Basic Beginners' Introduction to plotting in Python

Sarah Blyth

July 23, 2009

1 Introduction

Welcome to a very short introduction on getting started with plotting in Python! I would highly recommend that you refer to the official Matplotlib documentation which can be found at:
`http://matplotlib.sourceforge.net/contents.html`.
You can also download the Matplotlib manual from the Astronomy Department Vula repository (faster) or directly from the Matplotlib website (slower).

There are all sorts of examples and further very important and useful details listed and explained there. The aim of this document is merely to start you off on Day 1 - how to make a plot from scratch.

2 Getting started - what do you need?

You need to have the following installed on your computer to be able to make nice plots.

- **Python**
- **Numpy** - this is the module which does most array and mathematical manipulation
- **Matplotlib** - this is the module you will be using for plotting

You can check these are installed by going to a terminal and typing:

```
$ python
>>> import numpy as np
>>> import pylab as pl
```

If you get no errors, then they are installed. If they are not installed, speak to the department SysAdmin to help you install them.
You can also check which versions you have installed by doing:

```
$ python
>>> import numpy
>>> numpy.__version__
```

The output you get should look something like:

```
'1.2.1'
```

Typing:

```
>>> import matplotlib
>>> matplotlib.__version__
```

should give you something like:

```
'0.98.5.2'
```

3 Basic plots

Two basic plot types which you will find are used very often are (x,y) line and scatter plots and histograms. Some code for making these two types of plots is included in this section.

3.1 Line and scatter plots

3.1.1 Line plots

A common type of graph to plot is a line relating x-values to particular y-values. The code to draw the graph in Fig. 1 is below:

```
*****************************************************
# lineplot.py

import numpy as np
import pylab as pl

# Make an array of x values
x = [1, 2, 3, 4, 5]
# Make an array of y values for each x value
y = [1, 4, 9, 16, 25]

# use pylab to plot x and y
pl.plot(x, y)

# show the plot on the screen
pl.show()
*****************************************************
```

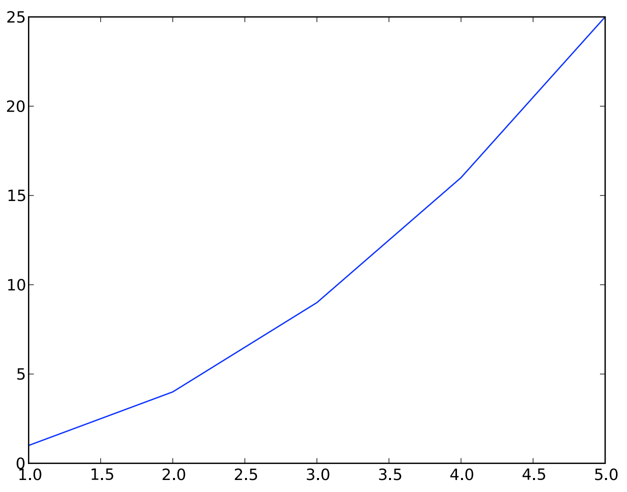

Figure 1: Line plot made with `lineplot.py`

3.1.2 Scatter plots

Alternatively, you may want to plot quantities which have an x and y position. For example, plotting the location of stars or galaxies in a field for example such as the plot in Fig. 2. Try running the script below to do this:

```
****************************************************
# scatterplot.py

import numpy as np
import pylab as pl

# Make an array of x values
x = [1, 2, 3, 4, 5]
# Make an array of y values for each x value
y = [1, 4, 9, 16, 25]

# use pylab to plot x and y as red circles
pl.plot(x, y, 'ro')

# show the plot on the screen
pl.show()
****************************************************
```

