**Lecture 8 (week 5.1)**

Tuesday October 4th 2016

Matplotlib

Python has a powerful plotting package which makes very nice plots that can be customized in details, it is called **matplotlib**.

This web page will guide you through a gallery of a variety of examples, which itself is only a tiny subset of what **matplotlib** can do:

<http://matplotlib.org/gallery.html>

The objective of this lecture is to have you familiarized with the most basic plotting tools you will encounter in your career. Most of the plots we do are not fancy nor super complicated, so it is relatively straightforward to cover the basics, from which your knowledge of **matplotlib** can grow over time as you keep using it more and more.

The Python package is called **matplotlib**, and it comes with a collection of modules listed there:

<http://matplotlib.org/py-modindex.html>

For the large majority of plotting jobs, we will use the module **pyplot**. The module **matplotlib.pyplot** comes with several functionalities (or methods) which can alter the plot in many ways. You can access the whole list of these functionalities there:

<http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot>

You can have a quick overview of all the functionalities form the prompt-based help menu:

**In [1]: import matplotlib.pyplot as plt**

**In [2]: help(plt)**

Note that in this lecture, *there are some functionalities that can be executed like functions and return some variables*. This is a new concept which you will learn a bit about today through some very specific examples. We will see how this more thoroughly next week, in the lecture on *tuples*. You are also expected to go to the **matplotlib.pyplot** functionalities page to figure out what these methods are doing.

Simple plots

Let’s get started with a simple plot like this one:

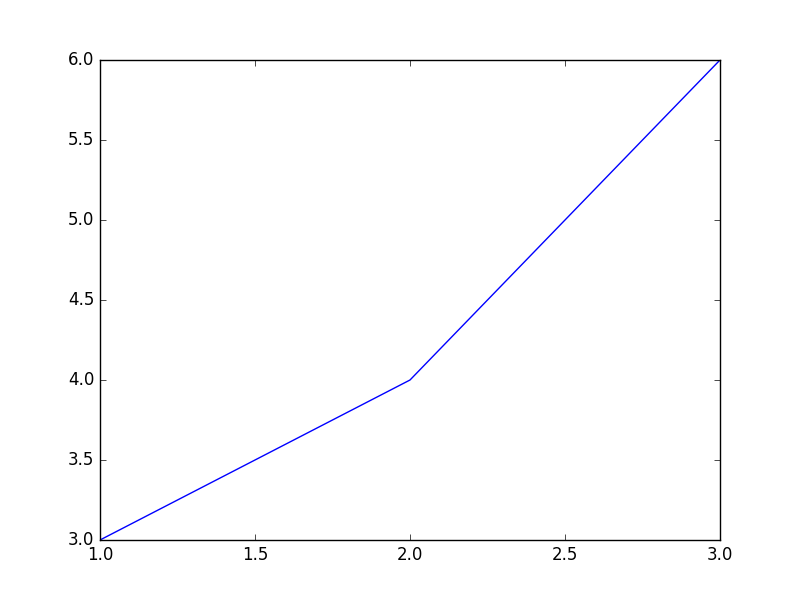
**In [3]: import matplotlib.pyplot as plt**

**In [4]: xlist = [1,2,3]** *# make xlist*

**In [5]: ylist = [3,4,6]** *# make ylist*

**In [6]: plt.plot(xlist, ylist)** *# plot y vs x*

**Out[6]: [<matplotlib.lines.Line2D at 0x10e718748>]**

**In [7]: plt.show(block=False)**

Note that **plt.plot** creates the plot object without showing the plot, while **plt.show(block=False)** displays the figure. The **block=False** argument allows you to retain access to the Python prompt. The same behavior is observed when you run the commands from a script.

Note the *controls buttons* at the bottom of the window, play with it, explore all it can do.

The optional third argument in **plot()** is related to *point/line/color* style.

**rgbcymkw** : red, green, blue, cyan, yellow, magenta, black, white

"**-**" line "**--**" dashed line "**-.**" dash-dot line "**:**" dotted line

"**o**" circle "**+**" plus "**x**" x-symbol "**s**" square "**\***" star

"**^**" up-triangle "**v**" down-triangle "**D**" diamond

Default "**b-**" is solid blue lines with no points.

