Plotting with Matplotlib

Author: Louise Dash (louise.dash@ucl.ac.uk) Last updated: 22.10.2015

Learning objectives

By the end of this session, you should

- be able to produce a simple plot using the matplotlib module
- know how to produce a plot in a suitable format for your lab book and formal report

Introduction

Using matplotlib is a great way to interact with and present your data. In this session we'll go through how to plot functions and data files, and how to best present the data. In following sessions we'll look at how to analyse data graphically using least squares fits, and how to interact with data.

Why not just plot the data with Excel?

Once you get used to it, you'll find that Matplotlib is just as easy to use (if not easier) than Excel graphs, especially when you need to include error bars and/or mathematical notation. Moreover, the default plot produced by Matplotlib nearly always looks better than the default Excel plot!

The first thing we need to do is to import the matplotlib pyplot module. We'll also need to import numpy, just as before. Remember, to run each cell you need to press `SHIFT+ENTER`.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
# The next line tells Ipythonnotebook to put the plots in a notebook, instead of
a separate window
# You only need it if you're using matplotlib with an Ipython notebook.
%matplotlib inline
```

We also need some data to plot. Even if we're plotting a function, we need to calculate that function at a discrete set of points. We'll use one of the numpy array functions we met last week, linspace, to generate numbers for the x-values, and then calculate the corresponding y-values for a simple function - in this case $\sin(x)$.

In [2]:

```
x = np.linspace(0,2*np.pi,100) # 100 points between 0 and 2pi
y = np.sin(x)
```

Obviously our plot also needs axis labels and a title. Note how we enclose text strings inside quotation marks here.

In [3]:

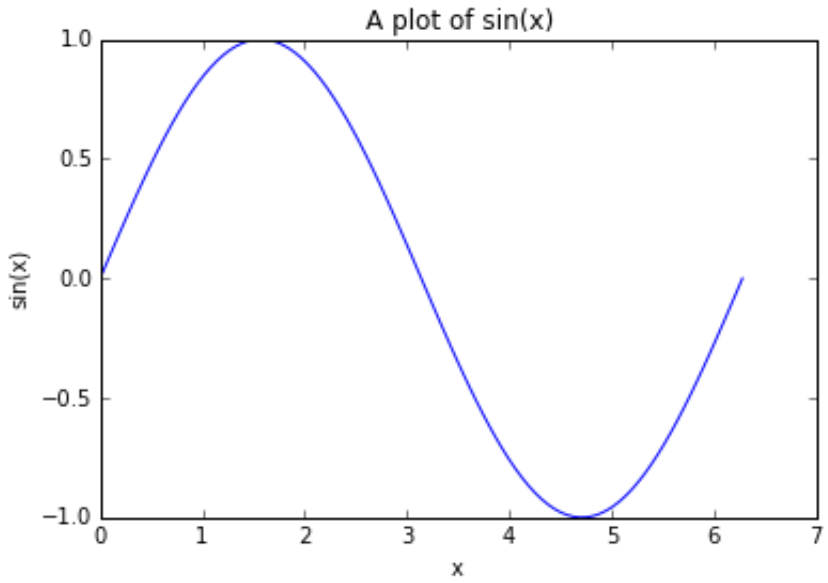
```
# now for the plot itself
plt.plot(x,y) # plot the array "x" on the x axis and "y" on the y-axis
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.title('A plot of sin(x)')

# IMPORTANT: If you're working with stand-alone code rather than a notebook,
# you'll also need this next line to tell matplotlib to actually show the plot on
the
# screen. If you're using an Ipython notebook, you don't need it.

