

Python for scientific research

Pattern matching and text manipulation

Bram Kuijper

University of Exeter, Penryn Campus, UK

March 27, 2019



Researcher
Development

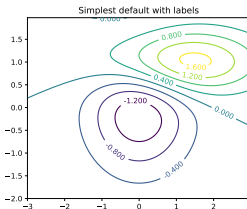
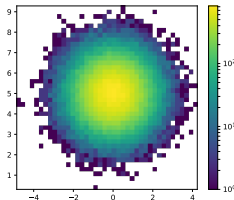
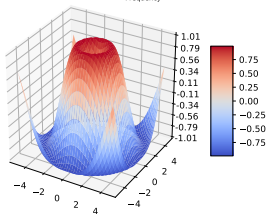
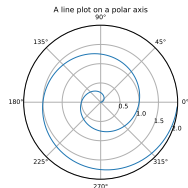
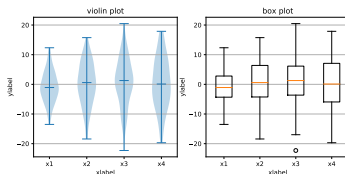
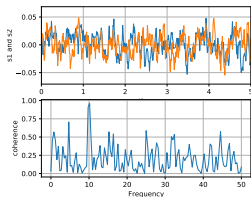


What we've done so far

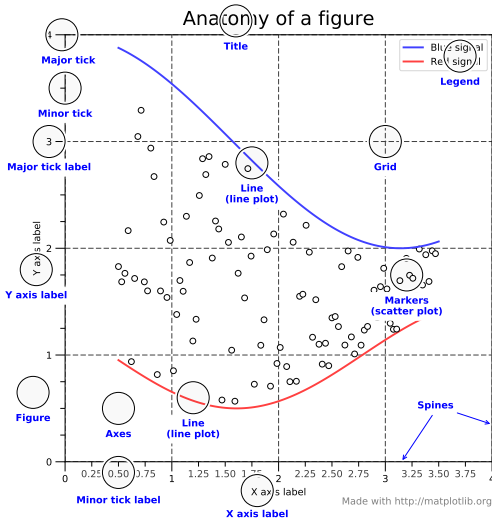
- 1 Declare variables using built-in data types and execute operations on them
- 2 Use flow control commands to dictate the order in which commands are run and when
- 3 Encapsulate programs into reusable functions, modules and packages
- 4 Use string manipulation and regex to work with textual data
- 5 Interact with the file system
- 6 Number crunching using NumPy/SciPy
- 7 **Next:** Introducing Matplotlib, Python's plotting library

Introduction

- Matplotlib is a 2D and 3D plotting library that produces publication-ready scientific figures in most formats (e.g. png, eps, svg)

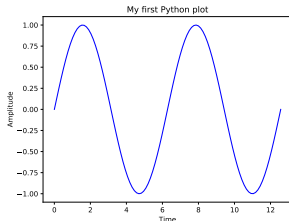


Anatomy of a figure



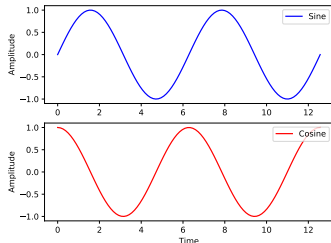
My first plot

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Generate sinusoidal data
5 x = np.linspace(0, 4*np.pi, 100)
6 y = np.sin(x)
7
8 # Plot sinusoidal curve
9 plt.plot(x, y, color="blue")
10 plt.xlabel("Time")
11 plt.ylabel("Amplitude")
12 plt.title("My first Python plot")
```



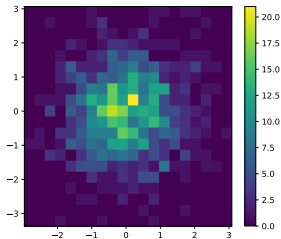
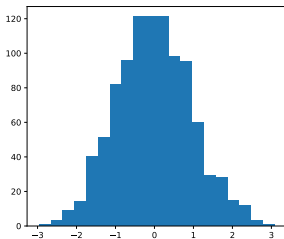
Subplots

