, marker="o", ls="")          fast

for i in range(n): plot(X[i])          slow
plot(sum([x+[None] for x in X],[]))    fast

cla(), imshow(…), canvas.draw()        slow
im.set_data(…), canvas.draw()          fast
```

Beyond Matplotlib

Seaborn: Statistical Data Visualization
Cartopy: Geospatial Data Processing
yt: Volumetric data Visualization
mpld3: Bringing Matplotlib to the browser
Datashader: Large data processing pipeline
plotnine: A Grammar of Graphics for Python

NUMFOCUS
OPEN CODE = BETTER SCIENCE

Matplotlib for beginners

Matplotlib is a library for making 2D plots in Python. It is designed with the philosophy that you should be able to create simple plots with just a few commands:

1 Initialize

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

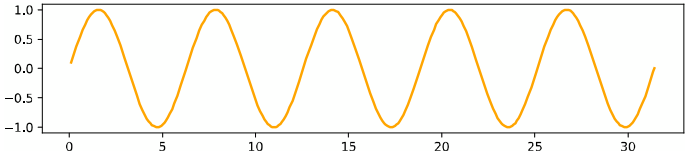
2 Prepare

```
X = np.linspace(0, 4*np.pi, 1000)
Y = np.sin(X)
```

3 Render

```
fig, ax = plt.subplots()
ax.plot(X, Y)
fig.show()
```

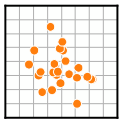
4 Observe



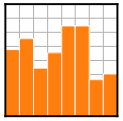
Choose

Matplotlib offers several kind of plots (see Gallery):

```
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
ax.scatter(X, Y)
```



```
X = np.arange(10)
Y = np.random.uniform(1, 10, 10)
ax.bar(X, Y)
```

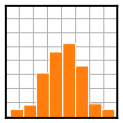


```
Z = np.random.uniform(0, 1, (8,8))

ax.imshow(Z)
```

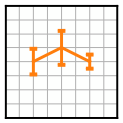
```
Z = np.random.uniform(0, 1, (8,8))

ax.contourf(Z)
```

```
Z = np.random.uniform(0, 1, 4)

ax.pie(Z)
```

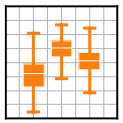
```
Z = np.random.normal(0, 1, 100)

ax.hist(Z)
```



```
X = np.arange(5)
Y = np.random.uniform(0, 1, 5)
ax.errorbar(X, Y, Y/4)
```



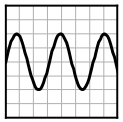
```
Z = np.random.normal(0, 1, (100,3))

ax.boxplot(Z)
```



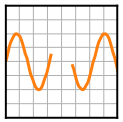
Tweak

You can modify pretty much anything in a plot, including limits, colors, markers, line width and styles, ticks and ticks labels, titles, etc.

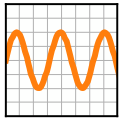
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, color="black")
```



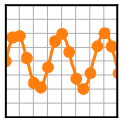
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linestyle="--")
```



```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linewidth=5)
```



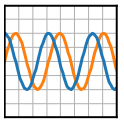
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, marker="o")
```



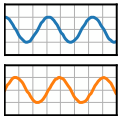
Organize

You can plot several data on the the same figure, but you can also split a figure in several subplots (named *Axes*):

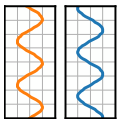
```
X = np.linspace(0, 10, 100)
Y1, Y2 = np.sin(X), np.cos(X)
ax.plot(X, Y1, X, Y2)
```



```
fig, (ax1, ax2) = plt.subplots((2,1))
ax1.plot(X, Y1, color="C1")
ax2.plot(X, Y2, color="C0")
```

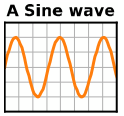


```
fig, (ax1, ax2) = plt.subplots((1,2))
ax1.plot(Y1, X, color="C1")
ax2.plot(Y2, X, color="C0")
```

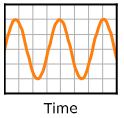


Label (everything)

```
ax.plot(X, Y)
fig.suptitle(None)
ax.set_title("A Sine wave")
```




```
ax.plot(X, Y)
ax.set_ylabel(None)
ax.set_xlabel("Time")
```