"**ro--**" is red circles connected by red dashed lines

For example, the following script shows you how you can alter the x and y range and put labels:

**import matplotlib.pyplot as plt**

**xlist = [1,2,3]**

**ylist = [3,4,6]**

**plt.plot(xlist, ylist)**

**plt.plot(xlist, ylist, "ro-")**

**plt.title("This is the Title")**

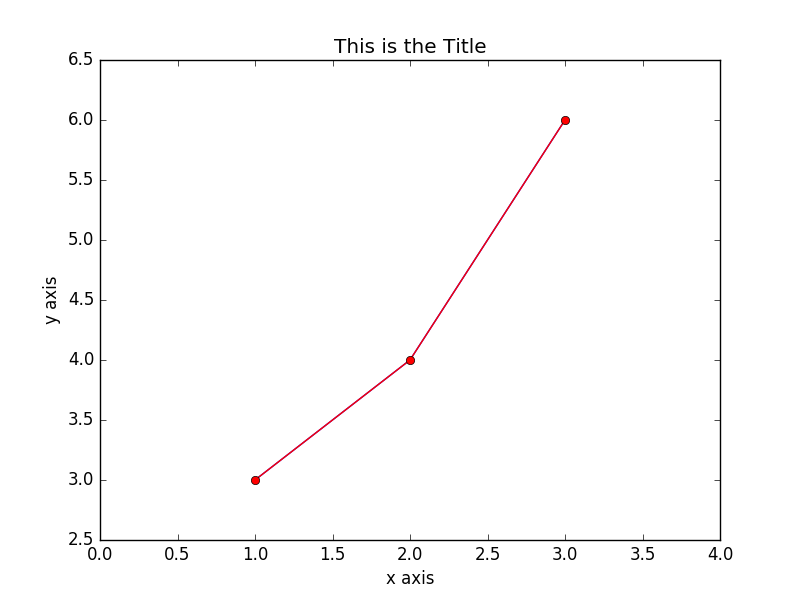
**plt.xlabel("x axis")**

**plt.ylabel("y axis")**

**plt.xlim(0.0, 4.0)**

**plt.ylim(2.5, 6.5)**

**plt.show(block=False)**



and how to change line style and add a legend box:

**import matplotlib.pyplot as plt**

**xlist = [1,2,3]**

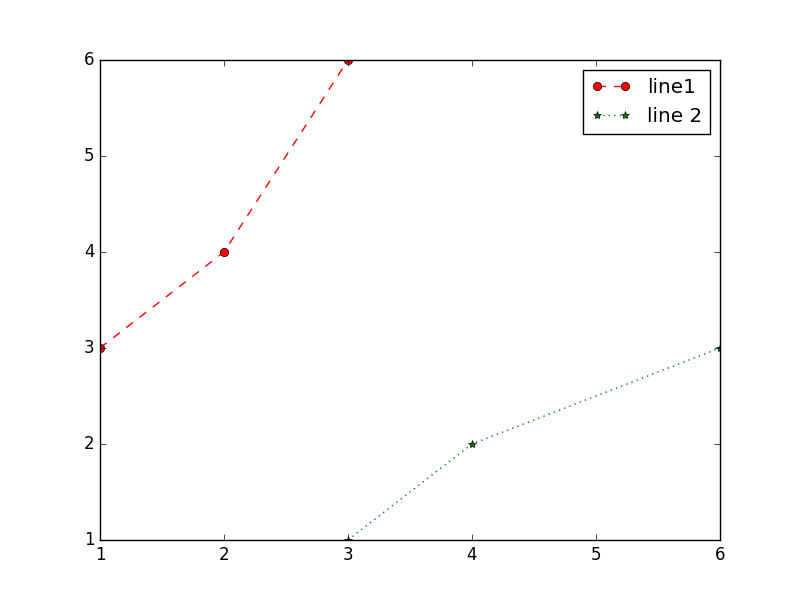
**ylist = [3,4,6]**

**plt.plot(xlist, ylist, "o--r", label="line1")**

**plt.plot(ylist, xlist, "\*:g" , label="line 2")**

**plt.legend()**

**plt.show(block=False)**



Note that **pyplot** functionalities are usually called with argument(s), some of them are optional (the full list of arguments for each method is given in the previous link above). For instance if you want to create a plot with error bars using the method **errorbar**:

