Introduction to Programming Language (ITP101)

Python Packages: Matplotlib

Mulualem Teku

GCIT, Bhutan

Nov 2019

...Previously & Today...

...Previously: The `NumPy` module

- The `array` object
- Basic array operations (creating, indexing, broadcasting, slicing ...)

Today...

matplotlib

- Line graphs
- Scatter graphs
- Histograms
- Pie charts

Recap

What is the Output of:

```
from numpy import array, arange

A = arange(1, 10, 0.5)
B = array([ [1, 2, 3], [ 7, 8, 9], [4, 5, 6] ], float)

print (A.size +  B.size)
print (A.shape)     # B.shape ??
print (B.ndim)

print (B * 4)
print (B + len(A) )

print (A[:5])
print (B[1:3, ::2])
```

Introduction

- Python is becoming the preferred language by scientists.

> *"There seems to be two sorts of people who love Python: those who hate brackets, and scientists."*

  - Steep learning curve
  - Short development time

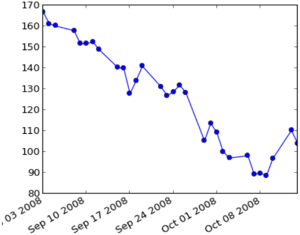
- Plots and visualizations are integral parts of scientific researches.
- Python is rich with such libraries.

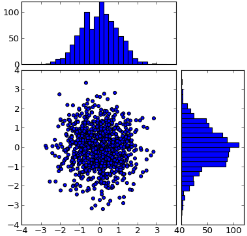
Plotting Libraries/Tools

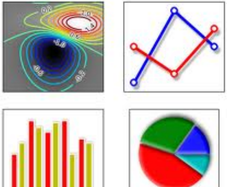
`Matplotlib/Pylab`, `Mayavi (3D)`, `Gnuplot.py`, `VPython`, ...

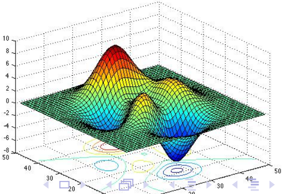
The Matplotlib Module

- A Python 2D plotting library for publication-quality figures.
- Can also work with 3D plotting toolkits (e.g. `mplot3d`).









- Get it from here.
- Requirements:
  - `NumPy`
  - `IPython` (recommended) - a matplotlib-aware interactive Python shell.
- Alternative distributions for scientific computing:
  - `Enthought's EPD` (now Enthought Canopy) - has `scipy, numpy` plus many other useful packages, preinstalled.
  - `Python(x,y)` (for Windows) - includes `matplotlib` and `pylab`, and lots of other useful tools.

Getting Started: The ipython --pylab Mode

- IPython's pylab mode is a convenient 2D plotting interface.

```
$ ipython -pylab                # start ipython in pylab mode
```

- Pylab combines pyplot with numpy into a single namespace.
- Pyplot (a module in matplotlib) interfaces to the underlying plotting library in matplotlib.
- The mode allows *interactive plotting* by making all of the plotting functions available for use.

Basic Plots: Line Graphs

plot()

- Takes arbitrary number of arguments and plots the points.

  Syntax: plot(args [, format])

```
plot(x,y)
plot(x,y,‘r-’)                # in red (r) solid line (-)
plot(y,‘b--’)                 # in blue (b) dashed line (--)
plot(x1,y1, x2, y2, ‘g^’)     # (x2,y2) in green triangular line
```

Example

```
import matplotlib.pyplot as p
x = [1, 2, 3, 4, 5, 6]
y = [1, 2, 3, 4, 5, 6]
p.plot(x,y)              # plot x and y
p.xlabel(‘X axis’)
p.ylabel(‘Y axis’)
p.show()               # show on screen
```

Basic Plots: Line Graphs

plot()

- Takes arbitrary number of arguments and plots the points.

