Python lab session 5

Dr Ben Dudson, Department of Physics, University of York

25th 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.

Getting data from the web

The department of electronics at York has a weather station, which records things like temperature, rainfall, wind speed etc. The main website is at

`http://weather.elec.york.ac.uk/index.html`

but there are also daily records of weather available in text files from here:

`http://weather.elec.york.ac.uk/archive.html`

Using this data, it's possible to write a program in Python to make plots of quantities like temperature over the last few years. To read web pages, Python has a module called `urllib2` which makes getting data from the internet quite straightforward.

Getting data from a network address works like reading from a file, so to fetch the data for November 2010:

```
import urllib2

addr="http://weather.elec.york.ac.uk/data/vaisala/archives/1110.txt"
f = urllib2.urlopen(addr) # Open a source of data
while True:
    line = f.readline()   # Read one line from the address
    if len(line) == 0:    # Check if the line is empty
        break             # Exit from the while loop
    print line            # Print out the line
f.close()                 # When the loop finishes close the source
```

This imports the `urllib2` package, sets the variable addr to the address to open, then uses urllib2.urlopen to get the data. This returns a variable f which represents the data, and which can be used like a file. As in the last lab, we can read individual lines, and extract values from it using split() and float().

Note that the first three lines contain the column headers, so that the data starts on line 4. We therefore want to skip the first 3 lines, and the easiest way to do this is to read the line but not do anything with it.

```
line = f.readline() # Read one line
line = f.readline() # Read next line. Previous line discarded
```

Task 1 Find data for given date (20%)

The date in these files has the format DD_MM_YY. Write a program which finds the average temperature on a given day in November 2010. You can compare two strings (bits of text) using == to test if they're equal. Use the program below as a starting point:

```
date = "27_11_10"  # Give a date to find
import urllib2
addr="http://weather.elec.york.ac.uk/data/vaisala/archives/1110.txt"
f = urllib2.urlopen(addr)
while True:
    line = f.readline()
    if len(line) == 0:
        break
    # Split the line into columns
    # Check if this line has the correct date
    # If so, get the temperature and break out of loop
f.close()
print ###  # Print temperature
```

If we want to get data for other months, then we'll need to open a different address. For data going back to December 2009, the address looks like

"http://weather.elec.york.ac.uk/data/vaisala/archives/MMYY.txt"

where MM is the month, and YY is the year.

Task 2 More general daily temperature (30%)

a) Define a function getMMYY(date) to convert a date string "DD_MM_YY" into a string "MMYY". Hence getMMYY("27_11_10") should return the result "1110".

   Hint: split () can be given the delimiter between fields, so s. split ("_") splits a string s into pieces separated by "_". To join bits of text together use +.

b) Use this function to modify your program from the last task and change the name of the address opened. Your program should be able to cope with any date back to December 2009.

c) Add checks so that your code prints a useful error message if it can't find the data

3D visualisation

Visual Python (VPython) is a way to quickly make 3D visualisations of objects and data. Objects which can be created include: arrow, box, cone, curve, cylinder, ellipsoid , helix, points, pyramid, ring, and sphere. There is also frame for grouping objects together, and vector for keeping track of locations, doing cross products etc. To find out how these work, just run help() e.g. help(curve).

The following code creates a red sphere with centre at (-5,0,0) and a green wall at (6,0,0). The wall is 0.2 units thick in the x direction, and 4 units long in y and z.

```
from visual import *
ball = sphere(pos=(-5,0,0), radius=0.5, color=color.red)
wallR = box(pos=(6,0,0), size=(0.2,4,4), color=color.green)
```

We can give the ball a velocity in the x direction

```
ball.velocity = vector(2,0,0)
```

To actually make the ball move, we need to tell Python how to update the position based on the velocity. To do this, we can use a loop to repeatedly update the position in small time steps δt since the change in position $\delta x = v \times \delta t$

```
dt = 0.02       # Size of the time step
while True:     # Just means loop forever
    rate(50)    # Limits to 50 times per second
    ball.pos = ball.pos + ball.velocity * dt
```

Task 3 Bouncing (30%)

a) If you run the above program, you'll see that the ball just goes straight through the wall. Adapt this program so that when the x position of the ball (given by ball.x) exceeds the x position of the wall then the x component of the velocity (given by ball.velocity.x) is reversed. The ball should now hit the wall and bounce before going off in the other direction

b) Add another wall at (-6,0,0) and add another check so that when the ball hits this wall it also bounces. The ball should now keep bouncing back and forwards between the two walls