**import matplotlib.pyplot as plt**

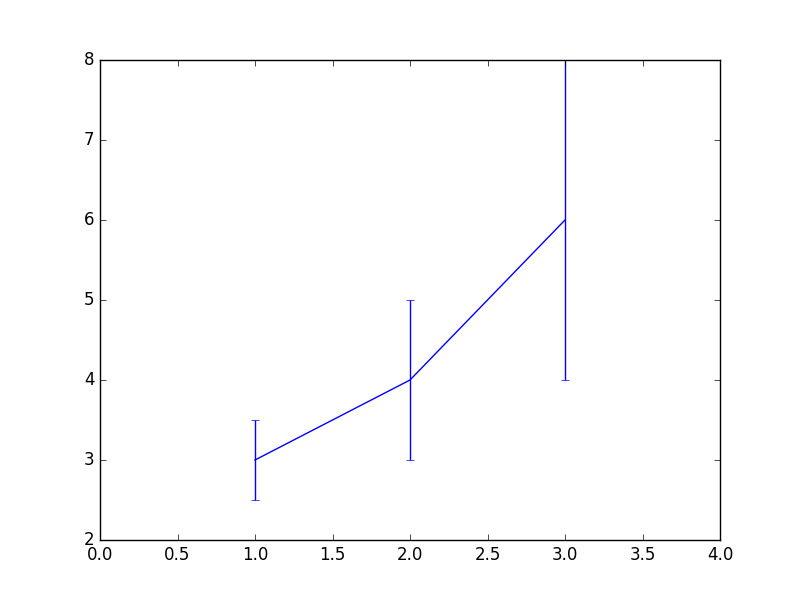
**xlist = [ 1, 2, 3]**

**ylist = [ 3, 4, 6]**

**yerrs = [0.5, 1, 2]**

**plt.errorbar(xlist, ylist, yerrs)**

**plt.xlim(0.0, 4.0) # moves bounds away from points**

**plt.show(block=False)**

Then supplying a **fmt** (format) argument to suppress the lines:

**import matplotlib.pyplot as plt**

**xlist = [ 1, 2, 3]**

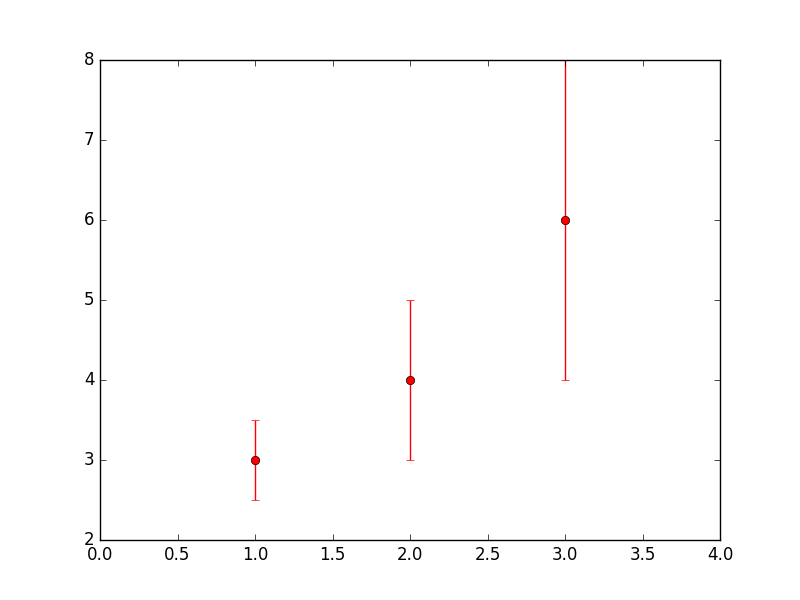
**ylist = [ 3, 4, 6]**

**yerrs = [0.5, 1, 2]**

**plt.xlim(0.0, 4.0)**

**plt.errorbar(xlist, ylist, yerrs, fmt="ro")**

**plt.show(block=False)**

****

Note that many of the functions in **matplotlib** (as well as **numpy**, and **scipy**) can be called with a large number of arguments. Fortunately, there are sensible default values for most of those arguments, and in practice it is ok to call most functions with only a few arguments. Learning what the function really does and how to adjust the input arguments comes with practice when trying to solve a specific problem.

