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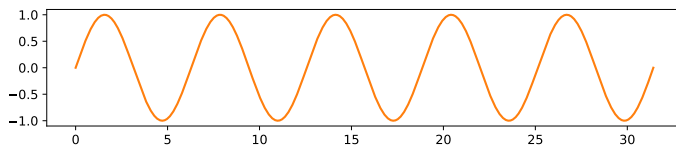
```
import numpy as np
import matplotlib.pyplot as plt
```

2 Prepare

```
X = np.linspace(0, 10*np.pi, 1000)
Y = np.sin(X)
```

3 Render

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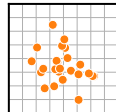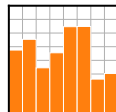
4 Observe

Choose

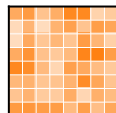Matplotlib offers several kind of plots (see Gallery):

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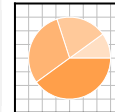
```
Z = np.random.uniform(0, 1, (8, 8))

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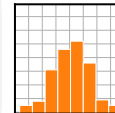
```
Z = np.random.uniform(0, 1, (8, 8))

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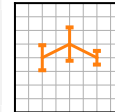
```
Z = np.random.uniform(0, 1, 4)

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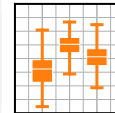
```
Z = np.random.normal(0, 1, 100)

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)
```
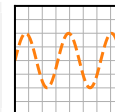
Tweak

You can modify pretty much anything in a plot, including limits, colors, markers, line width and styles, ticks and ticks labels, titles, etc.
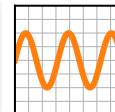
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, color="black")
```
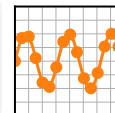
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linestyle="--")
```

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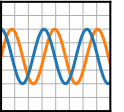
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, marker="o")
```
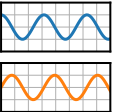
Organize

You can plot several data on the same figure, but you can also split a figure in several subplots (named *Axes*):
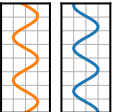
```
X = np.linspace(0, 10, 100)
Y1, Y2 = np.sin(X), np.cos(X)
ax.plot(X, Y1, X, Y2)
```

```
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

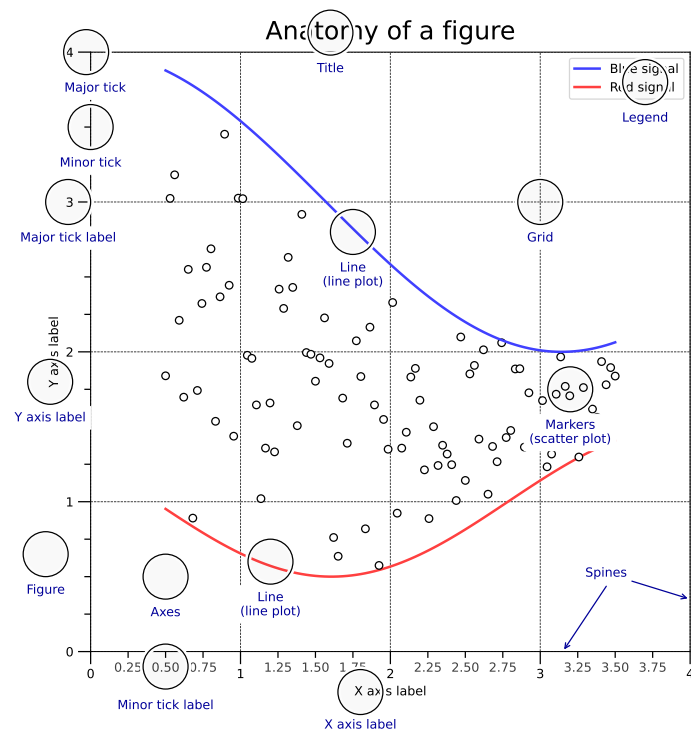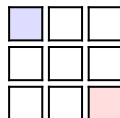A matplotlib figure is composed of a hierarchy of elements that forms the actual figure. Each element can be modified.
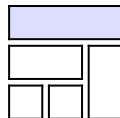

Anatomy of a figure

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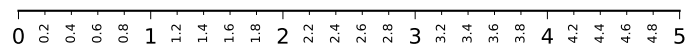
```
fig, ax = plt.subplots()
ax.spines["top"].set_color("None")
ax.spines["right"].set_color("None")
```
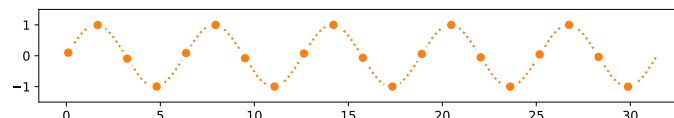
Ticks & labels