  Syntax: plot(args [, format])

```
plot(x,y)
plot(x,y,‘r-’)                 # in red (r) solid line (-)
plot(y,‘b--’)                  # in blue (b) dashed line (--)
plot(x1,y1, x2, y2, ‘g^’)      # (x2,y2) in green triangular line
```

Example

```
import matplotlib.pyplot as p
x = [1, 2, 3, 4, 5, 6]
y = [1, 2, 3, 4, 5, 6]
p.plot(x,y)            # plot x and y
p.xlabel(‘X axis’)
p.ylabel(‘Y axis’)
p.show()             # show on screen
```

Basic Plots: Line Graphs

plot()

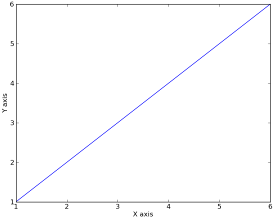
- Takes arbitrary number of arguments and plots the points.

  Syntax: `plot(args [, format])`

```
plot(x,y)
plot(x,y,‘r-’)                  # in red (r) solid line (-)
plot(y,‘b--’)                   # in blue (b) dashed line (--)
plot(x1,y1, x2, y2, ‘g^’)       # (x2,y2) in green triangular line
```

Example

```
import matplotlib.pyplot as p
x = [1, 2, 3, 4, 5, 6]
y = [1, 2, 3, 4, 5, 6]
p.plot(x,y)            # plot x and y
p.xlabel(‘X axis’)
p.ylabel(‘Y axis’)
p.show()             # show on screen
```



Formatting Line Graphs

- We can format our graphs (line color, style, etc) by inserting another parameter to the `plot()` function.

  - Line Colors

b	blue
g	green
r	red
c	cyan
m	magenta
y	yellow
w	white
k	black

  - Line Styles

`-`	solid line
`--`	dashed line
`:`	dotted line
`o`	circle marker
`D`	diamond marker
`*`	star marker
`^`	triangle marker
`h or H`	hexagon marker
`+`	plus marker

Basic Plots: Scatter Graphs

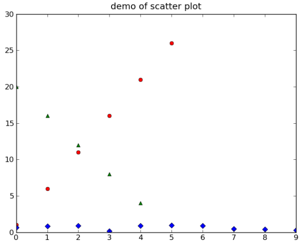
Example

```
import matplotlib.pyplot as p
from numpy import *
x = arange(1,30,5)
y = arange(20,0,-4)
z = random.random(10)   # 10 random Nos
p.plot(x,‘ro’, y, ‘g^’, z, ‘bD’)
p.title(‘demo of scatter plot’)
p.show()
```

Basic Plots: Scatter Graphs

Example

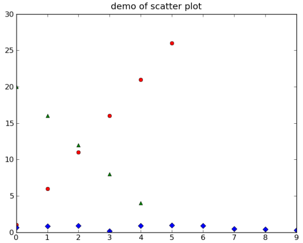
```
import matplotlib.pyplot as p
from numpy import *
x = arange(1,30,5)
y = arange(20,0,-4)
z = random.random(10)   # 10 random Nos
p.plot(x,‘ro’, y, ‘g^’, z, ‘bD’)
p.title(‘demo of scatter plot’)
p.show()
```



Basic Plots: Scatter Graphs

Example

```
import matplotlib.pyplot as p
from numpy import *
x = arange(1,30,5)
y = arange(20,0,-4)
z = random.random(10)   # 10 random Nos
p.plot(x,‘ro’, y, ‘g^’, z, ‘bD’)
p.title(‘demo of scatter plot’)
p.show()
```



Adding Legends

- You can add legends to your plots by using the legend() function.
- Used along with the **label** argument of `plot()` function.

Basic Plots: Sinusoidal

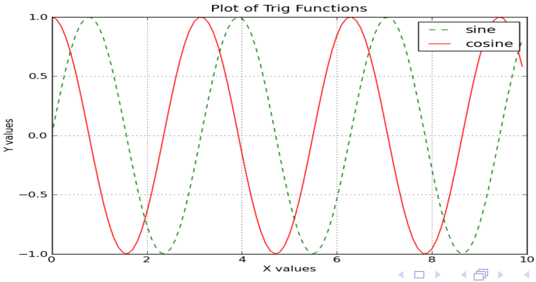
Example

```
import matplotlib.pyplot as p
import numpy as np
x = np.arange(0.0, 10.0, 0.1)
p.plot(x, np.sin(2*x), ‘g--’, label=‘sine’)
p.plot(x, np.cos(2*x), ‘r-’, label=‘cosine’)
p.legend()          # or p.legend((‘sine’, ‘cosine’))
p.xlabel(‘X values’)
p.ylabel(‘Y values’)
p.title(‘Plot of Trig Functions’)
p.grid(True)
p.show()
```

