DM561 / DM562

Linear Algebra with Applications

Intoduction to Python - Part 3

Marco Chiarandini

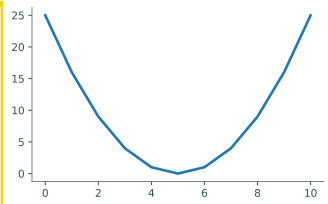
Department of Mathematics & Computer Science
University of Southern Denmark

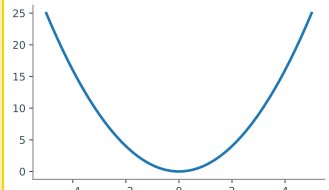
[*Based on booklet Python Essentials*]

Outline

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)   # Draw the line plot.
[<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()
```





- `np.arange()` evenly-spaced values in an interval specifying the spacing
- `np.linspace()` evenly-spaced values in an interval specifying the number of elements

Interactive Plotting

- `plt.ion()`
- `plt.clf()`
- `plt.ioff()`

In IPython Notebook.

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

Plot Customization

For `plt.plot()`

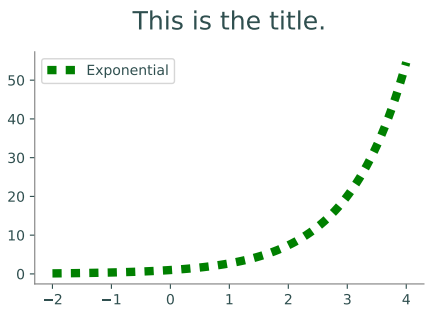
Key	Color
`'b'`	blue
`'g'`	green
`'r'`	red
`'c'`	cyan
`'k'`	black

Key	Style
`'-'`	solid line
`'--'`	dashed line
`'-.'`	dash-dot line
`':'`	dotted line
`'o'`	circle marker

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

```
>>> 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

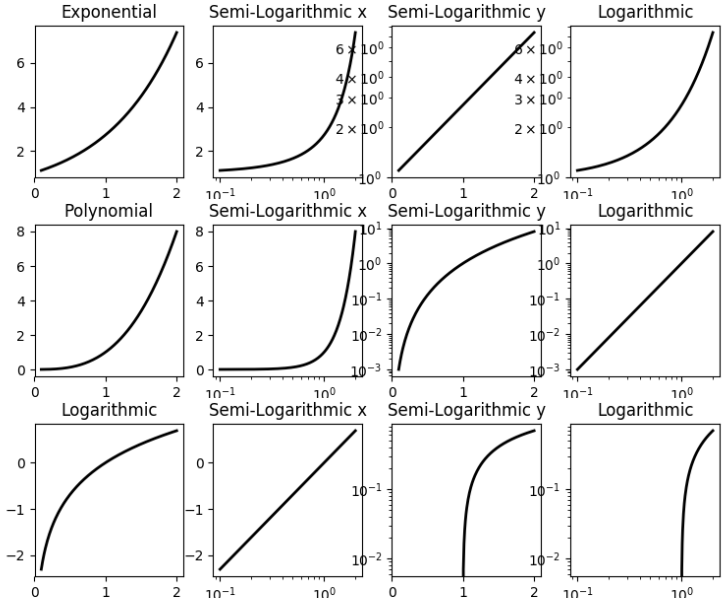
Function	Description
`axes()`	Add an axes to the current figure
`figure()`	Create a new figure or grab an existing figure
`gca()`	Get the current axes
`gcf()`	Get the current figure
`subplot()`	Add a single subplot to the current figure
`subplots()`	Create a figure and add several subplots to it

```
# 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)

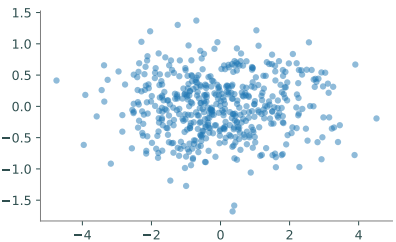
```python
import numpy as np
from matplotlib import pyplot as plt