Satellite orbits

We can use VPython to demonstrate orbital motion of satellites according to Newton's law of gravitation. If the mass of the Earth is M_E, and the satellite m then the force on the satellite is

$$\underline{F} = -\frac{G M_E m \underline{r}}{r^3}$$

where $\underline{r}$ is the vector from the Earth to the satellite. The acceleration of the satellite is $\underline{a} = \underline{F}/m$ i.e.

$$\underline{a} = -GM_E\underline{r}/r^3$$

The values are $G = 6.67 \times 10^{-11}$ and $M_E = 5.974 \times 10^{24}$ in standard SI units, but to keep our numbers reasonable we're going to use units of length in thousand km, and time in minutes. Using these units, $GM_E = 1.43$ (thousand km)^3 / minute^2. For a circular orbit,

$$\frac{mv^2}{r} = \frac{GM_Em}{r^2} \qquad \Rightarrow v = \sqrt{GM_E/r}$$

Hence a satellite at $r = 10$ thousand km has a velocity of $v = \sqrt{1.43/10} = 0.378$ thousand km per minute.

To visualise this, we'll create a sphere for the Earth of radius 6.378 (thousand km), and another sphere for the satellite at a height of 10 thousand km.

```
from visual import *
earth = sphere(pos=(0,0,0), radius=6.378, color=color.blue)
sat = sphere(pos = (10,0,0), radius=0.5, color=color.white)
```

(the satellite is quite big!) The initial velocity of the satellite is 0.378

```
sat.velocity = vector(0,0.378,0)
```

Task 4 Satellite orbit visualisation (20%)

a) Make a working animation by completing the code below to update both the position and velocity. The function mag() is used to get the magnitude of a vector, and this code also includes a trail to plot the path of the satellite.

```
sat.trail = curve(color=color.yellow) # Add a trail
dt = 0.1
while True:
    rate(50)
    accel = -1.43 * sat.pos / mag(sat.pos)**3
    #### Update the position
    #### Update the velocity
    sat.trail.append(pos=sat.pos) # update the trail
```

b) Arrows can be created using g = arrow(pos=sat.pos, color=color.green) In the loop arrow positions can be changed by modifying g.pos = #### and direction by setting g.axis = ###. Add two arrows which follow the position of the satellite, showing the velocity and acceleration directions. Note: You'll need to multiply the velocity vector axis by about 5 and the acceleration axis by 200 to make them visible.

Extra tasks

As always by this point, that's the end of the assessed tasks. Unlike previous labs, this is the last one so these extra tasks are more open-ended than usual: to do many of these you'll need to search the Internet and Python help pages.

Task 5 Graphing weather data

In this task the aim is to read weather data into NumPy arrays, then plot this data over time. One way to do this[1] is to start with empty arrays and then add additional values onto the end using append().

```
from numpy import *
x = array([])
x = append(x, 10) # Puts 10 into x
x = append(x, 20) # Puts 20 at the end of x
print x
```

Start with the same program as task 1, and adapt it so that it stores the day of the month in x and the average temperature in y

```
from numpy import *
###
x = array([])
y = array([])
while True:
    line = f.readline()
    if len(line) == 0:
        break
    ####
    x = append(array, ###)
    y = append(array, ###)
###
```

Task 6 Reading older weather station data

In task 2, your code could cope with any date back to December 2009. Before this date, a different weather station was used and the data is stored in a slightly different place. From January 1999 to December 2008, the data is stored under “oldStation” rather than “vaisala”

”http://weather.elec.york.ac.uk/data/oldStation/archives/0811.txt”

Adapt your program from task 2 so it automatically chooses which address to open, and can cope with all available dates back to January 1999.

Task 7 Plotting longer-term temperatures

Improve your program from task 5 so that it gathers data over a range of months and plots it. To do this, you will need to use the datetime module.

Task 8 Orbital mechanics

The satellite visualisation in task 4 can be extended and used in many ways

a) Using this visualisation you could explore what happens when the horizontal and vertical velocity is modified

b) You could add the moon to the simulation, and include the force between the moon and the “satellite”. This could be used to visualise the orbit needed to get to the moon

c) You’ll notice that the satellite doesn’t quite stay in the correct orbit. This is because our method for updating the position and velocity is not very accurate. This will be covered in more detail next year, but you could look up SciPy and odeint.

There are many good Python examples and tutorials online, covering all aspects of programming. If you’re interested, there are some links here:

`http://www-users.york.ac.uk/~bd512/teaching.shtml`