Explore

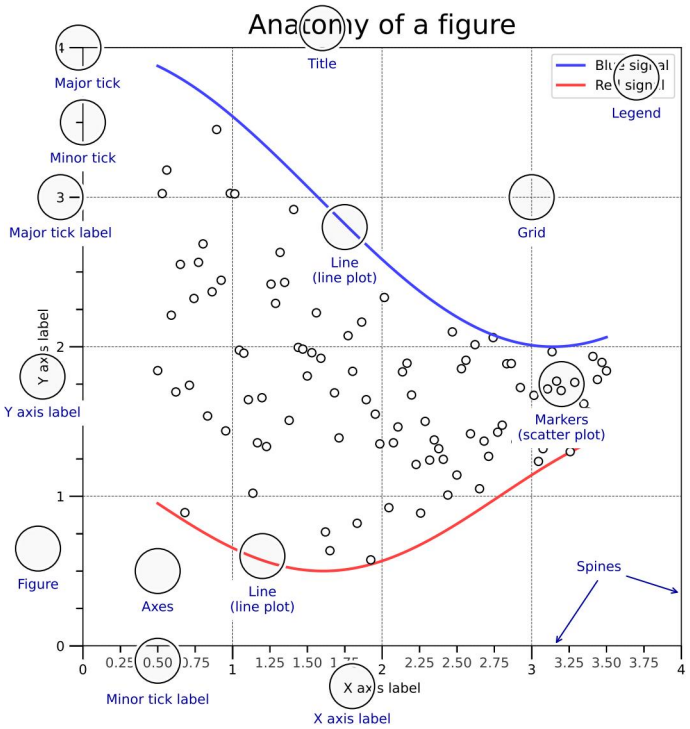
Figures are shown with a graphical user interface that allows to zoom and pan the figure, to navigate between the different views and to show the value under the mouse.

Save (bitmap or vector format)

```
fig.savefig("my-first-figure.png", dpi=300)
fig.savefig("my-first-figure.pdf")
```

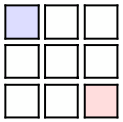
Matplotlib for intermediate users

A matplotlib figure is composed of a hierarchy of elements that forms the actual figure. Each element can be modified.

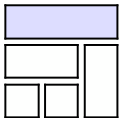


Figure, axes & spines

```
fig, axs = plt.subplots((3,3))
axs[0,0].set_facecolor("#ddddff")
axs[2,2].set_facecolor("#ffffdd")
```



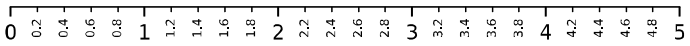
```
gs = fig.add_gridspec(3, 3)
ax = fig.add_subplot(gs[0, :])
ax.set_facecolor("#ddddff")
```



```
fig, ax = plt.subplots()
ax.spines["top"].set_color("None")
ax.spines["right"].set_color("None")
```

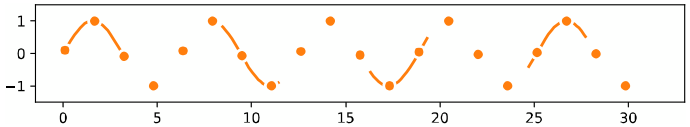
Ticks & labels

```
from mpl.ticker import MultipleLocator as ML
from mpl.ticker import ScalarFormatter as SF
ax.xaxis.set_minor_locator(ML(0.2))
ax.xaxis.set_minor_formatter(SF())
ax.tick_params(axis='x',which='minor',rotation=90)
```



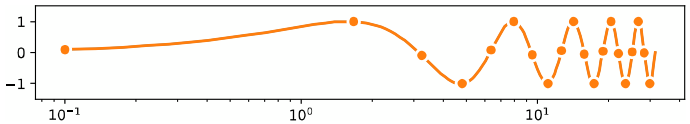
Lines & markers

```
X = np.linspace(0.1, 10*np.pi, 1000)
Y = np.sin(X)
ax.plot(X, Y, "C1o:", markevery=25, mec="1.0")
```



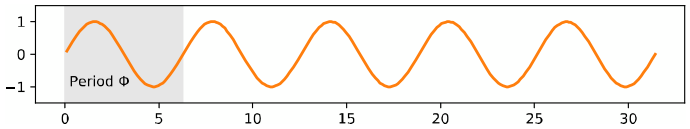
Scales & projections

```
fig, ax = plt.subplots()
ax.set_xscale("log")
ax.plot(X, Y, "C1o-", markevery=25, mec="1.0")
```



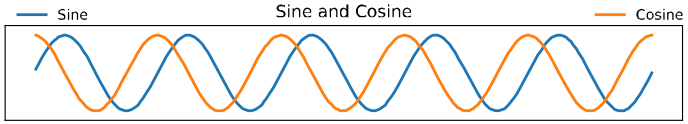
Text & ornaments

```
ax.fill_betweenx([-1,1],[0],[2*np.pi])
ax.text(0, -1, r" Period $\Phi$")
```



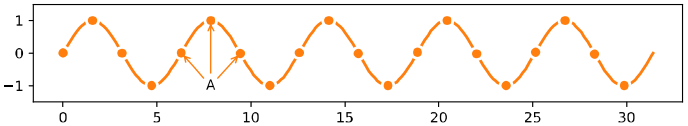
Legend

```
ax.plot(X, np.sin(X), "C0", label="Sine")
ax.plot(X, np.cos(X), "C1", label="Cosine")
ax.legend(bbox_to_anchor=(0,1,1,.1),ncol=2,
          mode="expand",  loc="lower left")
```



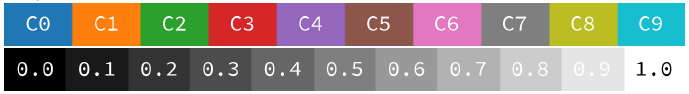
Annotation

```
ax.annotate("A", (X[250],Y[250]),(X[250],-1),
  ha="center", va="center",arrowprops =
  {"arrowstyle" : "->", "color": "C1"})
```



Colors

Any color can be used, but Matplotlib offers sets of colors:



Size & DPI

Consider a square figure to be included in a two-columns A4 paper with 2cm margins on each side and a column separation of 1cm. The width of a figure is (21 - 2*2 - 1)/2 = 8cm. One inch being 2.54cm, figure size should be 3.15×3.15 in.

