## Exercise 5: Euler’s method

You’ve probably noticed that you do a lot of “simplified” problems in your physics class. We assume air resistance is negligible in free-fall, for example, or pretend that there’s no friction in the pulley. The reason we do this is that otherwise the problem is too hard to solve analytically, and it’s worthwhile to find an approximate solution anyway. With a computer, though, we can often get numeric answers even if we can’t get analytic answers. This exercise will show you an easy computational method to find approximate solutions to Ordinary Differential Equations. We’ll start by finding a solution to a problem for which we already know the answer: free-fall with negligible air resistance.

But first, there’s another bit of Python to learn: How to do things more than once.

### FOR LOOPS

If you know how many times you want to do something in Python, the “for loop” is the best way to make Python do that something that many times. (If you aren’t sure how many times it’ll take, then the “while loop” is a better choice, but let’s just do the for loop for now.) Try this code in a new text file in Python:

for j in range(10):

print("j =", j, "j\*\*2 =", j\*\*2)

Here’s how that code works: It sets the value of j to be each item in the list “range(10)”, then does all the following indented lines once for each value of j. So “range(10)” produces a list that goes from 0 to 9 (ten elements total), and then goes through the indented print statements 10 times. The first time through, j=0. The second, j=1; and so on through j=9.

One important thing to note is that Python always starts counting with 0. Keep this in mind, because it causes trouble occasionally! For example, if you wanted j to range from 1 to 10 (instead of 0 to 9) you’d have to change the code a bit. This would work, and if you stare at it long enough it might make sense:

for j in range(1,11):

print("j =", j, "j\*\*2 =", j\*\*2)

Another important thing to note is that there’s nothing special about the name j. I could just as well call it Rumplestiltskin, it’d still work.

A third important thing to note is that the for loop doesn’t have to use range(): it can use any list of things.

listOthings=(1,2,3,42,17,"Hike!")

for QB in listOthings:

print(QB)

Of course, you can’t print the square of ‘Hike!’, but you get the idea. Ok, got for loops? Now let’s go back to doing physics.

### EULER’S METHOD

Let’s start with the differential equations for free-fall:

v = dx/dt a = dv/dt (3)

where a is a constant. We can rearrange these equations:

dx = v dt dv = a dt (4)

Remember what dx and dt and dv mean: these are the infinitesimal changes in position, time, and velocity. This form of the equations suggests a possible way of approximating the solution to the differential equations: take little time-steps dt, and calculate the new x and v for each time-step.

xnew = xold + v dt

vnew = vold + a dt (5)

This is called “Euler’s method”. We have to take really small steps to get a good solution, which means we have to take lots of steps to get anywhere. That’s what the computer is for: doing all those boring calculations! Here’s some example code. Don’t try putting this into a Python console one line at a time: put it in a new text file, save it, and run it as a program.

from pylab import \*

g = 9.8 # We are on Earth.

dt = 0.1 # 1/10 second time step

N = 100 # I'd like to see 100 points in the answer array

vi = 50.0 # initial velocity

xi = 25.0 # initial position

# first, set up variables and almost-empty arrays to hold our answers:

t = 0

x = xi

v = vi

time = array([0]) # initial value of time is zero!

height = array([xi]) # initial height is xi

velocity = array([vi]) # initial velocity is vi

# Note that 't', 'x' and 'v' are the current time, position and velocity, but

# 'time', 'height' and 'velocity' are arrays that will contain all the positions

# and velocities at all values of 'time'.

# Now let's use Euler's method to find the position and velocity.

for j in range(N):

# here are the calculations:

t = t + dt

x = x + v \* dt

v = v - g \* dt

# And here we put those calculations in the arrays to plot later:

time = append(time,t)

height = append(height,x)

velocity = append(velocity,v)

# Now that the calculations are done, plot the position:

plot(time, height, 'ro')

xlabel("Time (s)")

