PHYS 391, Homework #1
Due Weds, 11-October (with Lab 1) by noon

1.1 Key Concepts involving scientific programming

- data initialization
- 'for' loops
- if statements
- plots
- functions vs. scripts

1.2 Physics Background

1.2.1 ball drop without drag

For this assignment, you will program a simulation of a ball drop that includes the effects of wind resistance on the ball. Later you will turn this code into a callable *function*. For extra credit you can modify your function to include both upwards and downwards motion of the ball (a vertical ball 'toss'). You are welcome to do your coding in Matlab (available on all computers in Rm. 17 Willamette) or python. See class notes about the differences between these two programming environments. This assignment assumes you have not had any experience programming computers beyond (perhaps) some experience with Excel.

The physics: Recall that, neglecting air resistance, a ball dropped with zero initial (vertical) velocity, v_0, will undergo a constant acceleration, a, under the influence of gravity near the surface of the Earth. Thus the ball will see its speed increase at a constant rate over time. If the upwards direction is set as positive, then the ball's velocity will be observed to become more negative over time, and the slope of an observed velocity vs. time graph will be -9.8 m/s/s (aka g) Further, given this constant acceleration and using the notion of advancing in small time steps, Δt, one can write:

$$v_{i+1} = v_i + a\Delta t$$

(with $a = g$ on Earth) The $i + 1^{th}$ position can then be determined by:

$$y_{i+1} = y_i + v_{i+1}\Delta t$$

1.2.2 adding drag

Drag force is observed to have an effect on a ball that scales as a drag coefficient, D, related to the balls shape and cross-sectional area, and scales also as the balls velocity squared (v^2). The drag force always acts opposite the direction of motion. Thus the drag ***force*** on the ball can be written as:

$$F_d = -Dv|v|$$

(think about how this works in terms of the sign of F_d.)

1.2.3 how Newton's Second Law matters

Note that the expression for the ball drop without drag addresses the acceleration of the ball without mentioning it's mass. Why is that, doesn't mass matter? Adding drag entails moving to a formulation based on force (F_d), thus combining drag with gravity entails formulating both in terms of force, and then dividing by mass to get the resultant acceleration. Think about how, by adding drag force, the mass of the ball does matter. To code your simulation, below, you should remember to specify a mass for the ball in order to get acceleration (combined from gravity and drag forces) and then simulate the ball's velocity and position over small steps in time.
Note, also, that ***the drag force itself depends on the instantaneous velocity!*** Thus simulating a ball drop with drag entails correcting the total force on the ball for each time step to account for changing

velocity. This is why it is advantageous to solve this problem using scientific programming, using a *for loop* (explained below) one can take very small steps in time, calculate the ball's velocity and position, then use the former to update the drag force (ergo the total force) on the ball. Don't forget, to do this simulation, you will need to estimate a value for D, the drag coefficient and set it for a particular calculation.

1.3 Homework Problems

Problem 1

For problem 1 you will set up a Matlab or python script (series of programming commands producing desired output) that gives the vertical position of a dropped ball over time after it is dropped from a height, h, versus time. For problem 1 you can neglect air resistance. Here's a typical sequence for programming, in Matlab (python will look fairly similar):

```
% (note a percent sign in Matlab starts a comment (non-interpreted line)
% you usually begin coding by initializing some variables, g for example
h = 10;
g = -9.8;
dt = 0.1;  % note that you can name variables as you wish, but they should mean something
y(1) = h;  %  start ball drop at y = 10 meters
%   note we will build an 'array' (or vector) of data  containing y
v(1) = 0;  % similar to above, we'll build an array of velocities
t(1) = 0;
% note above arrays would be indexed by zeros for python, eg., y(0) = 10, etc.

% the for loop below will execute (without fail) 1000 times.  The value i (an integer) will start
%    at 1 and increment with each iteration.  Thus, the first time through the for loop (between
%    the for statement and the end statement), t(2) will be set equal to t(1) + dt.
%    Similarly for v and y.
for i=1:1000
    t(i+1) = t(i) + dt;
    v(i+1) = v(i) + g * dt;
    y(i+1) = y(i) + v(i+1) * dt;
    end
%  As this will iteration 1000 times, you may find the ball hits the ground before the for
%     statement stops iterating.  You can bail out of the for loop with a logical if statement
%     to cause it to stop when, say, y(i) < 0 (hit ground).

%     to do so, you could insert the following statement in the for loop, just before the
%     end statement:
if (y(i)<=0), break, end

% note also, that python doesn't use 'end' statements.  Rather, the indentation of lines
%     marks the loop,  stop indenting when the loop is done
```

For problem 1, then, just code the above and confirm that it works. Send me (dlivelyb@uoregon.edu) your code. I should be able to run it in Matlab or python without making ANY modifications.

Problem 2

For problem 2, add in a drag force to your results from problem 1. Note all my comments, above, about formulating in terms of forces first, then working backwards to acceleration. Confirm that the ball takes longer to fall to the 'ground' (y=0) from 10 meters (it should, shouldn't it?). Make a plot showing the ball drop with and without drag force and discuss it briefly. Note that in Matlab the command for plotting is very simple:

```
plot(t,y)
% if you want to make a second plot in Matlab using the same axes, try this:
hold on
plot(t,ywD,'r+')
```

As part of your assignment, send me your problem 2 code. I should be able to run it without problems and produce plots showing your results. Although I don't need to see this, you could also try plotting v vs. t for both cases as well, not hard to do!

Problem 3

The last challenge here is to turn your *script* into a *function*. A function is a piece of code that can be re-used with different starting variables (for example, height (h), mass (m), drag coefficient (D), time step (dt), etc. Functions look different in Matlab than in python. In Matlab, the first line of code of a function would look like this:

```
function [t,y,v] = my-ball-drop(h0,m,D,dt)
%  this function is called 'my_ball_drop', and calculates the ball's position (y) vs. time (t)
%    also, its velocity (v) vs. time.  Inputs include the initial height (h0), the ball's mass (m)
%    the drag coefficient (D) and the time increment for tracking motion and constructing
%    time array (t).
%
%  the rest of the code would be similar to your script, except you can set:
     y(1) = h0;
%  and you no longer need initialize m, D and dt as in your script
```

If you successfully code your function (concluding with a *return* statement in Matlab), and save that to a file named *my-ball-drop.m* in a directory where Matlab can find it (see *pathtool*), then one can invoke the function from the command line (or from another, calling function or script) with:

```
[t,y,v] = my-ball-drop(h0,m,D,dt)
%  where you've already defined h0, m, D, and dt in the calling script or function (e.g.):

h0 = 10;
m = 1 % kg, heavy ball!  Use SI units everywhere!
D = 0.004;
dt = .01;
[t,y,v] = my-ball-drop(h0,m,D,dt)
```

So for problem 3, turn your script into a callable function, and send me both the function and the bit of code needed to call it.

Bonus Problem

For extra credit, modify your function so that it can simulate the complete motion of a ball thrown upwards (vertically) with an initial (positive) velocity v_0. Send that function and calling code to me.