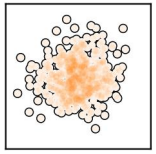
```
fig = plt.figure(figsize=(3.15,3.15), dpi=50)
plt.savefig("figure.pdf", dpi=600)
```

Matplotlib tips & tricks

Transparency

Scatter plots can be enhanced by using transparency (alpha) in order to show area with higher density. Multiple scatter plots can be used to delineate a frontier.

```
X = np.random.normal(-1, 1, 500)
Y = np.random.normal(-1, 1, 500)
ax.scatter(X, Y, 50, "0.0", lw=2) # optional
ax.scatter(X, Y, 50, "1.0", lw=0) # optional
ax.scatter(X, Y, 40, "C1",  lw=0, alpha=0.1)
```



Rasterization

If your figure has many graphical elements, such as a huge scatter, you can rasterize them to save memory and keep other elements in vector format.

```
X = np.random.normal(-1, 1, 10_000)
Y = np.random.normal(-1, 1, 10_000)
ax.scatter(X, Y, rasterized=True)
fig.savefig("rasterized-figure.pdf", dpi=600)
```

Offline rendering

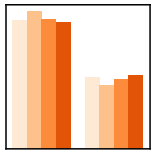
Use the Agg backend to render a figure directly in an array.

```
from matplotlib.backends.backend_agg import FigureCanvas
canvas = FigureCanvas(Figure()))
... # draw som stuff
canvas.draw()
Z = np.array(canvas.renderer.buffer_rgba())
```

Range of continuous colors

You can use colormap to pick from a range of continuous colors.

```
X = np.random.randn(1000, 4)
cmap = plt.get_cmap("Oranges")
colors = cmap([0.2, 0.4, 0.6, 0.8])

ax.hist(X, 2, histtype='bar', color=colors)
```



Text outline

Use text outline to make text more visible.

```
import matplotlib.patheffects as fx
text = ax.text(0.5, 0.1, "Label")
text.set_path_effects([
  fx.Stroke(linewidth=3, foreground='1.0'),
  fx.Normal()])
```

Multiline plot

You can plot several lines at once using None as separator.

```
X,Y = [], []
for x in np.linspace(0, 10*np.pi, 100):
  X.extend([x, x, None]), Y.extend([0, sin(x), None])
ax.plot(X, Y, "black")
```

Dotted lines

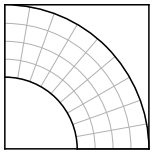
To have rounded dotted lines, use a custom `linestyle` and modify `dash_capstyle`.

```
ax.plot([0,1], [0,0], "C1",
        linestyle = (0, (0.01, 1)), dash_capstyle="round")
ax.plot([0,1], [1,1], "C1",
        linestyle = (0, (0.01, 2)), dash_capstyle="round")
```

Combining axes

You can use overlaid axes with different projections.

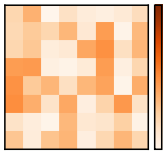
```
ax1 = fig.add_axes([0,0,1,1],
                   label="cartesian")
ax2 = fig.add_axes([0,0,1,1],
                   label="polar",
                   projection="polar")
```



Colorbar adjustment

You can adjust a colorbar's size when adding it.

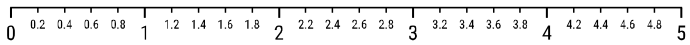
```
im = ax.imshow(Z)

cb = plt.colorbar(im,
        fraction=0.046, pad=0.04)
cb.set_ticks([])
```



Taking advantage of typography

You can use a condensed font such as Roboto Condensed to save space on tick labels.

```
for tick in ax.get_xticklabels(which='both'):
    tick.set_fontname("Roboto Condensed")
```



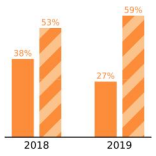
Getting rid of margins

Once your figure is finished, you can call `tight_layout()` to remove white margins. If there are remaining margins, you can use the `pdfcrop` utility (comes with TeX live).

Hatching

You can achieve a nice visual effect with thick hatch patterns.

```
cmap = plt.get_cmap("Oranges")
plt.rcParams['hatch.color'] = cmap(0.2)
plt.rcParams['hatch.linewidth'] = 8
ax.bar(X, Y, color=cmap(0.6), hatch="/" )
```



Read the documentation

Matplotlib comes with an extensive documentation explaining the details of each command and is generally accompanied by examples. Together with the huge online gallery, this documentation is a gold-mine.