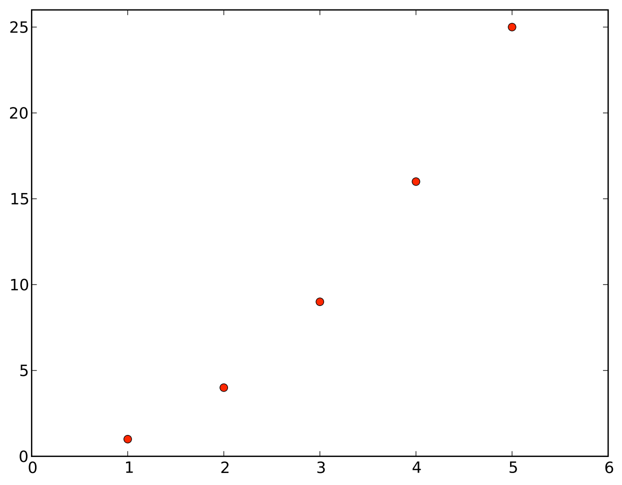

Figure 2: Plot made with `scatterplot.py`

3.2 Making things look pretty

3.2.1 Changing the line color

It is very useful to be able to plot more than one set of data on the same axes and to be able to differentiate between them by using different line and marker styles and colours. You can specify the colour by inserting a 3rd parameter into the `plot()` command. For example, in `lineplot.py`, try changing the line

```
pl.plot(x, y)
```

to

```
pl.plot(x, y, 'r')
```

This should give you the same line as before, but it should now be red.
The other colours you can easily use are:

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

3.2.2 Changing the line style

You can also change the style of the line e.g. to be dotted, dashed, etc. Try:

```
plot(x,y, '--')
```

This should give you a dashed line now.
Other linestyles you can use can be found on the Matplotlib webpage `http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot`.

3.2.3 Changing the marker style

Lastly, you can also vary the marker style you use. Try:

```
plot(x,y, 'b*')
```

This should give you blue star-shaped markers. The table below gives some more options for setting marker types:

's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin diamond marker

3.2.4 Plot and axis titles and limits

It is very important to always label the axes of plots to tell the viewer what they are looking at. You can do this in python by using the commands:

```
pl.xlabel('put text here')
pl.ylabel('put text here')
```

You can make a title for your plot by:

```
pl.title('Put plot title here')
```

You can change the x and y ranges displayed on your plot by:

```
pl.xlim(x_low, x_high)
pl.ylim(y_low, y_high)
```

Have a look at the modified macro `lineplotAxis.py` below:

```
**************************************************
#lineplotAxis.py

import numpy as np
import pylab as pl

# Make an array of x values
x = [1, 2, 3, 4, 5]
# Make an array of y values for each x value
y = [1, 4, 9, 16, 25]

# use pylab to plot x and y
pl.plot(x, y)

# give plot a title
pl.title('Plot of y vs. x')

# make axis labels
pl.xlabel('x axis')
pl.ylabel('y axis')

# set axis limits
pl.xlim(0.0, 7.0)
pl.ylim(0.0, 30.)

# show the plot on the screen
pl.show()

**************************************************
```

This should give you the plot shown in Fig. 3.

3.2.5 Plotting more than one plot on the same set of axes

It is very easy to plot more than one plot on the same axes. You just need to define the x and y arrays for each of your plots and then:

```
plot(x1, y1, 'r')
plot(x2, y2, 'g')
```

Check out this macro, `lineplot2Plots.py` and the resulting Fig. 4.

```
**************************************************
#lineplot2Plots.py

import numpy as np
import pylab as pl
```

