USING EULER' S METHOD AND DISCRETE TECHNIQUES TO SOLVE TRAJECTORY PROBLEMS

Let' s start by considering the very simple scenario we used in class : an object is launched vertically with an initial speed of 20 m/s. In this first and simplest case, we will ignore air friction. We can apply Newton' s second law to this case and obtain :

$$\sum F = m \frac{d^2 x}{dt^2} = -m g$$

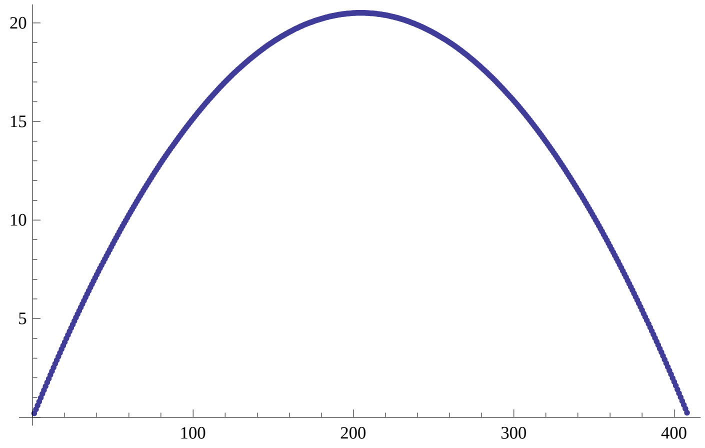
where we are defining the direction of the gravity vector as negative. We know from basic physics that we can trivially derive the time of flight and maximum height achieved by the object :

$$t_{\text{flight}} = \frac{2 v_0}{g} \qquad y_{\max} = \frac{v_o^2}{2 g}$$

If we wish to solve this problem using Euler' s method, we have to solve recursively for velocity and height. Consider the following program :

```
In[427]:= Clear[y, v, h, g]
          v[0] = 20; y[0] = 0; h = 0.01; g = 9.8;
          v[n_] := v[n] = v[n - 1] - h g
          y[n_] := y[n] = y[n - 1] + v[n - 1] h
          ListPlot[Table[y[n], {n, 408}]]
```

Out[431]=



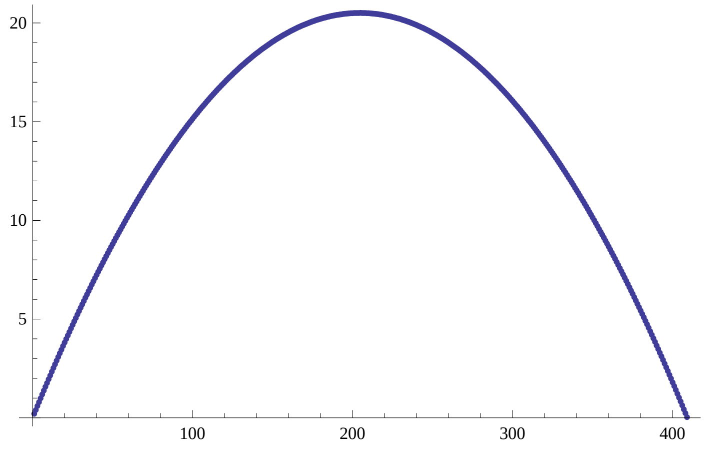
Notice that the equations for v[n] and y[n] are simply applications of Euler' s method. We compute the new values for v (and y) by adding a term equal to h*derivative to the old (most recent) value of v (or y). Note also the ListPlot statement specifies that we will plot the first 408 terms. I did not choose this number randomly; since I know the time of flight = 2 (20 m/s)/9.8 m/s/s = 4.0816 s.

Determining the number of points to plot within the program :

It would be valuable to avoid manually entering the number of points to plot each time we try a different simulation. There are a few ways we can have the program internally compute how many points to plot; the examples below show you how to do this. Using the same physical scenarion as above, consider the programs :

```
Clear[y, v, h, g, nterms]
v[0] = 20; y[0] = 0; h = 0.01; g = 9.8;
v[n_] := v[n] = v[n - 1] - h g
y[n_] := y[n] = y[n - 1] + v[n - 1] h
nterms = Catch[Do[If[y[n] < 0, Throw[n - 1]], {n, 1000}]];
ListPlot[Table[y[n], {n, nterms}]]
```

Out[448]=

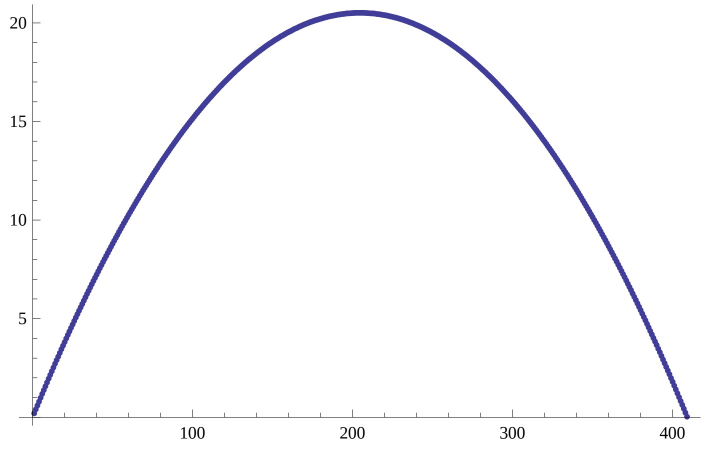


In this program, I define a variable ' nterms' to represent the number of points to plot. Since I only want to plot points where the projectile is above the horizon, I include an If condition that exits the program once y[n] < 0. Read up on Catch/Throw statements. In the example here, once y < 0, the Do loop is exited, and the current value of (n - 1) is ' thrown' out of the loop. Setting nterms equal to this statement allows me to determine the number of points to plot. I deliberately overestimate the upper limit of the do loop to ensure that I will achieve the condition y[n] < 0 before we reach n = 1000.

