DM587/DM579

Scientific Programming

Linear Algebra with Applications

Intoduction to Python - Part 3

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

[*Based on booklet Python Essentials*]

Outline

1. Matplotlib
2. Other Data Visualization Libraries
3. Pandas

Outline

Matplotlib Library

Matplotlib is a low level graph plotting library in Python that serves as a visualization utility. Most of the Matplotlib utilities lies under the `pyplot` submodule, and are usually imported under the `plt` alias

```
>>> %matplotlib inline
>>> import matplotlib.pyplot as plt
```

In IPython, Jupyter and Jupyter Lab, functions with `%` are extra functions of IPython that add functionalities to the environment. They are called magic functions.

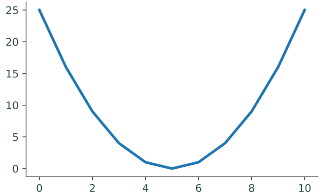
- `%matplotlib inline` shows the plot
- `%matplotlib notebook` shows the plot and provides controls to interact with the plot

Other useful magic function are:

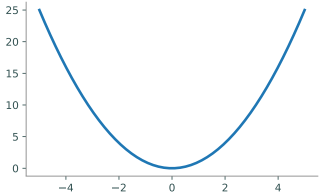
- `%timeit` to determine running time of a command and
- `%run` to run a script from a file.

Line Plots

```
>>> import numpy as np
>>> from matplotlib import pyplot as plt

>>> y = np.arange(-5,6)**2
>>> y
array([25, 16, 9, 4, 1, 0, 1, 4, 9, 16, 25])

# Visualize the plot.
>>> plt.plot(y)    # default values x-values 0, 1, 2, 3,
[<matplotlib.lines.Line2D object at 0x10842d0>]
>>> plt.show()     # Reveal the resulting plot.
```




```
>>> x = np.linspace(-5, 5, 50)
>>> y = x**2       # Calculate the range of f(x) = x**2.
>>> plt.plot(x,y)
>>> plt.show()
```



Plotting a Polynomial Function

Let's plot the following polynomial of degree 3:

$$P_3(x) = x^3 - 7x + 6 = (x-1)(x-2)(x+3)$$

The `numpy` function `numpy.poly1d` takes an array of coefficients of length n+1 (try `numpy.polyfit?`):

`a[0] * x**n + a[1] * x**(n-1) + ... + a[n-1]*x + a[n]`

```
>>> import numpy as np
>>> a=[1,0,-7,6]
>>> P=np.poly1d(a)
>>> print(P)
   3
1 x - 7 x + 6

>>> x = np.linspace(-3.5, 3.5, 500)
>>> plt.plot(x, P(x), '-')
>>> plt.axhline(y=0)
>>> plt.title('A polynomial of order 3');
```

Interactive Plotting

In non-interactive mode (deafult):

- newly created figures and changes to figures will not be reflected until explicitly asked to be;
- `.pyplot.show` will block by default.

The interactive mode is mainly useful to build plots from the command line and see the effect of each command while building the figure.
In interactive mode:

- newly created figures will be shown immediately;
- figures will automatically redraw on change;
- `.pyplot.show` will not block by default.

`plt.ion()`	Enable interactive mode
`plt.clf()`	Clear figure
`plt.ioff()`	Disable interactive mode
`plt.show()`	Show all figures (and maybe block)
`plt.pause()`	Show all figures, and block for a time.

Plot Customization

For `plt.plot()`

`marker`	Style
`’.’`	point marker
`’o’`	circle marker
`’*’`	star marker
`’+’`	plus marker
...	...

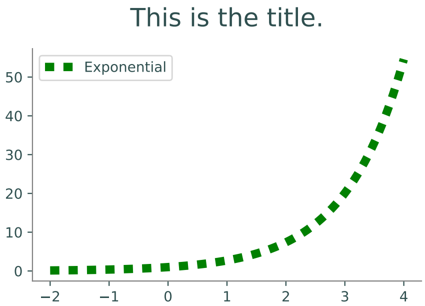
`linestyle`	Style
`’-’`	solid line
`’-’`	dashed line
`’-.’`	dash-dot line
`’:’`	dotted line
...	...