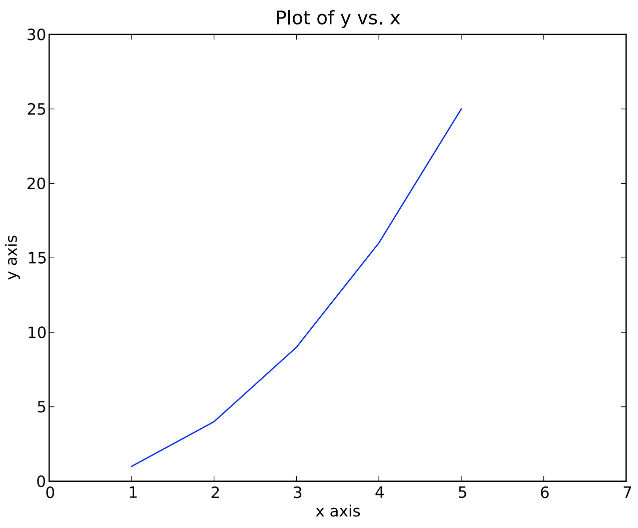

Figure 3: Plot made with `lineplotAxis.py`

```
# Make x, y arrays for each graph
x1 = [1, 2, 3, 4, 5]
y1 = [1, 4, 9, 16, 25]

x2 = [1, 2, 4, 6, 8]
y2 = [2, 4, 8, 12, 16]

# use pylab to plot x and y
pl.plot(x1, y1, 'r')
pl.plot(x2, y2, 'g')

# give plot a title
pl.title('Plot of y vs. x')
# make axis labels
pl.xlabel('x axis')
pl.ylabel('y axis')

# set axis limits
pl.xlim(0.0, 9.0)
pl.ylim(0.0, 30.)

# show the plot on the screen
pl.show()
****************************************************
```

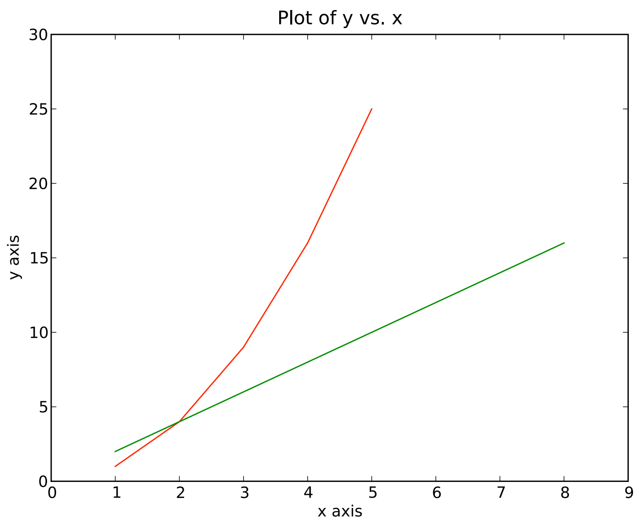

Figure 4: Plot made with `lineplot2Plots.py`

3.2.6 Figure legends

It's very useful to add legends to plots to differentiate between the different lines or quantities being plotted. In python you can make a legend as follows:

```
pl.legend((plot1, plot2), ('label1, label2'), 'best', numpoints=1)
```

The first parameter is a list of the plots you want labelled. The second parameter is the list of labels. The third parameter is where you would like matplotlib to place your legend. Other optiions are:
'upper right', 'upper left', 'center', 'lower left', 'lower right'.
'best' means that matplotlib will try to choose the position on your plot where the legend will interfere least with what is plotted (i.e. avoid overlaps etc.).
Have a look at Fig. 5 which is made using the macro below:

```
****************************************************
# lineplotFigLegend.py

import numpy as np
import pylab as pl

# Make x, y arrays for each graph
x1 = [1, 2, 3, 4, 5]
y1 = [1, 4, 9, 16, 25]

x2 = [1, 2, 4, 6, 8]
y2 = [2, 4, 8, 12, 16]
```

```
# use pylab to plot x and y : Give your plots names
plot1 = pl.plot(x1, y1, 'r')
plot2 = pl.plot(x2, y2, 'go')

# give plot a title
pl.title('Plot of y vs. x')
# make axis labels
pl.xlabel('x axis')
pl.ylabel('y axis')

# set axis limits
pl.xlim(0.0, 9.0)
pl.ylim(0.0, 30.)

# make legend
pl.legend([plot1, plot2], ('red line', 'green circles'), 'best', numpoints=1)

# show the plot on the screen
pl.show()
****************************************************
```