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(142)
    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")
```

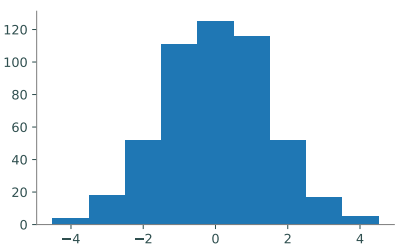
```python
xx = np.linspace(.1, 2, 200)
make_figure(xx, lambda xx: np.exp(xx), ↩
    "Exponential")
make_figure(xx, lambda xx: xx**3, "↩
    Polynomial")
make_figure(xx, lambda xx: np.log(xx), ↩
    "Logarithmic")
```


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

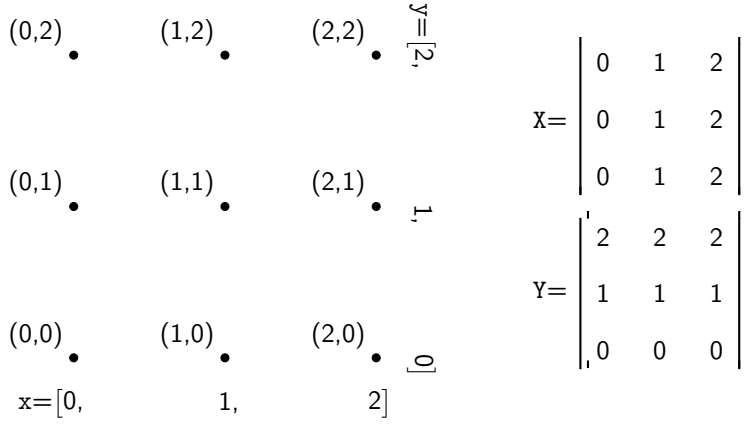
```
>>> x = np.random.normal(scale=1.5, size=500)
>>> y = np.random.normal(scale=0.5, size=500)

>>> 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

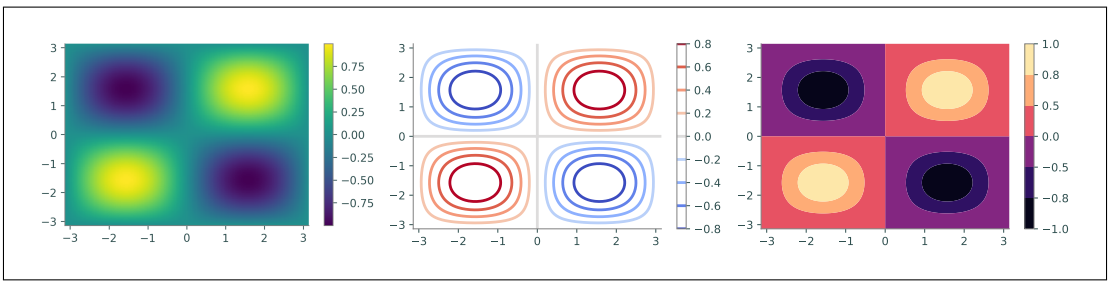
`np.meshgrid()` given two 1-dimensional coordinate arrays, creates two corresponding coordinate matrices: `(X[i,j], Y[i,j]) = (x[i],y[j])`.



```
>>> 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]))
```

Heat Map and Contour Plot

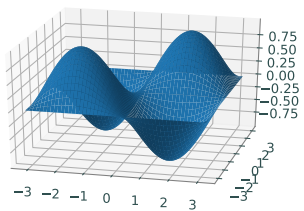
```
>>> x = np.linspace(-np.pi, np.pi, 100)
>>> y = x.copy()
>>> X, Y = np.meshgrid(x, y)
>>> Z = np.sin(X) * np.sin(Y)       # Calculate g(x,y) = sin(x)sin(y).
# Plot the heat map of f over the 2-D domain.
>>> plt.subplot(131)
>>> plt.pcolormesh(X, Y, Z, cmap="viridis")
>>> plt.colorbar()
>>> plt.xlim(-np.pi, 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

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()
```



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()`.

```
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.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