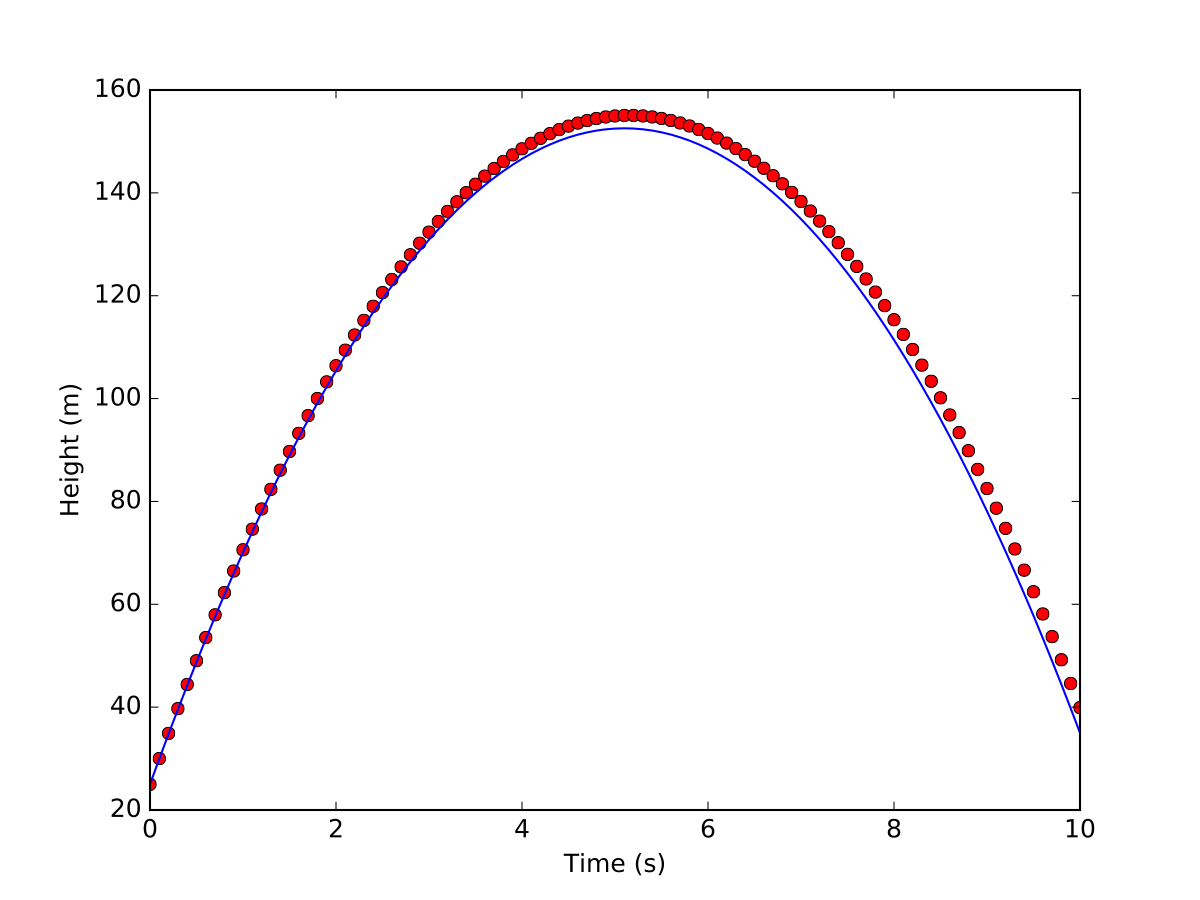
ylabel("Height (m)")

# just for comparison, I'll also plot the known solution!

plot(time, xi + vi\*time - 0.5\*g\*time\*\*2, 'b-')

show()

You can see on the graph below that this actually works pretty well. It’s a little bit off, but not bad; you can make it better by making dt smaller and N correspondingly larger.



### ASSIGNMENT

Change that sample code to ‘solve’ Simple Harmonic Motion instead of free-fall. The equations for SHM, in Euler format, are

dx = v dt

dv = a dt (6)

but instead of a=−g, use a=−ω2 x. Use xi=1, vi=0 , and ω=1 . Plot the position for 10 seconds. Make the code put your name in the title, of course, and ‘A’ students will plot the known solution also!