

# Python Reference

## Introduction

These notes contain commands useful for analysing and plotting data using the Python programming language, together with the Numpy and Matplotlib libraries.

## Installation

Most easy using Enthought Canopy:

<https://www.enthought.com/products/canopy/>

You can get the free express version (or upgrade to the free academic version if you wish, though we won't use the extra packages in the academic version in this course).

This program is new - previously I recommended its predecessor, the enthought python distribution (free version).

## Running

If you have Canopy, run it. Start up the editor, the go to "file", then click the new file button in the top left. Save it and run form the buttons at the top.

If using the older enthought python distribution, go to the enthought directory and run the "IDLE" program. To write a program go to "file", "new window".

Save it then run from the top menu.

## Import Pylab

We will make heavy use of the Numpy (numerical python) and matplotlib (plotting) packages.

Load them both together by typing:

```
import pylab
```

Then to use a command from either package, just preface it with pylab. (pylab followed by a full stop).

So for example, the plot command (from matplotlib) would be executed by typing

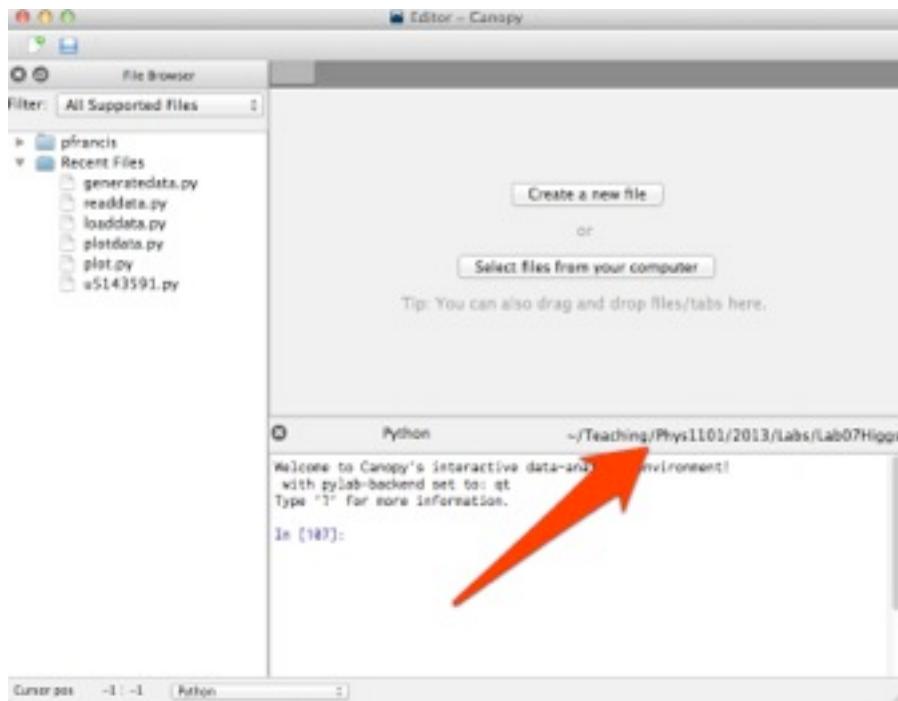
```
pylab.plot(...)
```

You may also want to import some other packages, like "math" to do maths, or "os" to do operating system commands.

## Reading in Data

To read in any sort of data, you first need to make sure that python's "current working directory" is the one in which your data is stored.

In Canopy, you change the current working directory to wherever you have stored the binary by clicking on the directory listed below the program window: see where in the picture below:



In the older enthought python distribution, the easiest way to do this is to put your program file and the data in the same directory.

## Binary Data

To read in binary data, use:

```
my_data_array = pylab.fromfile("datafile.dat", dtype=float)
```

(replace my\_data\_array by whatever you want to call the array in your program, and datafile.dat by the name of the file you want to read)

### .csv data

.csv files stand for “comma separated variables” - i.e. a text file with commas between the different elements. You can write these from Excel and most spreadsheet programs.

To read such a file, if it contains just numbers, use a command like:

```
my_data_array = pylab.loadtxt("dataset.csv", delimiter=",")
```

## Manipulating Arrays

The pylab.loadtxt and pylab.fromfile command load data into multi-dimensional arrays (numpy arrays). Here are some commands for manipulating them.

If necessary you can create a new array with a command like:

```
my_array = pylab.arange(0,10,1)
```

which will produce an array (0,1,2,3,4,5,6,7,8,9)

print my\_array

- will print it out (and is clever enough only to plot out the start and finish if it is big)

print my\_array.shape

- tells you the size and dimensions of your array.

To add up all the elements in an array, use the following command:

sum = pylab.sum(my\_array)

## Functions on arrays

If you apply a function to an array, it applies to each and every element in the array. For example

a+b will add element 1 of a to element 1 of b, element 2 of a to element 2 of b, etc. The result will be a new array of the same size as the ones you added together.

pylab.sin(my\_array) will compute the sine of every element in my\_array and put these values in a new array of the same size.

pylab.sqrt(my\_array) will compute the square root of every element in my\_array and put these values in a new array of the same size.

pylab.power(my\_array,n) will raise every element in my\_array to the power n.