`color`	Color
`’b’`	blue
`’g’`	green
`’r’`	red
`’c’`	cyan
`’k’`	black

The parameter `fmt` is written with this syntax `marker|line|color`
`markersize` (or `ms`) to set the size of the markers. Other functions

Function	Description
`legend()`	Place a legend in the plot
`title()`	Add a title to the plot
`xlim()` / `ylim()`	Set the limits of the x- or y-axis
`xlabel()` / `ylabel()`	Add a label to the x- or y-axis
`grid()`	Add the grid lines

```
>>> x1 = np.linspace(-2, 4, 100)
>>> plt.plot(x1, np.exp(x1), 'g:', ↪
    ↪linewidth=6, label="Exponential"↪
    ↪)
>>> plt.title("My title", fontsize=18)
>>> plt.legend(loc="upper left")
>>> plt.show()
```



Layout

Function	Description
`figure()`	Create a new figure or grab an existing figure
`axes()`	Add an axes to the current figure
`gca()`	Get the current axes
`gcf()`	Get the current figure
`subplot()`	Add a single subplot to the current figure
`subplots()`	Create a figure and add several subplots to it

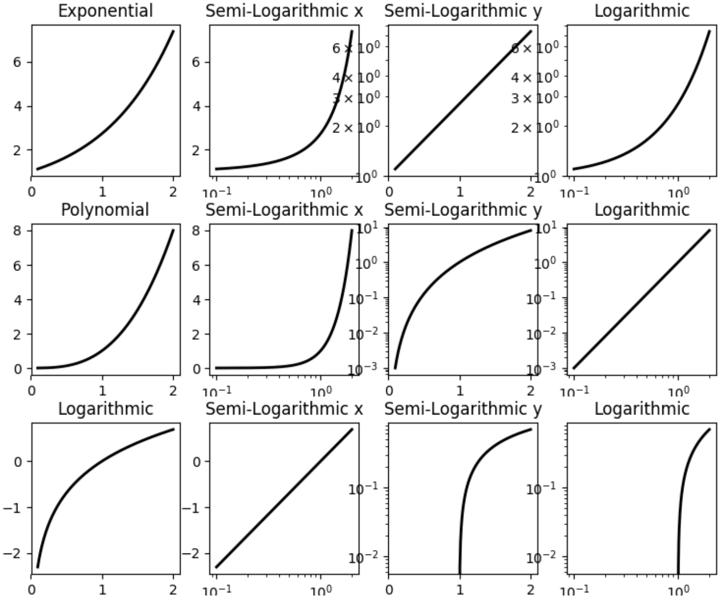
`subplot` takes three arguments: the layout is organized in rows and columns, which are represented by the first and second argument. The third argument represents the index of the current plot.

```
# 3. Use plt.subplots() to get the figure and all subplots simultaneously.
>>> fig, axes = plt.subplots(1, 2)
>>> axes[0].plot(x, 2*x)
>>> axes[1].plot(x, x**2)
```

Compare `axes()` vs `axis()` (access properties of the current plot)

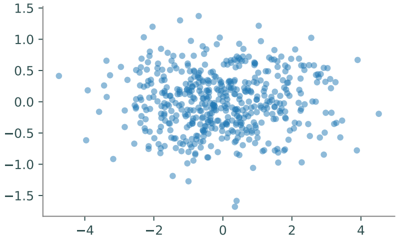
```python
import numpy as np
from matplotlib import pyplot as plt

def make_figure(x,f,name):
    plt.figure(figsize=(9,2))
    ax1 = plt.subplot(141)
    ax1.plot(x, f(x), 'k', lw=2)
    plt.title(name)

    ax2 = plt.subplot(1,4.2)
    ax2.semilogx(x, f(x), 'k', lw=2)
    ax2.set_title("Semi-Logarithmic x")

    ax3 = plt.subplot(143)
    ax3.semilogy(x, f(x), 'k', lw=2)
    ax3.set_title("Semi-Logarithmic y")

    ax4 = plt.subplot(144)
    ax4.loglog(x, f(x), 'k', lw=2)
    ax4.set_title("Logarithmic")
    plt.savefig(name+".png")
```