Basic Plots: Sinusoidal

Example

```
import matplotlib.pyplot as p
import numpy as np
x = np.arange(0.0, 10.0, 0.1)
p.plot(x, np.sin(2*x), ‘g--’, label=‘sine’)
p.plot(x, np.cos(2*x), ‘r-’, label=‘cosine’)
p.legend()          # or p.legend((‘sine’, ‘cosine’))
p.xlabel(‘X values’)
p.ylabel(‘Y values’)
p.title(‘Plot of Trig Functions’)
p.grid(True)
p.show()
```



Basic Plots: Histograms

- Often used in scientific applications.

  e.g. plotting probability distributions

hist()

```
hist(data [, bins=10, range=None, histtype='bar', orientation ])
```

`bins` : number of bins. Default is 10.

`range` : Lower and upper range of the bins. Default assumes `data.min()` and `data.max()`.

`histtype` : One of {bar, barstacked, step, stepfilled}. Defaults to 'bar'.

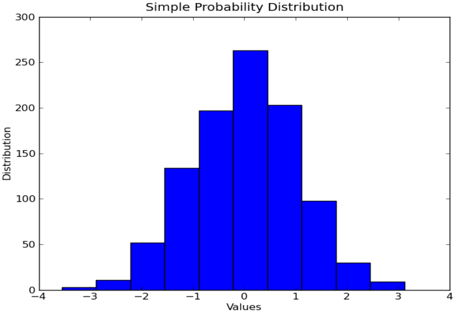
`orientation` : either {vertical, horizontal }. Default is 'vertical'.

Example

```
import matplotlib.pyplot as p
import numpy as np
data = np.random.randn(1000)           # generate 1000 random numbers
p.hist(data)                           # plot with the defaults
p.xlabel(‘Values’)
p.ylabel(‘Distribution’)
p.title(‘Simple Probability Distribution’)
p.show()
```

Example

```
import matplotlib.pyplot as p
import numpy as np
data = np.random.randn(1000)          # generate 1000 random numbers
p.hist(data)                          # plot with the defaults
p.xlabel(‘Values’)
p.ylabel(‘Distribution’)
p.title(‘Simple Probability Distribution’)
p.show()
```



Basic Plots: Pie Charts

pie()

- Makes a pie chart of the array <data>.

```
pie(data, explode, labels, colors , autopct, shadow)
```

`explode` : If given, specifies the offset of each wedge/slice.

`labels` : labels each wedge.

`colors` : none or sequence of colors.

`autopct` : If given, labels the wedges with their numeric value. It can be a format string of the form `fmt%pct`.

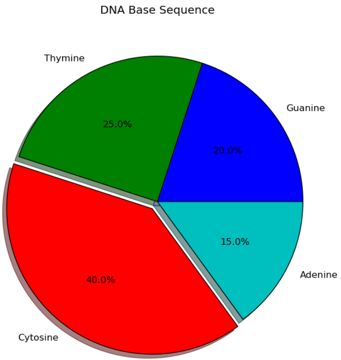
`shadow` : shadow beneath the chart. Can be either `true` or `false`.

Example

```
import matplotlib.pyplot as p
p.figure(figsize = (8,8))          # looks best if the figure & axes are squares
p.axes([0.1, 0.1, 0.8, 0.8])
names = (‘Guanine’, ‘Thymine’, ‘Cytosine’, ‘Adenine’)
data = [20, 25, 40, 15]
exp = (0, 0, 0.06, 0)
p.pie(data, explode=exp, labels=names, autopct=‘%1.1f%%’, shadow=True)
p.title(‘DNA Base Sequence’)
p.show()
```

Example

```
import matplotlib.pyplot as p
p.figure(figsize = (8,8))          # looks best if the figure & axes are squares
p.axes([0.1, 0.1, 0.8, 0.8])
names = (‘Guanine’, ‘Thymine’, ‘Cytosine’, ‘Adenine’)
data = [20, 25, 40, 15]
exp = (0, 0, 0.06, 0)
p.pie(data, explode=exp, labels=names, autopct=‘%1.1f%%’, shadow=True)
p.title(‘DNA Base Sequence’)
p.show()
```