# plt.show() # uncomment this line if you're not using an Ipython notebook
```

Out[3]:

```
<matplotlib.text.Text at 0xa1d5be0>
```



Remember, this looks like a continuous function, but in reality it's a series of straight line segments between discrete points. Try changing the number of points from 100 to 10 to see the difference!

Plotting two (or more) lines together

Sometimes we may want to plot two lines together on the same graph. To do this, we just add a new plot command for the new line. When you have more than one line, you'll generally need a legend as well. Look at this code to see how to do this:

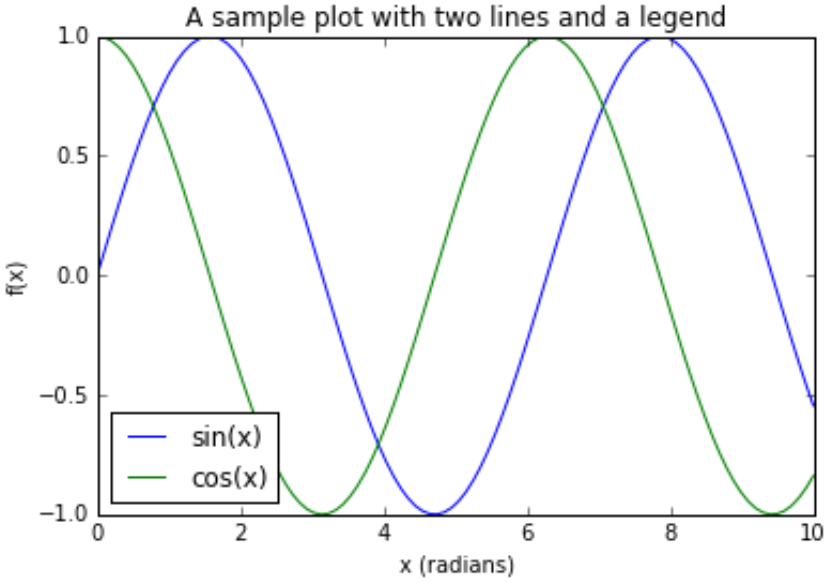
In [4]:

```
x = np.linspace(0,10,100) # set up array of 100 x-values from 0-10
y1 = np.sin(x)
y2 = np.cos(x)

plt.plot(x,y1, label="sin(x)") # plot the first line and set the label
plt.plot(x,y2, label="cos(x)") # plot the second  line
plt.xlabel('x (radians)') # put a label on the x axis, including units
plt.ylabel('f(x)') # put a label on the y axis
plt.title('A sample plot with two lines and a legend') # a title for the plot
plt.legend(loc="best")  # this will try and guess the best location for the legend
# or try "lower left, center right, etc~
#plt.legend(loc="upper right")
```

Out[4]:

```
<matplotlib.legend.Legend at 0xa558710>
```



Loading a dataset from a file

Usually we'll want to plot some data we already have, e.g. from a lab experiment. We'll need a way to get this data into a plottable format, i.e. arrays like the ones we created for x and y above. There are a few ways to do this.

- Entering the data by hand straight into Python, using the np.array() function. This is generally inefficient and you won't want to do this.
- Entering the data into another source and then loading it into Python:
  - You can use Excel, and then save the Excel file as a .csv (Comma Separated Values) file, which is a plain text format. This is a good choice if you want to do some data manipulation before analysing it.
  - You can use a text editor like Notepad to enter the data, and save it as a .txt file. Useful if you only have a few data points and don't want to fire up the full force of Excel.

For this example, I created a text file called "sampledata.txt" in Notepad with three columns of data, separated by tabs, which represent the x-values (in this case voltage), the y-values (in this case current) , and the error in the y-values.

Make sure you've downloaded this file from Moodle, and put it in the same directory as this notebook. Open the file in Notepad so you can compare the data in the file with the data imported in the code cell below.

To get this data into Python, the easiest way is to use the `np.loadtxt()` function , which we already met last week. This time though, we have *three* columns of data, so now we need to "unpack" the data into three separate arrays, like this:

In [5]:

```
#unpack=True unpacks each column into a separate array.
xdata, ydata, yerror = np.loadtxt("sampledata.txt", unpack=True)
print "our xdata: ", xdata
print "our ydata: ", ydata
print "errors on the y-values: ", yerror
```

