Python lab session 3

Dr Ben Dudson, Department of Physics, University of York

11^{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.

This lab session goes over some of the things you have seen before such as creating functions from flow charts, creating arrays and plotting. It also introduces some new topics like 2D arrays, complex numbers and indexing arrays. At the end of the lab you should be able to combine these things to plot a picture of a fractal (the Mandelbrot set).

Remember to import the NumPy and Matplotlib modules by putting

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

at the top of all your programs.

Plotting

1D arrays can be plotted using plt.plot, which can also plot symbols rather than lines.

```
x = linspace(0, 2*pi, 21)
plt.plot(x, sin(x), 'bo', label='sin(x)')
plt.plot(x, cos(x), 'r+', label='cos(x)')
plt.legend()
plt.show()
```

Here 'bo' means "blue circles" and 'r+' means "red + symbols". 2D arrays can be made into contour plots using contour, filled contour plots with contourf and surface plots using plot_surface. The following makes a filled contour plot of $f = e^{-x^2}\sin(y)$

```
x,y = mgrid[-2:2:20j, 0:(2*pi):20j]
f = exp(-x**2) * sin(y)
plt.contourf(f)
plt.show()
```

See the Matplotlib website at `http://matplotlib.sourceforge.net/`

1 Plotting revisited

As a brief revision of last week's plotting, you're going to write a program to visualise Taylor expansions to $\sin(x)$:

$$\sin(x) \simeq \sum_{k=0}^{n} (-1)^k \frac{x^{2k+1}}{(2k+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots$$

The first step is to note that each term can be calculated from the last one: if we write

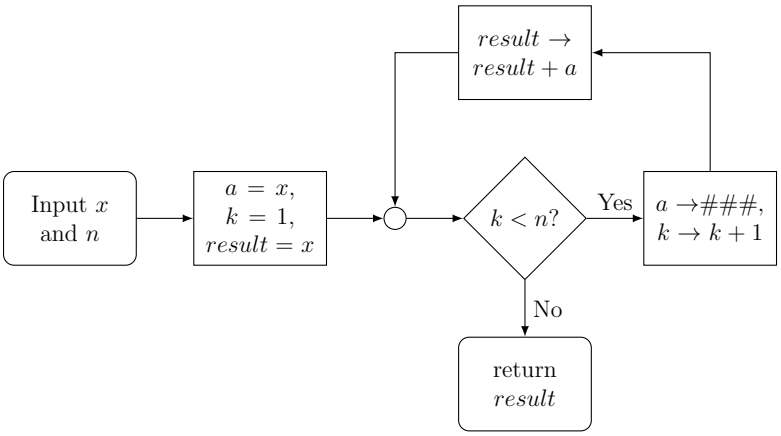
$$\sin(x) \simeq \sum_{k=0}^{n} a_k \qquad a_k = (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

then

$$a_k = -\frac{x^2}{(2k+1)\,2k} a_{k-1}$$

Task 1 Plotting approximations to $\sin(x)$ (20%)

a) Write a function called mysin which calculates an approximation to $\sin(x)$ using the first n terms $a_0, a_1, \dots, a_{n-1}$. You can use the following flowchart as a guide:



where you'll have to fill in the details for the $a \rightarrow$### calculation.

b) This function should work if x is an array. Use this to calculate the approximation to $\sin(x)$ for $n = 1, 2, 3, 4$ using 100 points in x between -2π and 2π. Plot these on the same graph along with the result of the NumPy sin function, with legend and axis labels. Use str() to convert a number to text, and join text using '+'.

Put the graph and your program in your lab document.

2D arrays

A convenient way to make 2D arrays (or 3D,4D etc) is to use NumPy's mgrid[] method (note the square brackets).

```
x,y = mgrid[0:5:20j, -1:2:40j]
```

produces 2 arrays x and y, both with 20 points in the first dimension, and 40 in the second dimension. The x array has values going from 0 to 5, and the y array from -1 to 2.

Task 2 2D sinc function (10%)

Create two arrays x and y using mgrid[], both going from -2 to 2 and with 40 points in each dimension. Use these to plot a contour plot of the function $f(x, y) = \sin(r)/r$ where $r^2 = x^2 + y^2$. Put your figure and program into your lab document.

Complex numbers

Before going on to fractals, we need to look at how Python handles complex numbers. Fortunately this is very simple: to set $x = 1 + 2i$ in Python we just write

```
x = 1. + 2j
```

Note:

- Python uses 'j' for $\sqrt{-1}$ rather than i in mathematics
- You must put a number before the 'j', otherwise Python will think you mean a variable called j. Hence if you want to write the number i in Python you need 1j rather than j.