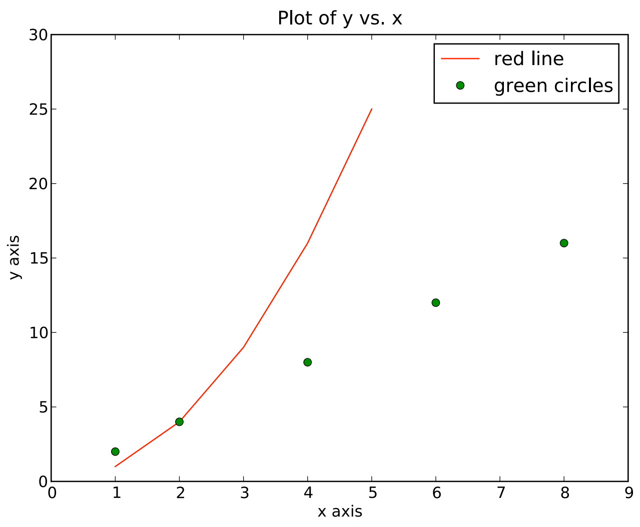

Figure 5: Plot made with `lineplotFigLegend.py`

3.3 Histograms

Histograms are very often used in science applications and it is highly likely that you will need to plot them at some point! They are very useful to plot distributions e.g. what is the distribution of galaxy velocities in my sample? etc. In Matplotlib you use the `hist` command to make a histogram. Take a look at the short macro below which makes the plot shown in Fig. 6:

```
***************************************************
# histplot.py

import numpy as np
import pylab as pl


# make an array of random numbers with a gaussian distribution with
# mean = 5.0
# rms = 3.0
# number of points = 1000
data = np.random.normal(5.0, 3.0, 1000)

# make a histogram of the data array
pl.hist(data)

# make plot labels
pl.xlabel('data')

pl.show()
***************************************************
```

If you don't want to see the black outlines between the bars in the histogram, try:

```
pl.hist(data, histtype='stepfilled')
```

This is how you make the plot in Fig. 7.

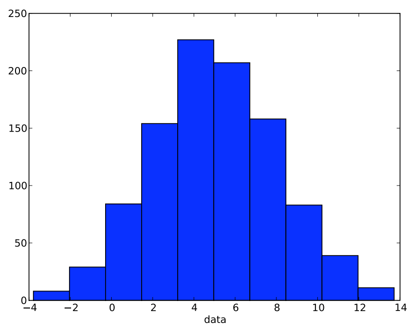

Figure 6: Plot made with `histplot.py`

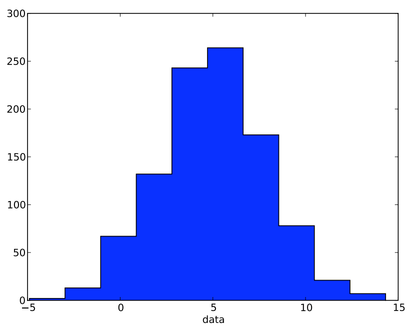

Figure 7: Plot made with `histplot.py`

3.3.1 Setting the width of the histogram bins manually

You can also set the width of your histogram bins yourself. Try adding the following lines to the macro `histplot.py` and you should get the plot shown in Fig. 8.

```
bins = np.arange(-5., 16., 1.)
pl.hist(data, bins, histtype='stepfilled')
```

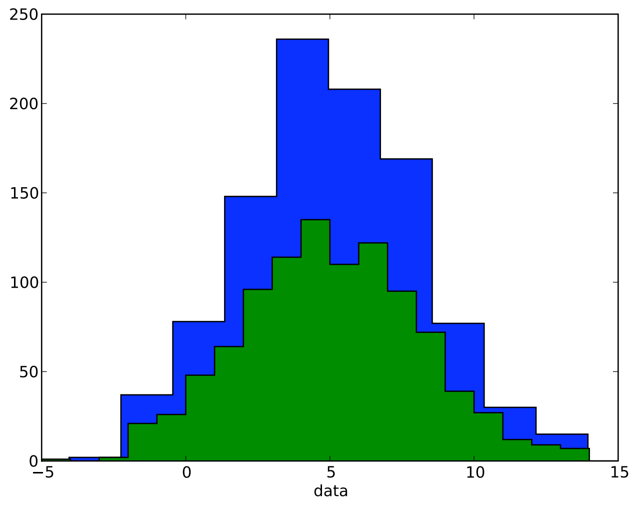

Figure 8: Manually setting the bin width for a histogram

4 Plotting more than one plot per canvas

Matplotlib is reasonably flexible about allowing multiple plots per canvas and it is easy to set this up. You need to first make a figure and then specify *subplots* as follows:

```
fig1 = pl.figure(1)
pl.subplot(211)
```