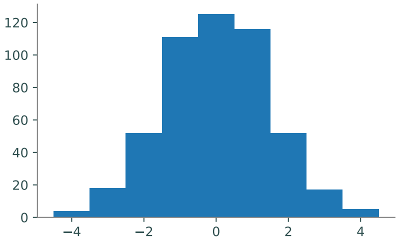
```python
xx = np.linspace(.1, 2, 200)
make_figure(xx, lambda xx: np.exp(xx), ↪
    ↪"Exponential")
make_figure(xx, lambda xx: xx**3, "↪
    ↪Polynomial")
make_figure(xx, lambda xx: np.log(xx), ↪
    ↪"Logarithmic")
```


Exponential
Semi-Logarithmic x
Semi-Logarithmic y
Logarithmic
Polynomial
Semi-Logarithmic x
Semi-Logarithmic y
Logarithmic
Logarithmic
Semi-Logarithmic x
Semi-Logarithmic y
Logarithmic

Scatter Plots and Histograms

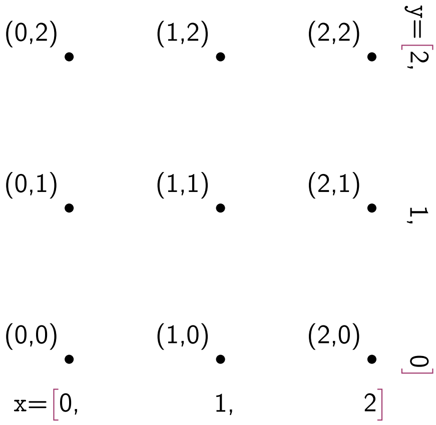
```
>>> x = np.random.normal(scale=1.5, size=500)
>>> y = np.random.normal(scale=0.5, size=500)

>>> ax1 = plt.subplot(121)
>>> ax1.plot(x, y, 'o', markersize=5, alpha=.5) # transparent circles

>>> ax2 = plt.subplot(122)
>>> ax2.hist(x, bins=np.arange(-4.5, 5.5))
>>> plt.show()
```





3D Surfaces

`np.meshgrid()` given two 1-dimensional coordinate arrays, creates two corresponding coordinate matrices: `(X[i,j], Y[i,j]) = (x[i],y[j])`.



$$\mathtt{X}= \begin{vmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{vmatrix}$$

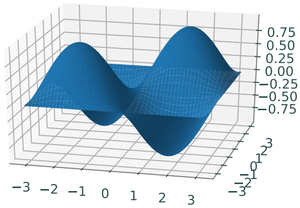
$$\mathtt{Y}= \begin{vmatrix} 2 & 2 & 2 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{vmatrix}$$

```
>>> x, y = [0, 1, 2], [3, 4, 5]      # A rough domain over [0,2]x[3,5].
>>> X, Y = np.meshgrid(x, y)         # Combine the 1-D data into 2-D data.
>>> for trows in zip(X,Y):
...     print(trows)
...
(array([0 1 2]), array([3 3 3]))
(array([0 1 2]), array([4 4 4]))
(array([0 1 2]), array([5 5 5]))
```