We could also compute nterms by using a While statement :

```
In[722]:= Clear[y, v, h, g, nterms, g1, g2]
v[0] = 20; y[0] = 0; h = 0.01; g = 9.8;
v[n_] := v[n] = v[n - 1] - h g
y[n_] := y[n] = y[n - 1] + v[n - 1] h
n = 1; While[y[n] > 0, n++; nterms = n - 1]
g1 = ListPlot[Table[y[n], {n, nterms}]]
```

Out[727]=



Motion with linear friction.

In class, we considered the case of air friction that is proportional to the first power of velocity. In this case, Newton' s second law becomes :

$$\sum F = m\frac{d^2 x}{dt^2} = -kv - mg$$

This allows us to recognize that the acceleration of the object is :

$$a = -(k/m)v - g$$

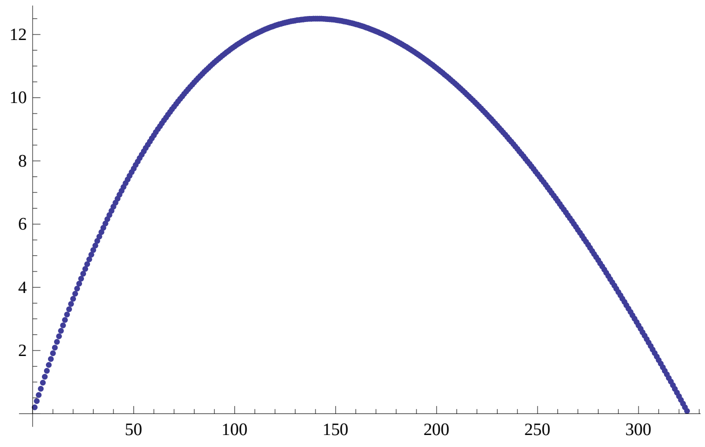
As we discussed in class, air friction always acts in the direction opposite motion; therefore, when the projectile is rising, the velocity vector is positive and the frictional term is - kv and points downward as it should. When the projectile is descending, its velocity is negative, so the frictional term is - k (-v) and points upward, as is the case with an object falling when there is friction. We can model the motion in this case using the following program (to distinguish this case from the no friction scenario, I call the variables ntermsf, vf and yf, and set k = 0.5) :

```
In[896]:= Clear[a, vf, yf, h, g, k, m]
vf[0] = 20; yf[0] = 0; h = 0.01; m = 1; k = 0.5; g = 9.8;
a[vf_] := a[vf] = (-k / m) vf - g
vf[n_] := vf[n] = vf[n - 1] + h a[vf[n - 1]]
yf[n_] := yf[n] = yf[n - 1] + vf[n - 1] h
ntermsf = Catch[Do[If[yf[n] < 0, Throw[n - 1]], {n, 1000}]];
maxht = Catch[Do[If[vf[n] < 0, Throw[yf[n - 1]]], {n, 1000}]];
Print["Time of flight = ", ntermsf h, " seconds"]
Print["Maximum height achieved = ", maxht, " meters"]
g2 = ListPlot[Table[yf[n], {n, ntermsf}], PlotRange → All]
```

```
Time of flight = 3.24 seconds

Maximum height achieved = 12.4994 meters
```

Out[905]=

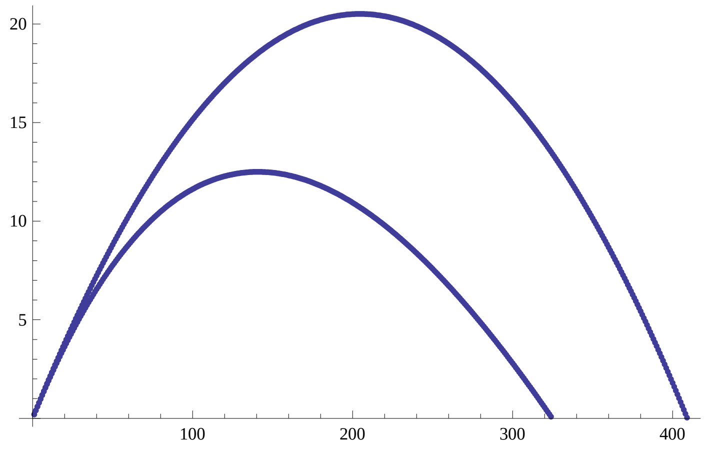


Look at the variable maxht and make sure you understand how it is computing maximum height. What would happen if you tried to find max height by using the fact that the vertical velocity is zero at maximum height?

Finally, I plot the graphs for y (t) for both curves below :

In[865]:=
```
Show[g1, g2]
```

Out[865]=


20
15
10
5
100
200
300
400