Multiple Plots

subplot(arg)

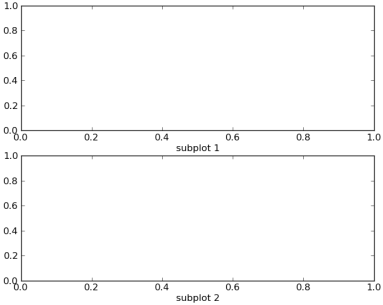
- Creates multiple subplots by manipulating layouts.
- <arg> is an integer of the form **rcp** ≡ **r**ow, **c**olumn and drawing **p**osition respectively.

```
from matplotlib.pyplot import subplot

# 2 rows, 1 col, draw @ top
subplot(211)

# 2 rows, 1 col, draw @ bottom
subplot(212)

subplot(222)
```



Example

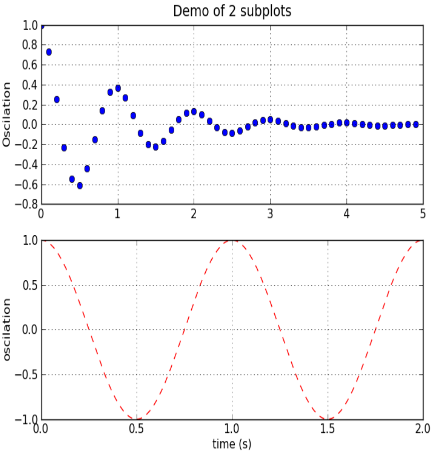
```
from matplotlib.pyplot import *
from numpy import *

x = arange(0.0, 5.0, 0.1)
y = arange(0.0, 2.0, 0.01)

subplot(211)
plot(x, cos(2*pi*x)*exp(-x), ‘bo’)
grid(True)
ylabel(‘Oscilation’)
title(‘Demo of 2 subplots’)

subplot(212)
plot(y, cos(2*pi*y), ‘r--’)
grid(True)
xlabel(‘time (s)’)
ylabel(‘oscilation’)

show()
```



Plotting from Files

- NumPy has the loadtxt() function to read from a text file (refer to its manual for more).

Assume a text file input.txt contains:

```
0         0
1         1
2         4
.         .
.         .
.         .
9         81
```

Example

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

data = np.loadtxt(‘input.txt’)
p.plot(data[:,0], data[:,1], ‘mD’)
p.xlabel(‘X values’)
p.ylabel(‘Y values’)
p.xlim(0.0, 10.)      # set x limit
p.show()
```

Plotting from Files

- NumPy has the loadtxt() function to read from a text file (refer to its manual for more).

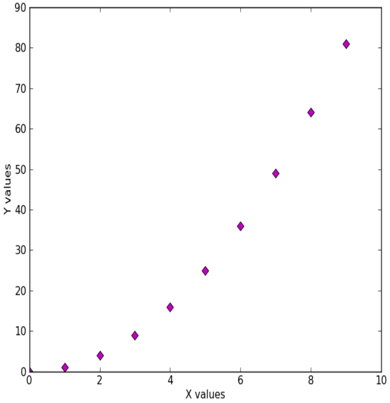
Assume a text file input.txt contains:

```
0        0
1        1
2        4
.        .
.        .
.        .
9        81
```

Example

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

data = np.loadtxt('input.txt')
p.plot(data[:,0], data[:,1], 'mD')
p.xlabel('X values')
p.ylabel('Y values')
p.xlim(0.0, 10.)      # set x limit
p.show()
```



Other Graphs

- In addition to the basic plots addressed today, you can plot many other graphs.

  - Contours
  - Bar charts
  - Polar charts
  - Maps
  - Error bars
  - Imshow
  - Quiver plots
  - Artistic paintings
  - 3D plots

  etc...

Useful Resources

- The `help()` facility of course!

Example

```
import matplotlib.pyplot
help(matplotlib)
help(matplotlib.pyplot.plot)
```

- The official `matplotlib` documentation/tutorial from here.
- Check out the examples and gallery links there.
- The `mplot3d` toolkit for simple 3D plotting. The tutorial here.