`subplot(211)` means that you will have a figure with 2 rows, 1 column, and you're going to draw in the top plot as shown in Fig. 9. If you want to plot something in the lower section (as shown in Fig. 10), then you do:

```
pl.subplot(212)
```

You can play around with plotting a variety of layouts. For example, Fig. 11 is created using the following commands:

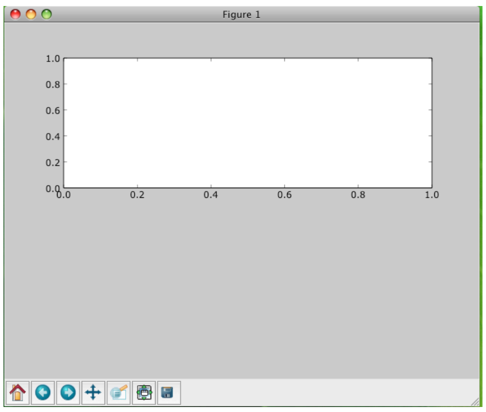

Figure 9: Showing subplot(211)

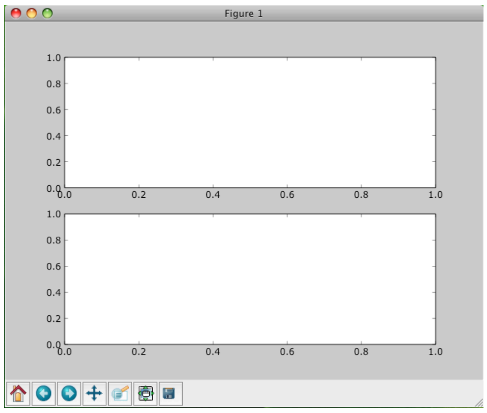

Figure 10: Showing subplot(212)

```
f1 = pl.figure(1)
pl.subplot(221)
pl.subplot(222)
pl.subplot(212)
```

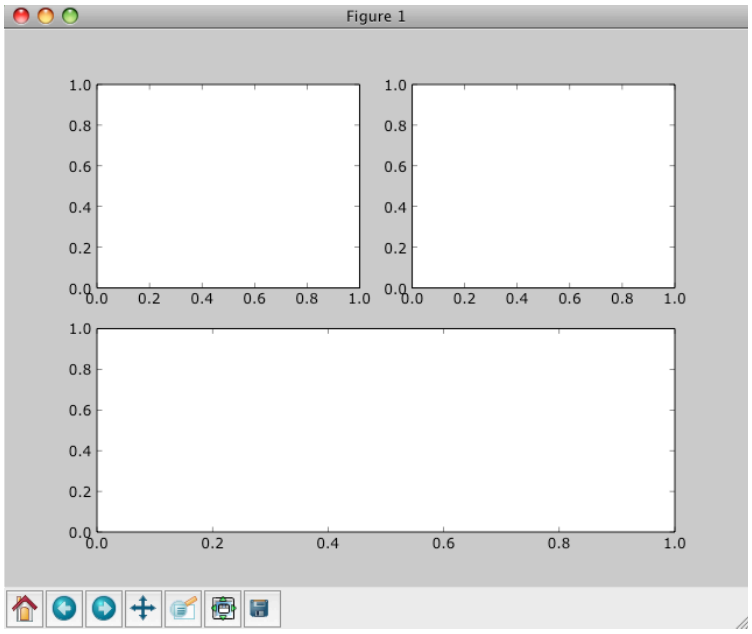

Figure 11: Playing with subplot

You can play with the sizes of the margins and gaps between the subplots using the command:

```
pl.subplots_adjust(left=0.08, right=0.95, wspace=0.25, hspace=0.45)
```

5 Plotting data contained in files

Often you will have data in ascii (text) format which you will need to plot in some way. In this section we will briefly discuss how to open, read from, and write to, files.