Using the NumPy module, it's possible to calculate things like sin, cos and tan of complex numbers. Sometimes you need to make sure NumPy knows it should be dealing with complex numbers, for example **print** sqrt(−1.) will give an error nan (Not A Number), but **print** sqrt(−1. + 0j) will print the correct value 1j.

To create a complex number z from the real and imaginary parts (re and im respectively), you can use

```
z = re + im * 1j
```

To get the real and imaginary parts of a complex number, Python has the real() and imag() functions. To get the angle $\tan^{-1}(imaginary/real)$, Python has the angle() function. Example:

```
x = 5.
y = 3.
```

```
c = x + y*1j
print c              -> "(5+3j)"
print real(c)        -> "5.0"
print imag(c)        -> "3.0"
print angle(c)       -> "0.540419500271"
```

Task 3 Complex functions (20%)

a) Write a function called func to take two arguments (x and y), convert these into the real and imaginary components of a complex number z, then calculate

$$f(z) = \left(z^2 - 1\right)(z - 2 - i)^2 / \left(z^2 + 2 + 2i\right)$$

Your function should return the **angle** of $f(z)$. As a check, func(1., 1.) should give the answer 0.927295218002

b) Use this function to produce a filled contour plot of $f(z)$ using plt.contourf(). On the x axis should be the real component between and on the y should be the imaginary component. Both should have 200 points between -3 and 3.

Save the plot, and put it along with your program into your lab document

Fractals

Fractals have produced some very well known images, many of which were only seen after computers made plotting them possible. The Mandelbrot set was first plotted in 1978, and is created by starting from $z = 0$ and repeatedly calculating $z \to z^2 + c$ where z and c are complex numbers. Some values of c will cause z to get bigger forever (called unbounded), whilst others lead to z staying within a particular size. For example, $c = 1$ leads to

$$z = 0 \to 1 \to 2 \to 5 \to 26 \to 677 \to \ldots$$

so z just gets larger and larger. On the other hand $c = i$ leads to

$$z = 0 \to i \to -1 + i \to -i \to -1 + i \to \ldots$$

so z just keeps going between $-1 + i$ and $-i$. Because the magnitude of z always stays less than or equal to $|-1 + i| = \sqrt{2}$, this is said to be bounded.

The first step is to write a function to determine if a given value of c leads to z being bounded or unbounded. Fortunately, it has been proven that if the magnitude of z ever gets larger than 2 then the sequence is unbounded i.e. z will keep getting bigger and bigger.

Task 4 Iteration of $z \to z^2 + c$ (20%)

Write a function which has an input c, and which repeatedly calculates $z \to z^2+c$ (starting at $z = 0$). Return the number of times you have to do this until the magnitude of z given by abs(z) becomes greater than 2. Since this might never happen, your function should also have a maximum number of times (start with 20) before returning.

When $c = 1$ your function should return 3 since after 3 steps z reaches 5 which is greater than 2. When $c = i$, your function should go to the maximum number of steps (20) and so return 20.

Now we have a function to test if a sequence is bounded or not, we need to create a 2D array a[x,y]. The x index is going to represent the real component of c, whilst the y index represents the imaginary component.

Task 5 Indexing arrays and the Mandelbrot set (30%)

---

a) Create 3 arrays: a 1D array for the x axis, which should contain 200 points between -2 and 2, a 1D array for the y axis with 200 points between -1 and 1, and a 200 by 200 2D array a to store the result in.

b) For each i and j between 0 and 199, create a complex number c = x[i] + y[j]∗1j. Use this as input to your function from task 4 and put the result into a[i,j].

c) Make a filled contour plot of this result, and put the graph in your lab document along with your program.

Extra tasks

That's the end of the assessed part of the lab, but if you have time you can try these more challenging problems

Task 6 Improvements to the $\sin(x)$ program

In task 1 you overplotted approximations to the sin function.

a) Can you turn this into an animation instead?

b) As x gets bigger the approximation gets worse. Can you think of a way to make your function give an accurate result for any input x? (without using a built-in function like sin!)

Task 7 Complex functions

You could modify task 3 to calculate different functions, and see the difference between functions with a singularity (where they go to ±infinity) and functions which don't.

Task 8 Improvements to the fractal program

The fractal program from task 5 could be developed in several ways

a) Add the ability to modify the range of the plot, to allow you to zoom in and out.

b) Create an animation, zooming in on a particular point. The Misiurewicz point at $-0.1011 + 0.9563i$ is a good one. To make the animation smoother, you might have to do the calculations at the start before animating them