In general you will use **numpy** arrays (the fact is, **pyplot** converts numbers in **numpy** arrays internally). The example below illustrates a plotting several lines with different format styles in one command using arrays:

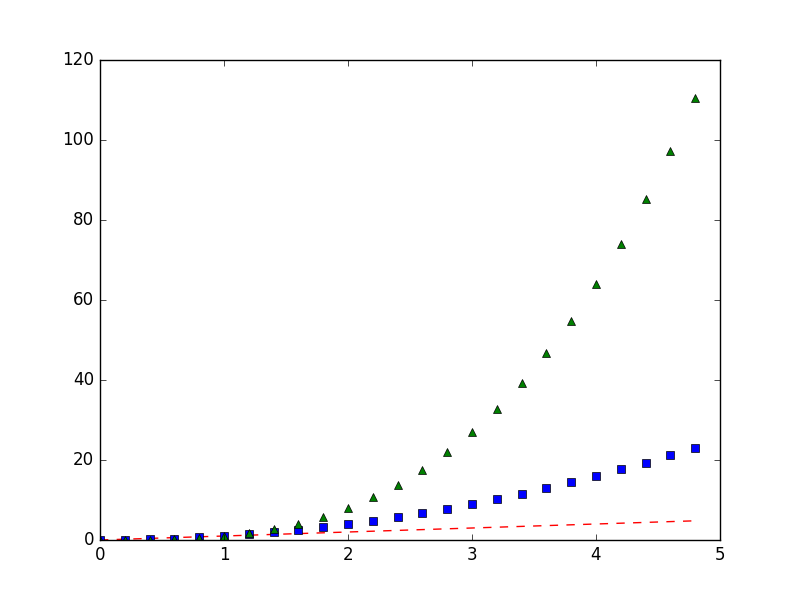
**import numpy as np**

**import matplotlib.pyplot as plt**

**t = np.arange(0., 5., 0.2)**

**plt.plot(t, t, 'r--', t, t\*\*2, 'bs', t, t\*\*3, 'g^')**

**plt.show(block=False)**

****

Saving your figure

When you want to save your file in **filename**, e.g. to use it in a document or print it, you can use the **savefig(filename)** method from **pyplot**. Note that **savefig()** comes with several arguments which can alter the layout of the save figure. Suported formats are png, pdf, ps, eps and svg.

Multiple plots

The **subplot()** command specifies **numrows**, **numcols**, **fignum**, where **fignum** ranges from 1 to numrows\*numcols. Try the following script:

**import numpy as np**

**import matplotlib.pyplot as plt**

**def f(t):**

**return np.exp(-t) \* np.cos(2\*np.pi\*t)**

**t1 = np.arange(0.0, 5.0, 0.1)**

**t2 = np.arange(0.0, 5.0, 0.02)**

**plt.figure(1)**

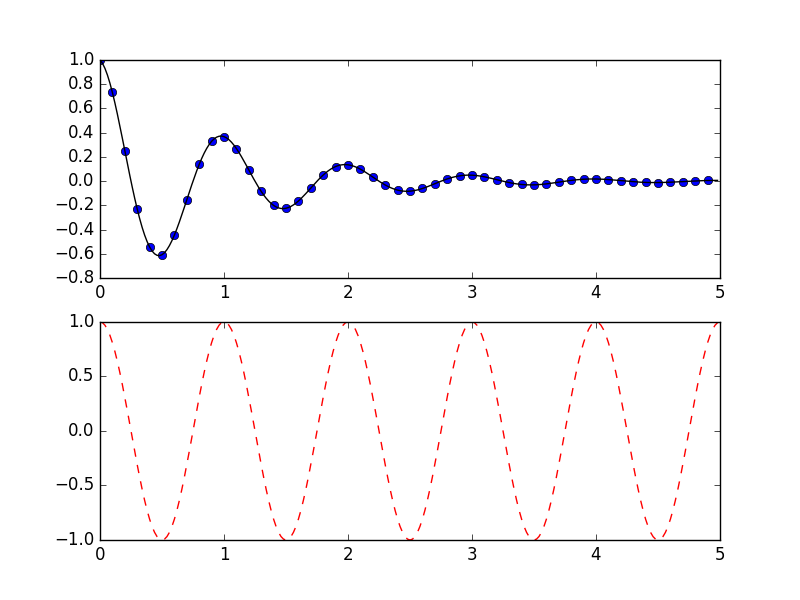
**plt.subplot(211)**

**plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')**

**plt.subplot(212)**

**plt.plot(t2, np.cos(2\*np.pi\*t2),'r--')**

**plt.show(block=False)**

****

Histograms

The method **hist()** creates and histogram of values in an array or arrays. The following script will first create a normal distribution of random values and then makes an histogram:

**import numpy as np**

**import matplotlib.pyplot as plt**

**mu, sigma = 100, 15**

**x = mu + sigma \* np.random.randn(10000)**

*# the histogram of the data*

**n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75)**

**plt.xlabel('Smarts')**

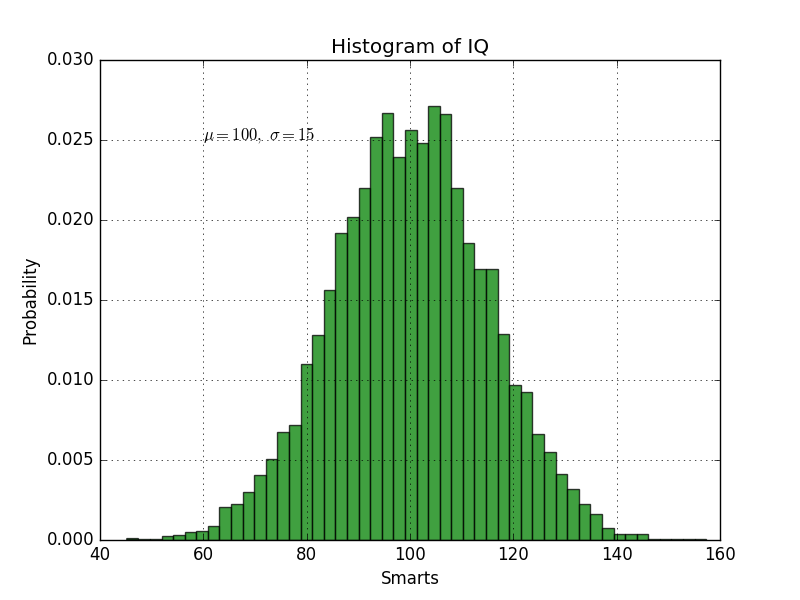
**plt.ylabel('Probability')**

**plt.title('Histogram of IQ')**

**plt.text(60, .025, r'$\mu=100,\ \sigma=15$')**

**plt.axis([40, 160, 0, 0.03])**

**plt.grid(True)**

**plt.show(block=False)**

Note that the method **hist()** is used as a function and returns some values. What are these values? how are they used by the subsequent calls to **pyplot** methods to make the plot?

Scatter plot

The example below is a 2-dimensional scatter plot with a bar next to it to indicate the color scheme:

**import numpy as np**

**import matplotlib.pyplot as plt**

**x = np.random.random(50)**

**y = np.random.random(50)**

**c = np.random.random(50)** *# color of points*

**s = 500 \* np.random.random(50)** *# size of points*

**fig, ax = plt.subplots()**

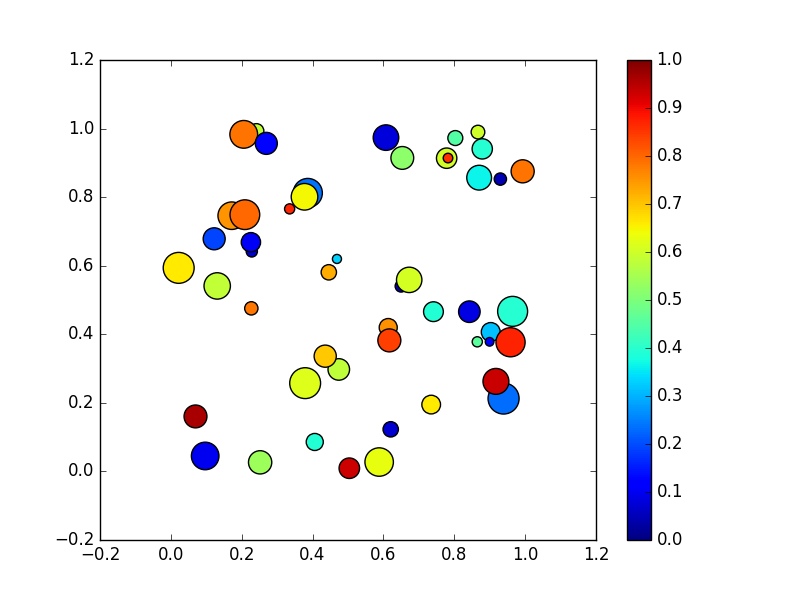
**im = ax.scatter(x, y, c=c, s=s, cmap=plt.cm.jet)**

*# Add a colorbar*

**fig.colorbar(im, ax=ax)**

*# set the color limits - not necessary here, but good to know how.*

**im.set\_clim(0.0, 1.0)**

**plt.show(block=False)**

Note that the **plt.subplots()** method is also called as a function here, while in the multiple plot section above it was not. What is different here?

3D plot

This is a bit more advanced and we won’t use it for now, but it shows how to create an interactive 3D scatter plot. It used the package mpl\_toolkits, which is different from matplotlib. Information about this package and how we use it here can be found there:

<http://matplotlib.org/1.4.3/mpl_toolkits/mplot3d/index.html>

**import numpy as np**

**import matplotlib.pyplot as plt**

**import mpl\_toolkits.mplot3d.axes3d as p3**

**x = np.random.random(50)**

**y = np.random.random(50)**

**z = np.random.random(50)**

**c = np.random.random(50)**

**fig=plt.figure()**

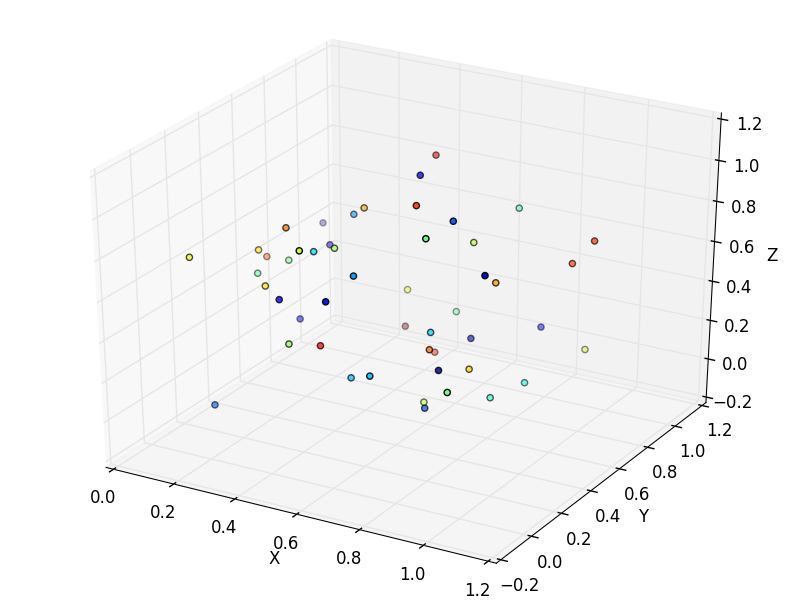
**ax = p3.Axes3D(fig)**

**ax.scatter(x,y,z,c=c)**

**ax.set\_xlabel('X')**

**ax.set\_ylabel('Y')**

**ax.set\_zlabel('Z')**

**plt.show(block=False)**