Lesson 3: Modules, Arrays and Plotting

1 Modules

Consider again the motion of an object thrown vertically. This time we want to know the time t the object passes through a given vertical position y. The analytical solution is:

$$t = \frac{v_0 \pm \sqrt{v_0^2 - 2gy}}{g} \tag{1}$$

The minus sign applies when the object is moving upward; the plus sign applies when the object is moving downward.

In Exercise I you will be asked to write a routine to compute the time, given the initial velocity and vertical position. In doing so, you will need to use the `sqrt()` command to compute the square root function. However python interpreters do not automatically recognize every mathematical function that one can imagine. Instead, functions are grouped together into *modules* that can be imported into your routine, and thereby added to the python interpreter's vocabulary. The command `sqrt()`, along with trig functions, exponentials, logarithms, *etc*, is contained in the python `math` module. The `math` module can be imported by adding `import math` at the beginning of your routine. To compute the square root of 5.0 and place the result in the variable `result`, you can write

```
import math
result = math.sqrt(5.0)
print result
```

Note that the python interpreter needs to know that `sqrt()` comes from the `math` module, so you have to provide the full name `math.sqrt()`.

It can be tiresome to always type the prefix `math.` as part of the name for common mathematical functions. Instead we can write

```
from math import *
result = sqrt(5.0)
print result
```

This imports the entire math module with a lighter notation. However, we need to be careful when importing modules in this way. If we import more than one module, we run into the possibility of importing two different functions with the same name but different effects. A good solution is to import the module with a shorter name:

```
import math as m
result = m.sqrt(5.0)
print result
```

It is also possible to import just a single function, rather than an entire module. However, it is often more convenient to simply import the whole module.

> Exercise I: Write a routine that asks you for the initial velocity and vertical position and then computes the times according to the equation above. What happens if the vertical position is too large?

2 Arrays

Let's assume that we want to compute the position of the vertically thrown object at times $t = 1.0$, 1.5, 2.0, 2.5 and 3.0 using the formula

$$y = v_0 t - \frac{g}{2} t^2 \tag{2}$$

from the previous lesson. We can create a program to carry out this calculation for a given time t, then run the program for each desired time. Alternatively, we can instruct the program to compute the y values for all of the desired t values in a single step. To do this, we need to create a set of times. This can be done in several ways, but we need to be careful. For example, we can create a *list* of times like this:

```
t = [1.0,1.5,2.0,2.5,3.0]
```

However, lists behave differently than we would often like. For example, if we add two lists together we get

```
t + t = [1.0,1.5,2.0,2.5,3.0,1.0,1.5,2.0,2.5,3.0]
```

In most cases we would prefer to have the corresponding list elements added; for example, we want `t + t` to yield `[2.0,3.0,4.0,5.0,6.0]`.

To get the desired behavior, we need to define `t` as an *array* using the `numpy` module. There are several ways to do this. In this case, the best way is to use the `linspace` command:

```
import numpy as n
t1 = n.linspace(1.0,3.0,5)
```

The command `linspace(a,b,s)` creates an array whose first element is `a` and last element is `b`. The total number of elements (including `a` and `b`) is `s`, which must be an integer. The elements are evenly spaced. Note that `linspace` always creates an array of *real* numbers, even if `a` and `b` are given as integers.

Other numpy commands for creating arrays include

```
t2 = n.array([1.0,1.5,2.0,2.5,3.0])
t3 = n.arange(1,3.1,0.5)
```

The `array` command simply creates an array with the elements listed. If any one of the elements is real, then all of the elements are treated as real. The `arange(a,b,s)` command creates an array whose first elements is `a`, then increments this element by `s` up to the maximum value *less than* `b`.

As a general rule, you should use `linspace(a,b,s)` for creating arrays of real numbers. It can be tricky to use `arange(a,b,s)` if any of the values `a`, `b` and `s` are non–integers. Consider the following examples:

```
t4 = n.arange(2,7,1)
t5 = n.arange(15.1,18.1,0.3)
```

Can you predict what the array elements will be in each case?

> Exercise II: Write a routine that creates the time arrays `t1`, `t2` and `t3`. Verify that they are identical. What happens when you add arrays? What happens if you change the real numbers to integers? Create the arrays `t4` and `t5` and check to see if the element `b` is included.

The advantage of operating with arrays rather than lists of numbers is that Python can perform mathematical operations on arrays without cycling over the individual elements one by one. For example, to compute the position of our object at times $t = 1.0, 1.5, 2.0, 2.5, 3.0$, we simply write:

```
import numpy as n

# initial data and parameters
t = n.linspace(1.0,3.0,5)
v0 = raw_input("initial velocity: ")
v0 = float(v0)
g = 9.8

# compute the heights for the array of times t
y = v0*t - 0.5*g*t**2

# print the results
print "The heights at times t = {} are y = {}" .format(t,y)
```

Note that the line where we compute y is the same, whether `t` is an array or a single number.

3 Help

If you don't know exactly how to use a function inside a module, help is available. For example, type

```
help(n.arange)
```

to obtain help with the `n.arange()` function. Within Canopy, if you've imported the module either through the command-line or by running a file, you can display help by typing the name of the function and the opening parenthesis into the command-line. For example, type `n.arange(`.

4 Plotting graphs

As a final task, let's create a plot of the object's position as a function of time. To do that, we first need to compute the position in a fairly high resolution time array. Once we have the positions y for many times t, we use the `pylab` module to make the plot. Stripping the comments and embellishments, our routine will look like this:

```
import numpy as n
import pylab as p

t = n.linspace(0.0,2.0,100)
v0 = raw_input("initial velocity: ")
v0 = float(v0)
g = 9.8

y = v0*t - 0.5*g*t**2

p.plot(t,y,"r")
p.plot(t,y,"r.")
p.title("Height of object")
p.xlabel("Time (seconds)", fontsize=16)
p.ylabel("Height (meters)", fontsize=16)
p.show()
```

The command `show()` displays the figure as directed in the preceding commands. The command `plot(t,y,"r")` draws a smooth red curve through the data points. The command `plot(t,y,"r.")` displays each data point as a red dot.

To save a pdf copy of your figure, insert the `pylab` command `savefig("filename.pdf")` after the `show()` command. You can save the figure as other file types by using the appropriate filename extension. (For example, png rather than pdf.)

Homework: Using Newton's second law $\vec{F} = m\vec{a}$, find x and y as functions of t for a projectile shot with initial speed v_0 and initial angle θ above the horizontal. (The y axis is vertically up, the x axis is horizontal.) Write a python routine that computes the x and y position the projectile for an array of times t. Have v_0 and θ as line inputs and plot the resulting trajectory (y vs. x).

Your LaTeX document should show the results for x and y as functions of t, and include a graph of y vs. x for chosen values of v_0 and θ. Your document should also include a copy of your code using the "verbatim" environment.

5 numpy, math, pylab, and matplotlib

The numpy module contains its own set of mathematical commands like sqrt() and cos(). If you have imported numpy into your program, there is no need to import the math module as well.

The actual module that contains the plotting routines (like plot) is called matplotlib. When you import pylab, you are acually importing both matplotlib and numpy. (Actually, the plot command is in a submodule of matplotlib called matplotlib.pyplot.)

I often import both numpy (as n or np) and pylab (as p or py), even thought the numpy import is not strictly necessary. This seems more natural to me. It allows me to keep separate, in my mind, the math and array operations (which use numpy commands), from the graphing and plotting operations (which use pylab commands).