```
1 plt.subplot(2, 1, 1) # 2 rows, 1 column, first plot
2 plt.plot(x, np.sin(x), color="b", label="Sine")
3 plt.ylabel("Amplitude")
4 plt.legend(loc="upper right")
5
6 plt.subplot(2, 1, 2) # 2 rows, 1 column, second plot
7 plt.plot(x, np.cos(x), color="r", label="Cosine")
8 plt.xlabel("Time")
9 plt.ylabel("Amplitude")
10 plt.legend(loc="upper right")
```



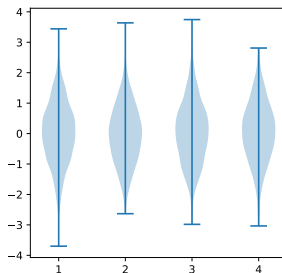
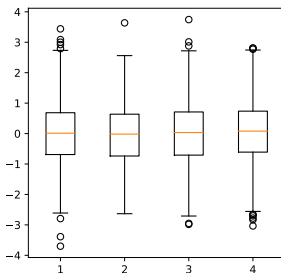
Histograms

```
1 # Generate 1000 normally distributed numbers
2 x1 = np.random.randn(1000)
3 x2 = np.random.randn(1000)
4
5 # 1D histogram (x1)
6 plt.subplot(1, 2, 1)
7 plt.hist(x1, bins=20)
8
9 # 2D histogram (x1 vs x2)
10 plt.subplot(1, 2, 2)
11 plt.hist2d(x1, x2, bins=20)
12 plt.colorbar()
```



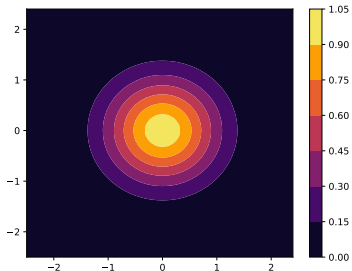
Boxplots and violin plots

```
1 # Generate four sets of random numbers
2 data = [np.random.randn(1000) for i in range(4)]
3
4 # Boxplot
5 plt.subplot(1, 2, 1)
6 plt.boxplot(data)
7
8 # Violin plot
9 plt.subplot(1, 2, 2)
10 plt.violinplot(data)
```



Contour plot

```
1 # import colormaps
2 from matplotlib import cm
3
4 # Generate some x, y, z data
5 x = np.arange(-2.5, 2.5, 0.1)
6 y = np.arange(-2.5, 2.5, 0.1)
7 x, y = np.meshgrid(x, y)
8 z = np.exp(-x**2 - y**2)
9
10 # Contour plot
11 plt.contourf(x, y, z, cmap=cm.inferno)
12 plt.colorbar()
```



- Typical structure of 3D data

column_x	column_y	column_z
0.897238	100000	4.06333241929155
0.916328	500	5.1788870872331
0.954509	100909	5.1788870872331
...

- Data needs to be transformed to a pivot table before plotting. Example:

```
1 print(z)
2 # [[3.72665317e-06 6.08307642e-06 9.73288695e-06 ... 1.52642052e-05
3 #    9.73288695e-06 6.08307642e-06]
4 #    ...
5 #    [6.08307642e-06 9.92950431e-06 1.58871492e-05 ... 2.49160097e-05
6 #    1.58871492e-05 9.92950431e-06]]
```

Working with data and matplotlib

- Typical structure of 3D data

column_x	column_y	column_z
0.897238	100000	4.06333241929155
0.916328	500	5.1788870872331
0.954509	100909	5.1788870872331
...

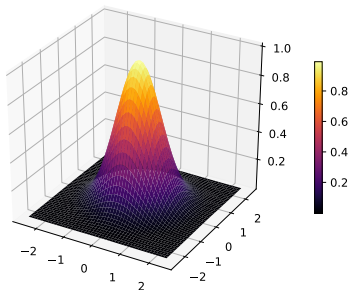
- Data needs to be transformed to a pivot table before plotting. Example:

```
1 print(z)
2 # [[3.72665317e-06 6.08307642e-06 9.73288695e-06 ... 1.52642052e-05
3 #    9.73288695e-06 6.08307642e-06]
4 #    ...
5 #    [6.08307642e-06 9.92950431e-06 1.58871492e-05 ... 2.49160097e-05
6 #    1.58871492e-05 9.92950431e-06]]
```

- Transform to pivot table with `pandas.pivot_table()` (see later)

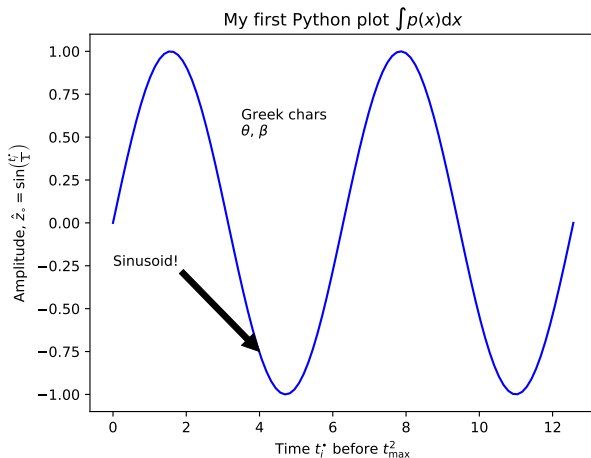
3D surface

```
1 from mpl_toolkits.mplot3d import Axes3D
2
3 # Create figure amenable for 3D plotting
4 hFig = plt.figure()
5 hAx = hFig.gca(projection="3d")
6
7 # Plot the surface
8 hSurf = hAx.plot_surface(x, y, z, cmap=cm.inferno)
9 plt.colorbar(hSurf, shrink=0.5)
```



Mathematical annotations in graphs

- Broad support for `text` annotation in matplotlib



Mathematical annotations in graphs

```
1 # Generate sinusoidal data
2 x = np.linspace(0, 4*np.pi, 100)
3 y = np.sin(x)
4
5 # Plot sinusoidal curve
6 plt.plot(x, y, color="blue")
7
8 # labels using latex commands
9 plt.xlabel(r"Time  $t_{i}^{\bullet}$  before  $t_{\mathrm{max}}$   

10  $\}^2$ ")
11 plt.ylabel(r"Amplitude,  $\hat{z}_{\circ} = \sin \left( \frac{t_{i}^{\bullet}}{1} \right)$ ")
12
13 # plot title
14 plt.title(r"My first Python plot  $\int p(x) \mathrm{d}x$ ")
15
16 # text label with arrow using pyplot.annotate
17 plt.annotate(s = "Sinusoid!", xy=(4.0, -0.75), xytext
18             =(0, -0.25), arrowprops=dict(facecolor="black", linewidth
19             =0.01))
20
21 # text annotation within the plot using pyplot.text
22 plt.text(x=4.7, y = 0.5, s = r"A label  $\theta$ ,  $\beta$ ")
```

Saving figures to files

- One can automate saving figures to files using `pyplot.savefig`

```
1 # at the end of the figure
2
3 # store figure in file (png, eps, pdf, svg, etc)
4 plt.savefig(fname="lineplot.svg")
5 plt.close() # plot stays in memory unless closed
```

- Note that svg format can easily be edited using vector graphics software like Illustrator or Inkscape

Finer control over multipanel figures using gridspec

```
1 import matplotlib.gridspec as gridspec
2
3 # set up a 2 x 2 grid with lower row half the size of top row
4 gs = gridspec.GridSpec(
5     nrows=2
6     ,ncols=2
7     ,height_ratios=[1,0.5]
8     ,width_ratios=[1,1])
9
10 plt.figure()
11 # first subplot top left corner, return Axes object for control over axes
12 ax = plt.subplot(gs[0,0])
13 ax.plot(...)
14
15 # work with axes, e.g., remove x-tick labels
16 ax.set_xticklabels([])
17
18 # second subplot top right corner
19 plt.subplot(gs[0,1])
20 plt.plot(...)
21
22 # third subplot bottom left corner
23 plt.subplot(gs[1,0])
24 plt.plot(...)
25
26 # second subplot bottom right corner
27 plt.subplot(gs[1,1])
28 plt.plot(...)
29
30 plt.show()
```