```
our xdata:  [  0.  10.  20.  30.  40.  50.  60.  70.  80.]
our ydata:  [  0.   11.3  20.4  28.7  39.8  50.7  59.5  71.2  80.5]
errors on the y-values:  [ 5.  5.  5.  5.  5.  5.  5.  5.  5.]
```

The full documentation for `np.loadtxt()` is here: http://docs.scipy.org/doc/numpy/reference/generated/numpy.loadtxt.html (http://docs.scipy.org/doc/numpy/reference/generated/numpy.loadtxt.html) Read it to find out how to ignore rows with header information, change the delimiter from tab to comma, or pick which columns you want to load. By default, lines starting with the comment # character are ignored, so it's useful to use this for column headers.

Specifying the plot style

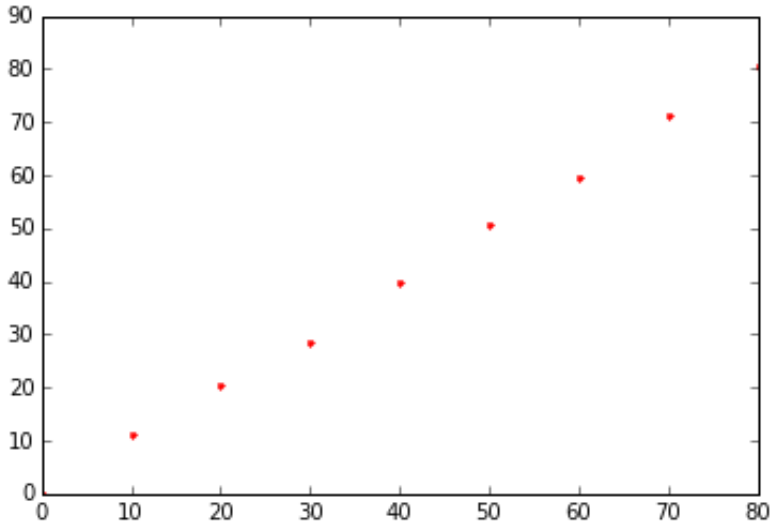
When we're plotting data, we usually want to plot it as points rather than as a line. By default, matplotlib will plot a line, but it's easy to plot points instead. You can specify both the colour and style of the datapoints using a short abbreviation - in this example red (r) points (.) - note that this is enclosed in quotation marks:

In [6]:

```
plt.plot(xdata,ydata, 'r.' ) # 'r.' specifies small red points
```

Out[6]:

```
[<matplotlib.lines.Line2D at 0xa6179e8>]
```



These abbreviations are reasonably intuitive, so they're fairly easy to remember:

- Colours:
  - b: blue
  - g: green
  - r: red
  - c: cyan
  - m: magenta
  - y: yellow
  - k: black
  - w: white
- Alternatively, you can specify an exact colour using an RGB hex code, eg " color = '#eeefff' " or a standard html name, eg: " color = 'red' ". If you do this, note the AmEng spelling of "color"!
- Marker style - there are lots of these, you may want to try
  - '-' solid line style
  - '--' dashed line style
  - '-.' dash-dot line style
  - ':' (a colon) dotted line style
  - '.' point
  - 'o' circle
  - 'v' triangle_down
  - '*' star
  - '+' plus
  - 'x' x
- There's a full list of the possibilities here: http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot (http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot)

How to plot error bars

We didn't include any error bars in the plot above. In fact, to include errorbars we need to use the errorbar() command *instead of* the plot() command (this is not necessarily intuitive, as you might think you'd use the errorbar() *and* plot() commands).

Note how we do this:

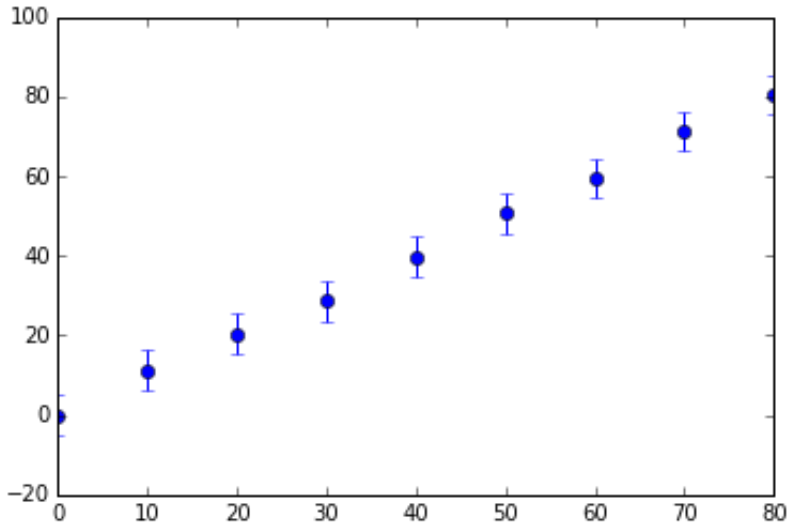
- We specify the xdata and ydata as before
- The y-errorbars are specified with the yerr keyword (hopefully you can thus use logic to deduce what to do if you want x-errorbars)
- We have a fmt statement to specify the point style (here blue circles). If you leave this out, the default is to draw a straight line between the datapoints, and in general you *don't* want this.

In [7]:

```
plt.errorbar(xdata,ydata,yerr=yerror,fmt='bo')
```

Out[7]:

```
<Container object of 3 artists>
```



To see how we can combine these, we'll:

- Add a straight line x=y through the datapoints (we'll come on to how to *fit* a line through the data next session)
- Specify the range for the x and y axes
- Add all the other things our plot needs: labels, titles, legends, gridlines, etc
- Note that (in general), the order of the plt.* commands doesn't matter - but they are executed in sequence, so it's aesthetically better to have the grid command before the data, so that it gets plotted underneath, etc., as in the example below.

In [8]:

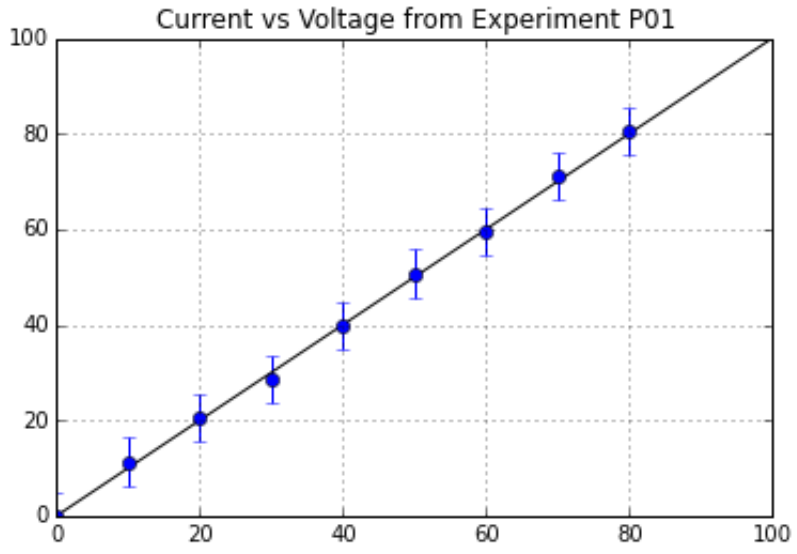
```
# first generate data for the straight line y = x
# we'll create two separate arrays for this
xline = np.linspace(0,100,5)
yline = xline

# Now for the plot commands.
plt.grid(True) # Add gridlines
plt.plot(xline,yline, 'k-') # 'k-' specifies a black line
plt.errorbar(xdata,ydata,yerr=yerror,fmt='bo') # plot the data with errorbars and
blue circles
plt.xlim(0,100) # set xrange to be from 0 to 100
plt.ylim(0,100)# set the y range to be from 0 to 100
plt.title('Current vs Voltage from Experiment P01') # a suitable concise title

# outputting to a file - see the section below
#plt.savefig('Myplot.png') # saves to a PNG file
#plt.savefig('Myplot.png', dpi=300) # saves to a PNG file with a higher resolution
#plt.savefig('Myplot.pdf') # saves to a PDF file
```

Out[8]:

```
<matplotlib.text.Text at 0xb597e10>
```



Outputting your plot to a file

To save your plot to a file for printing or including in another document (eg. Word), use the savefig() command. The full documentation for this is here:
http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.savefig
(http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.savefig)

Note that you need to put the savefig command in the same cell as the other plot commands if you're using a IPython Notebook. Some sample commands are in the cell above, commented out - try uncommenting each of them in turn to see the results. The files will be saved in the same directory as your IPython Notebook.

You can save in a number of formats, but generally you'll want to save as either a .png or .pdf file. The file type can be autodetermined from the filename extension, as in these examples. You'll probably also find that all the default settings work OK, but if you want to change the resolution of a png (for better quality), try changing the dpi (dots per inch) setting, as above.

Preparing plots for your lab book or formal report

When you're using this in lab classes, remember there are specific rules and guidelines for how your plots should look, depending on whether you're preparing your plot to stick into your lab book or for part of a formal report. Here are some basic checklists to help you make sure you haven't forgotten anything:

Lab book checklist

Make sure:

- The axes are labelled, complete with units
- You have gridlines to make the data easier to decipher
- If appropriate, you have included error bars. If you don't include error bars because they'd be smaller than the data symbols, say so in the caption.
- If you have more than one dataset on the same plot, include a legend, and make sure the legend is positioned so that it doesn't cover up important data!
- All components should be an appropriate size for printed output (generally with Matplotlib the default settings are good, but check this in case!)
- You've chosen appropriate line and point styles for a printout - if you are printing the plot on a black and white printer, don't rely on different colours to distinguish between datasets!
- You will need a (handwritten) caption in your lab book. However it's good practice to include a title on your plot as well so that when you look at the *electronic* version of the plot you know what it refers to! If you want to, you can crop this off the final printout with scissors before sticking it into your lab book.

Formal report checklist

- Axes are labelled, complete with units
- Grid is present
- If appropriate, you have included error bars. If you don't include error bars because they'd be smaller than the data symbols, say so in the caption.
- If you have more than one dataset on the same plot, include a legend, and make sure the legend is positioned so that it doesn't cover up important data!
- All components are an appropriate size for the two-column formal report layout - you may need to change the font sizes.
- Appropriate line and point styles so that the data is clearly legible when it's printed at the size it appears on the formal report.
- Don't include a title for a formal report - instead use a figure caption within the document.

Assessed task for this session

This is an exercise in plotting data from a laboratory experiment.

The data is from an x-ray diffraction experiment similar to one that many of you will do next year in PHAS2400 (Experiment A10). You do not need to know anything about x-ray diffraction to plot this data, but if you are interested you can read about the theory here: http://en.wikipedia.org/wiki/Bragg%27s_law (http://en.wikipedia.org/wiki/Bragg%27s_law)

What you need to do:

- Download the two data files from the Moodle page for this session. These were generated in Microsoft Excel and saved in .csv (comma separated values) format. In both files, the first column (x-data) is twice the diffraction angle (2 theta) and the second column (y-data) is the number of counts recorded per second.
- Create a new IPython Notebook. Remember to start with a suitable title, and use text cells to explain what you're doing as you go along. You'll also want to include some comments in the code cells too.
- Import the data into your Notebook like we did for the sample file above, and plot the data as two lines on a single plot. The line styles, colours, etc are up to you - experiment as much as you like, but choose something appropriate for the final version.
- You'll need to consider whether or not to use a continuous line, data points (and if so which style/size), or both. Which is most appropriate here, and why? Briefly explain your choice in a text cell.
- Include axis labels, a legend, and a title. Again, the choice of appropriate fonts, sizes, etc is up to you.
- Use `savefig` to save the plot in a suitable form for printing out and sticking into a lab book. You will upload this figure file as part of the assignment.

Hints:

- Look at the data file in Notepad to see what format it is in before trying to import it (if you open it in Excel, it just looks like a spreadsheet, which doesn't really help you decipher the format!)
- Read the loadtxt help at http://docs.scipy.org/doc/numpy/reference/generated/numpy.loadtxt.html (http://docs.scipy.org/doc/numpy/reference/generated/numpy.loadtxt.html) carefully, in particular how to set the delimiter.
- You can find information on typesetting maths / greek letters in Matplotlib here: http://matplotlib.org/users/mathtext.html (http://matplotlib.org/users/mathtext.html)