```
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)
```
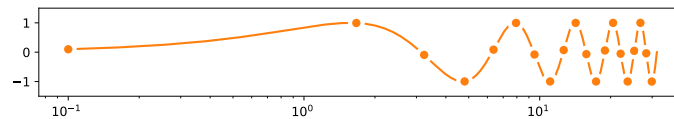
0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2 2.2 2.4 2.6 2.8 3 3.2 3.4 3.6 3.8 4 4.2 4.4 4.6 4.8 5

Lines & markers

```
X = np.linspace(0.1, 10*np.pi, 1000)
Y = np.sin(X)
ax.plot(X, Y, "C1o:", markevery=50, mec="1.0")
```
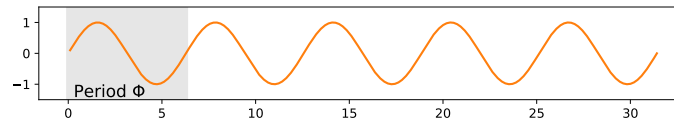

Scales & projections

```
fig, ax = plt.subplots()
ax.set_xscale("log")
ax.plot(X, Y, "C1o-", markevery=50, mec="1.0")
```
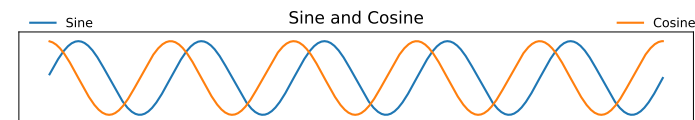

Text & ornaments

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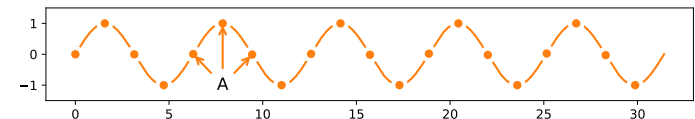

Legend

```
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:

C0	C1	C2	C3	C4	C5	C6	C7	C8	C9

0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0

Size & DPI

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

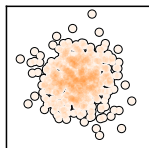Scatter plots can be enhanced by using transparency (alpha) in order to show area with higher density. Multiple scatter plots can be used to delineate a frontier.

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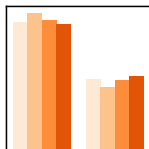
```
from matplotlib.backends.backend_agg import FigureCanvas
canvas = FigureCanvas(Figure())
... # draw some stuff
canvas.draw()
Z = np.array(canvas.renderer.buffer_rgba())
```

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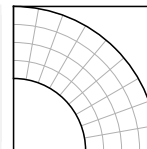
```
ax.plot([0, 1], [0, 0], "C1",
        linestyle=(0, (0.01, 1)), dash_capstyle="round")
ax.plot([0, 1], [1, 1], "C1",
        linestyle=(0, (0.01, 2)), dash_capstyle="round")
```

Combining axes

You can use overlaid axes with different projections.
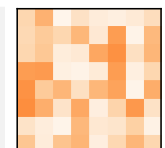
```
ax1 = fig.add_axes([0, 0, 1, 1],
                   label="cartesian")
ax2 = fig.add_axes([0, 0, 1, 1],
                   label="polar",
                   projection="polar")
```

Colorbar adjustment

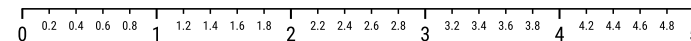You can adjust a colorbar's size when adding it.

```
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")
```
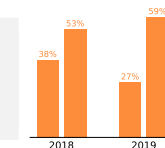
Getting rid of margins

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.