# Python lab session 2

Dr Ben Dudson, Department of Physics, University of York

$4^{th}$ February 2011

**Important**: Remember to create a new document to write your answers in and put your name at the top. For each task, put your code and the result in a different section. At the end of the lab session, email this document to `phys-python@york.ac.uk`. Anything submitted after 14:00 **today** will not be marked. To avoid disappointment, please double-check that you have attached the document.

## NumPy and arrays

This lab we'll start using arrays and plotting, and in the process get some more practice at writing functions. Before we can use any NumPy arrays or functions, we first need to import them. There are several ways to do this, but for now I recommend putting this at the top of all your programs:

```
from numpy import *
```

This imports several functions for creating arrays, including arange, zeros, ones, and random.random. See the help page for each one to see how to use them, for example

```
help(arange)
```

The examples here will use linspace, which has 3 arguments: the minimum value, maximum value, and how big the array is.

```
a = linspace(min, max, n)
```

## Using arrays

Once we've got arrays of data, we can manipulate them all at once using NumPy's mathematical functions. The following table lists some of the more commonly used ones

| Function | NumPy | Inverse | NumPy |
|---|---|---|---|
| $\sin(x)$ | sin(x) | $\sin^{-1}(x)$ | arcsin(x) |
| $\cos(x)$ | cos(x) | $\cos^{-1}(x)$ | arccos(x) |
| $\tan(x)$ | tan(x) | $\tan^{-1}(x)$ | arctan(x) |
| $e^x$ | exp(x) | $\log_e(x)$ | log(x) |
| $10^x$ | 10**x | $\log_{10}(x)$ | log10(x) |
| $x^2$ | x**2 | $\sqrt{x}$ | sqrt(x) |
| $\|x\|$ | fabs(x) | | |

**Note**: All trigonometric functions use radians, not degrees

There are also the mathematical constants pi ($\pi = 3.14159\ldots$) and e ($e = 2.71828\ldots$), so **print** sin(pi/2) will produce the result 1.

We can treat arrays like we did single numbers, send them to functions and return them as results. We could define a function to calculate $\sin^2(x)$:

```
from numpy import *

def sin2(x):
    return sin(x) ** 2
```

This function can be used to calculate results for a single number at a time, or for a whole array of numbers:

```
print sin2(0.2)
```

produces the result 0.0394695029986, whilst

```
print sin2(linspace(0,pi,4))
```

produces

```
[  0.00000000e+00   7.50000000e-01   7.50000000e-01   1.49966072e-32]
```

Note that the final point ($\sin^2(\pi)$) isn't quite zero. This is due to errors in calculating with floating point numbers. Often the results you get will be very close to the actual result, but out by a small amount. This is called **rounding error**, and is why it's a bad idea to compare if two floating point numbers are equal. If you had some code

```
if sin2(pi) == 0.0:
    print "Result is zero"
else:
    print "Result is not zero!"
```

This would print "Result is not zero!" because it compares 0.0 against $1.49966072e-32$ and finds that they're not equal. Instead, what you should do with floating point numbers is check if they're close enough. Rather than

```
if a == b:
  print "equal"
```

you should use

```
if fabs(a - b) < 1.e-8:
  print "Close enough"
```

Note: It's perfectly fine to compare two whole numbers (integers) to each other, since there is no rounding error. This comes back to computers handling whole numbers in a different way to floating point ones.

**Task 1** Mathematical functions (15%)

Write a program to create an array $x$ with 11 values going from 0 to $2\pi$, and then print the values of $x^2 \cos(x)$. This should produce the result

```
[  0.           0.31938711   0.48798008  -1.09795518  -5.11019372
  -9.8696044  -11.49793587  -5.97775596   7.80768125  25.87035571
  39.4784176 ]
```

**Task 2** Mathematical functions of two variables (15%)

Write a function to calculate a Gaussian

$$G(x, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-x^2/\sigma^2}$$

You can start with the following code

```
from numpy import *

def gaussian(x, sigma):
    #####

print gaussian(linspace(-2,2,11), 1)
```

where as usual you should replace ##### with your code. This should give the result

```
[ 0.00730688  0.03084013  0.0945205   0.21035924  0.33995619  0.39894228
  0.33995619  0.21035924  0.0945205   0.03084013  0.00730688]
```

# Plotting

Plotting is a good way to help understanding what's going on with your data, rather than staring at tables of numbers. In this course we'll be using the Matplotlib package, so before you can use this you'll need to add this to the top of your program

```
import matplotlib.pyplot as plt
```

To plot some data, plt.plot(x,y) needs an array for the $x$ position of each point, and an array for the $y$ position. If you only give it one array then it will treat this as the $y$ position and use the position in the array as the $x$ value. For example, the following will plot $sin(x)$ from 0 to $2\pi$:

```
from numpy import *
import matplotlib.pyplot as plt
x = linspace(0, 2*pi, 21)
p = plt.plot(x, sin(x))
plt.show()
```

The plt.show() at the end is needed to make Matplotlib display the plot.