3D Surfaces

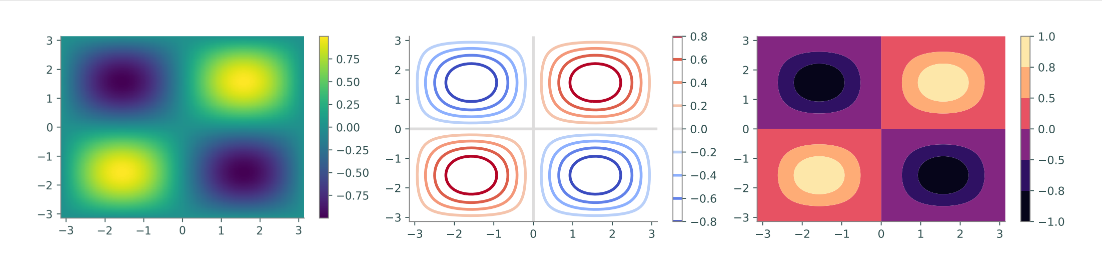
$$g(x, y) = \sin(x)\sin(y)$$

```
>>> x = np.linspace(-np.pi, np.pi, 200)
>>> y = np.copy(x)
>>> X, Y = np.meshgrid(x, y)
>>> Z = np.sin(X) * np.sin(Y)

# Draw the corresponding 3-D plot using some extra tools.
>>> from mpl_toolkits.mplot3d import Axes3D
>>> fig = plt.figure()
>>> ax = fig.add_subplot(1,1,1, projection='3d')
>>> ax.plot_surface(X, Y, Z)
>>> plt.show()
```



Heat Map and Contour Plot

```
>>> x = np.linspace(-np.pi, np.pi, 100)
>>> y = x.copy()
>>> X, Y = np.meshgrid(x, y)
>>> Z = np.sin(X) * np.sin(Y)       # Calculate g(x,y) = sin(x)sin(y).
# Plot the heat map of f over the 2-D domain.
>>> plt.subplot(131)
>>> plt.pcolormesh(X, Y, Z, cmap="viridis")
>>> plt.colorbar()
>>> plt.xlim(-np.pi, np.pi)
>>> plt.ylim(-np.pi, np.pi)
# Plot a contour map of f with 10 level curves.
>>> plt.subplot(132)
>>> plt.contour(X, Y, Z, 10, cmap="coolwarm")
>>> plt.colorbar()
# Plot a filled contour map, specifying the level curves.
>>> plt.subplot(133)
>>> plt.contourf(X, Y, Z, [-1, -.8, -.5, 0, .5, .8, 1], cmap="magma")
>>> plt.colorbar()
>>> plt.show()
```


3
2
1
0
−1
−2
−3
−3 −2 −1 0 1 2 3
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
1.0
0.8
0.5
0.0
−0.5
−0.8
−1.0

Animations

1. Calculate all data that is needed for the animation.
2. Define a figure explicitly with `plt.figure()` and set its window boundaries.
3. Draw empty objects that can be altered dynamically.
4. Define a function to update the drawing objects.
5. Use `matplotlib.animation.FuncAnimation()`.

```python
from matplotlib.animation import FuncAnimation
from mpl_toolkits.mplot3d import Axes3D

def sine_animation():
    # 1. Calculate the data to be animated.
    x = np.linspace(0, 2*np.pi, 200)[:-1]
    y = np.sin(x)    #
    # 2. Create a figure and set the window boundaries of the axes.
    fig = plt.figure()
    plt.xlim(0, 2*np.pi)
    plt.ylim(-1.2, 1.2) #
    # 3. Draw an empty line. The comma after 'drawing' is crucial.
    drawing, = plt.plot([],[])  #
    # 4. Define a function that updates the line data.
    def update(index):
        drawing.set_data(x[:index], y[:index])
        return drawing,                     # Note the comma!
    # 5.
    a = FuncAnimation(fig, update, frames=len(x), interval=10)
```