A full list of functions can be found at:

[http://www.scipy.org/Numpy Functions by Category](http://www.scipy.org/Numpy_Functions_by_Category)

## Scientific Notation

To write a constant like  $6.67 \times 10^{-11}$ , type in 6.67e-11.

## Pulling out part of an array.

To pick out the  $n^{\text{th}}$  element of a 1D array, just put the element number in square brackets:

element = my\_array[n]

Note that in python, the first element of any array is given the number n=0 (not n=1).

Pick out a range with a colon:

range = my\_array[3:6]

If you have a 2D array, you can pick out a given element with a command like:

element = my\_array[4,5]

If you want to pull out a whole column, use a command like:

first\_column = my\_array[:,0]

or

second\_column = my\_array[:,1]

If you want to pull out elements based on some condition (i.e. every number greater than 10), use the following command to create an index array:

```
index_array = my_array<10
```

This gives a list of “TRUE” and “FALSE” values depending on whether each element met the criterion (in this case was less than 10). To pick out only the chosen elements, and put them in a new array, use something like:

```
new_array = my_array[index_array]
```

## Plotting

We use Matplotlib (included in pylab) to plot.

The normal sequence is to put in a command to plot a graph, the commands to modify it (like putting labels on the axes) then finally we use the pylab.show() command to display it.

### Scatter plots or lines

pylab.plot(x,y,’+b’) is a typical command. X is an array containing the x values, y is an array containing the y values, and ‘+b’ is a format specifier.

+ indicates plot crosses.

- indicates plot a line

O indicates plot circles

b indicates plot in blue, r in red, g in green, k in black etc.

If instead of giving pylab.plot two arrays (x and y) you just give it one (e.g. pylab.plot(my\_array,”b-“)) it will plot numbers in the array as y-values, and the position in the array as x-values.

### Histograms

To take some data, turn it into a histogram and plot it, use the pylab.hist command:

```
pylab.hist(my_array,bins=20,range=(10,30))
```

will plot a histogram of the data in my\_array, broken up into 20 bins over the range of values 10-30.

The bins and range commands are optional - the program will pick its own if you don’t set them explicitly.

If you want to actually see the numbers for how many counts were found in each bin, see the pylab.histogram command -

```
print pylab.histogram(my_array,bins=20,range=(10,30))
```

which will return two lists - firstly a list of counts, and secondly a list of the edges of the bins. To just get the former, add [0] on the end - i.e.

```
print pylab.histogram(my_array,bins=20,range=(10,30))[0]
```

## Modifying a plot.

Modify a plot by adding some of the following commands between the `pylab.plot` command and the `pylab.show()` command.

You can add axis labels or a title with the following commands

```
pylab.ylabel("y axis")
pylab.xlabel("x axis")
pylab.title("My Data")
```

To change the range over which the data is plotted, you can use the following command:

```
pylab.axis([0,6,-2,2.5])
```

which will set the x-axis to run from 0 to 6, and the y axis to run from -2 to 2.5.

You can add a second set of data or a line to your first plot by putting a second `pylab.plot` command in (useful for comparing a model to data, for example).

## Plotting with error bars

If your x-values are in an array called `x`, your y-values in an array called `y`, and you have uncertainties in `y` only in an array called `err`, you can plot uncertainties with the following command:

```
pylab.errorbar(x, y, yerr=err, fmt='ko')
```

(the `fmt` sets the format - same options as for “plot” above).

## Showing and saving your plot

To display your plot:

```
pylab.show()
```

You can save it in various formats from a button on this window.

Alternatively, use a command like this:

```
pylab.savefig("pretty.jpg")
```

To save in whatever format you like (python works out the format from the extension - in this case it will be a jpeg).

## Calculating chi-squared

You first need to calculate your model: e.g.  
`model = pylab.sqrt(times*b)+c`

Then calculate residuals: e.g.  
residuals = model-data

You then produce a new array containing the residuals squared divided by the errors squared:

```
normresiduals = (residuals*residuals)/(errors*errors)
```

Finally you need to sum all the elements in this new array - to get the chi-squared value:  
chisq = pylab.sum(normresiduals)

To work out reduced chi-squared, divide this by the number of degrees of freedom n.

To work out the p-value,

```
import scipy  
pval = scipy.stats.chi2.cdf(chisq,n)
```

## For Loops

Python loops are given a list, and take each value in that list in turn.

First, you need to set up a list - typically using the pylab.arange command, For example,  
list = pylab.arange(2.3,7.5,0.2)

gives a list that is

```
[ 2.3 2.5 2.7 2.9 3.1 3.3 3.5 3.7 3.9 4.1 4.3 4.5 4.7 4.9 5.1  
 5.3 5.5 5.7 5.9 6.1 6.3 6.5 6.7 6.9 7.1 7.3]
```

All commands that follow the colon and are indented by the same amount are inside the loop. For example:

```
for a in pylab.arange(0.0,10.0,2.0):  
    b=a**2  
    print a,b
```

will produce output

```
0.0 0.0  
2.0 4.0  
4.0 8.0  
6.0 12.0  
8.0 16.0
```