5.1 Reading data from ascii files

There are many ways to read data from a file in python. Here I am going to illustrate one simple method, but you should read the Python, Numpy and Matplotlib manuals to find out about the other ways which might sometimes apply better to you particular situation.
You can use Numpy to read in numerical values from a text file. For example, let's take the text file called `fakedata.txt` which contains the following data (2 columns of numbers):

```
****************************************************
# fakedata.txt
0 0
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
9 81
0 0
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
9 81
****************************************************
```

We can read this into a numpy 2D array and plot the second column against the first using the macro `readFileAndPlot.py` and shown in Fig. 12:

```
****************************************************
#readFileAndPlot.py

import numpy as np
import pylab as pl

# Use numpy to load the data contained in the file
# 'fakedata.txt' into a 2-D array called data
data = np.loadtxt('fakedata.txt')

# plot the first column as x, and second column as y
pl.plot(data[:,0], data[:,1], 'ro')
pl.xlabel('x')
pl.ylabel('y')
pl.xlim(0.0, 10.)

pl.show()
*****************************************************
```

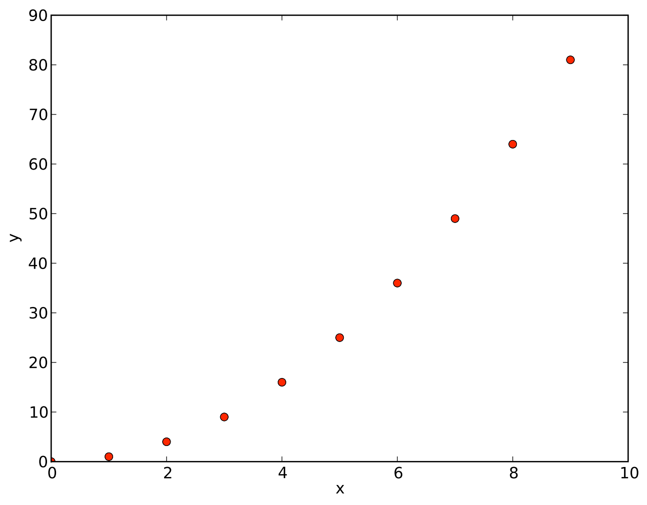

Figure 12: Plotting data from a file

5.2 Writing data to a text file

There are also various ways to write text to a text file. Here we show you one possible way to do it.
You first have to *open* a file to write to, then write what you want to the file, and then, do not forget to *close* the file! See the macro below, `writeToFile.py` to see how to do this:

```
*****************************************************
# writeToFile.py
```

```
import numpy as np

# Let’s make 2 arrays (x, y) which we will write to a file

# x is an array containing numbers 0 to 10, with intervals of 1
x = np.arange(0.0, 10., 1.)
# y is an array containing the values in x, squared
y = x*x

print ’x = ’, x
print ’y = ’, y

# Now open a file to write the data to
# ’w’ means open for ’writing’
file = open(’testdata.txt’, ’w’)

# loop over each line you want to write to file
for i in range(len(x)):
    # make a string for each line you want to write
    # ’\t’ means ’tab’
    # ’\n’ means ’newline’
    # ’str()’ means you are converting the quantity in brackets to a string type
    txt = str(x[i]) + ’\t’ + str(y[i]) + ’ \n’
    # write the txt to the file
    file.write(txt)

# Close your file
file.close()
****************************************************
```

You should then check that the file you created looks as you would expect! A quick way to do this on the Linux/Unix terminal command line is:

```
$ more testdata.txt
```

This command will print out the contents of your file to the screen and you will quickly be able to check if things look right!