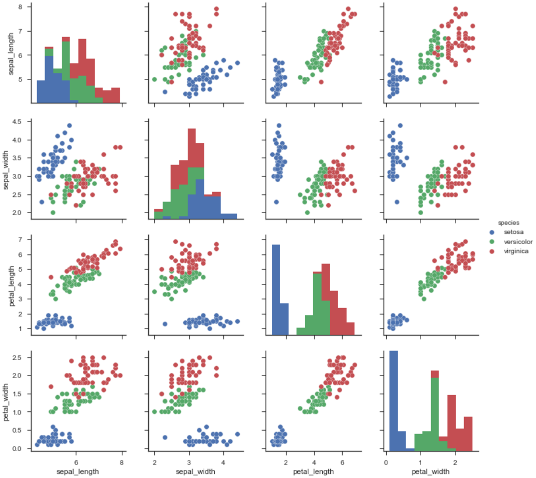
Further Reading and Tutorials

- https://www.w3schools.com/python/matplotlib_intro.asp
- https://www.labri.fr/perso/nrougier/teaching/matplotlib/.
- https://matplotlib.org/users/pyplot_tutorial.html.
- http://www.scipy-lectures.org/intro/matplotlib/matplotlib.html.
- https://matplotlib.org/2.0.0/examples/animation/index.html

Outline

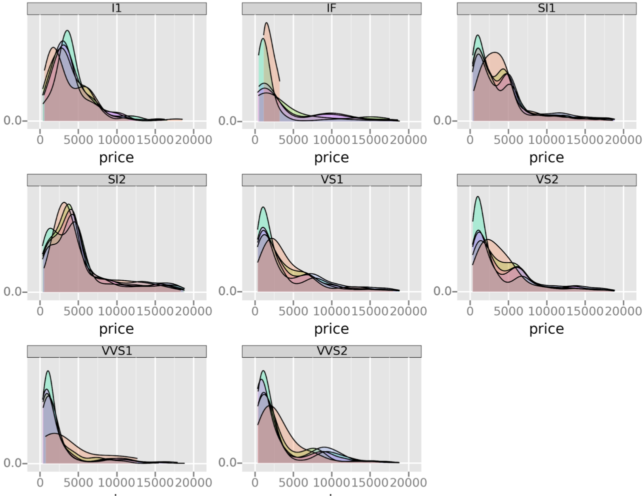
Seaborn

A high-level library on the top of Matplotlib. It's easier to generate certain kinds of plots: eg, heat maps, time series, and violin plots.



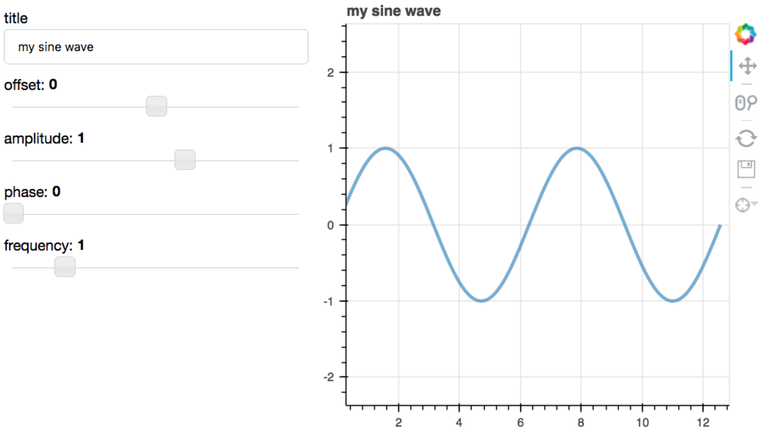
ggplot

Based on R's ggplot2 and the Grammar of Graphics



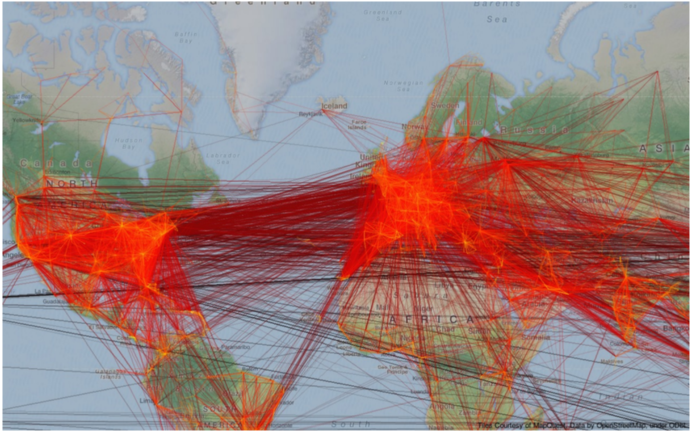
Bokeh, Plotly, Gleam, Dash and Altair

Create interactive, web-ready plots, as JSON objects, HTML documents, or interactive web applications.
Bokeh is based on the Grammar of Graphics like ggplot.
Gleam is inspired by R's Shiny package.