---

**Task 3** Plotting (20%)

Adapt this to plot the sinc function $\sin(x)/x$ for $x$ between $-4\pi$ and $4\pi$. How do you make sure that $x$ doesn't have a point at 0? Put your code and plot into your lab document.

---

To add axis labels to our plot, Matplotlib has xlabel and ylabel functions, and title function to set the plot title.

```
from numpy import *
import matplotlib.pyplot as plt
x = linspace(0, 2*pi, 21)
p = plt.plot(x, sin(x))
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.title('Example matplotlib plot')
plt.show()
```

To save your plots, either click on the save icon at the bottom of the plot, or you can do it automatically by using the plt.savefig function **before** calling plt.show()

```
plt.savefig("figure.png")
```

where "figure.png" is the file you want to save it as. The format is determined by the ending, so ".png" means save as a PNG image, ".pdf" means save as a PDF file.

---

**Task 4** Fancy plotting (15%)

Use the function you wrote in task 2 to produce a plot of a Gaussian $G(x, \sigma)$ between $x = -2$ and $x = 2$ for $\sigma = 1$. Add labels 'x' on the x-axis and 'G(x,sigma)' on the y axis.

Save your plot, and paste it into your lab document, along with your code

**Note:** If you know LaTeX, you can use symbols like '$\sigma$' in labels and text.

---

**Task 5** Multiple plots (15%)

If you want to put several plots on a graph, just call plt.plot() several times before calling plt.show(). You can also add a legend by calling legend() before show()

```
plt.plot(x, sin(x), label='sin(x)')
plt.plot(x, cos(x), label='cos(x)')
plt.legend()
plt.show()
```

Plot three Gaussians with $\sigma = 0.5$, 1.0 and 2.0 on the same graph with a legend.

---

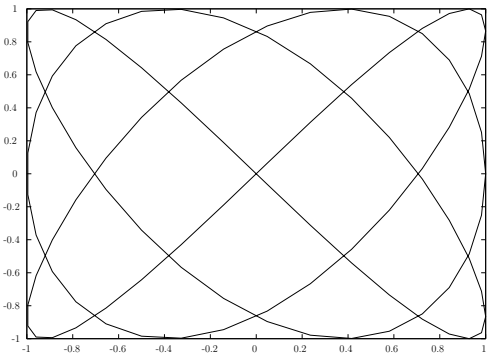
# Parametric equations

Parametric equations are where both the $x$ and $y$ values are a function of a third variable usually called $t$. This could be the position of an object on a surface as it moves in time.

A nice example of a parametric equation is Lissajous figures. These are given by

$$x = \sin(at + \delta) \qquad y = \sin(bt)$$

and look something like the following:




Lissajous figure for $a = 3$, $b = 4$ and $\delta = \pi/2$

---

**Task 6** Lissajous figure (20%)

Write a function to calculate and plot a Lissajous figure, given the parameters $a$, $b$ and $\delta$. Start with the following code:

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

def lissajous(a, b, delta):
    #####

lissajous(3, 4, pi/2)
plt.show()
```

You should get a decent result using 100 points between 0 and $2\pi$. As always, put your code and graph into your lab document.

---

# Extra tasks

That's the end of the marked part of the lab. If you have time though, you might like to try the following more "fun" problems.

First, animations. Matplotlib can be used to make simple animations, by changing the $x$ and $y$ data in plots.The time module can be used to introduce delays between each plot. For example, the following makes an animation of a sin wave

```
from numpy import *
import matplotlib.pyplot as plt
import time

plt.ion()

x = linspace(0, 2*pi, 100)
line, = plt.plot(x, sin(x))
for b in linspace(0,4*pi,40):
    line.set_ydata(sin(x + b))
    plt.draw()
    time.sleep(0.1)
```

Note that you need to turn on interaction using plt.ion(), there's no need to call plt.show(), and plt.draw() re-draws the graph after the data is changed.

**Task 7** Extra task: Lissajous animation

Adapt the above code to create an animation of a Lissajous figure by varying $b$ between 3 and 6 in 50 steps, using $a = 3$ and $\delta = \pi/2$. In addition to changing the $y$ data, you will need to change the $x$ data using line.set_xdata.

Finally, if you're waiting around: why not try out more Matplotlib examples. The web is full of examples and tutorials on Matplotlib and Python in general.