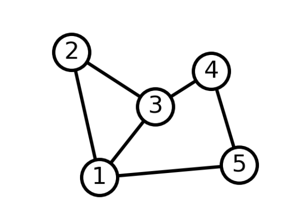
Geoplotlib, Leaflet and MapBox

Toolbox for plotting geographical data: map-type plots, like choropleths, heatmaps, and dot density maps.



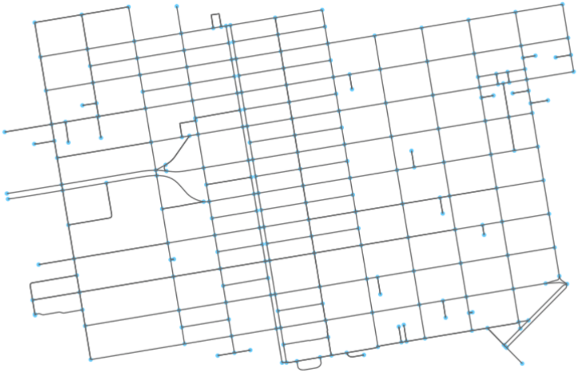
Graph Algorithms, Graph Drawings

```
import matplotlib.pyplot as plt

G = nx.Graph()
G.add_edge(1, 2)
G.add_edge(1, 3)
G.add_edge(1, 5)
G.add_edge(2, 3)
G.add_edge(3, 4)
G.add_edge(4, 5)

# explicitly set positions
pos = {1: (0, 0), 2: (-1, 0.3), 3: (2, 0.17), 4: (4, 0.255), 5: (5, 0.03)}

nx.draw_networkx(G, pos)

# Set margins for the axes so that nodes aren't clipped
ax = plt.gca()
ax.margins(0.20)
plt.axis("off")
plt.show()
```



Street Networks

```python
import osmnx as ox
G = ox.graph_from_bbox(37.79, 37.78, -122.41, -122.43, network_type='drive')
G_projected = ox.project_graph(G)
ox.plot_graph(G_projected)
```



Outline

1. Matplotlib
2. Other Data Visualization Libraries
3. Pandas

Pandas Data Structures: Series

Pandas library for data management and analysis that combines functionality of NumPy, MatPlotLib, and SQL

- `Series` is a one-dimensional array that can hold any datatype, similar to a `ndarray` but with an `index` that gives a label to each entry.

```
>>> import pandas as pd
>>>
>>> # Initialize Series of student grades
>>> math = pd.Series(np.random.randint(0,100,4), ['Mark', 'Barbara', 'Eleanor',↪
    ↪ 'David'])
>>> english = pd.Series(np.random.randint(0,100,5), ['Mark', 'Barbara','David',↪
    ↪ 'Greg', 'Lauren'])
>>> math
Mark        30
Barbara     71
Eleanor     94
David       41
dtype: int64
```

Pandas Data Structures: Data Frames

- `DataFrame` is a collection of multiple `Series`. It can be thought of as a 2-dimensional array, where each row is a separate datapoint and each column is a feature of the data. The rows are labelled with an `index` (as in a `Series`) and the columns are labelled in the attribute `columns`.

```
>>> # Create a DataFrame of student grades
>>> grades = pd.DataFrame({"Math": math, "English": english})
>>> grades
          Math  English
Barbara   52.0     73.0
David     10.0     39.0
Eleanor   35.0      NaN
Greg       NaN     26.0
Lauren     NaN     99.0
Mark      81.0     68.0
```

Summary

1. Matplotlib
2. Other Data Visualization Libraries
3. Pandas