Loop Control : Using Catch and Throw

We have seen how using discretization techniques and recursion formulae allow us to take equations and solve them numerically. We have learned how to compute multiple iterations using Do, For and While Loops. In this classnote, we learn how to interrupt those loops.

Let' s use a very simple physical system; consider an object launched vertically from the surface of the Earth with a velocity v0. Neglecting friction, we know the equations governing this motion are :

$$y(t) = v_0 t - \frac{1}{2} g t^2$$

$$v(t) = v_0 - g t$$

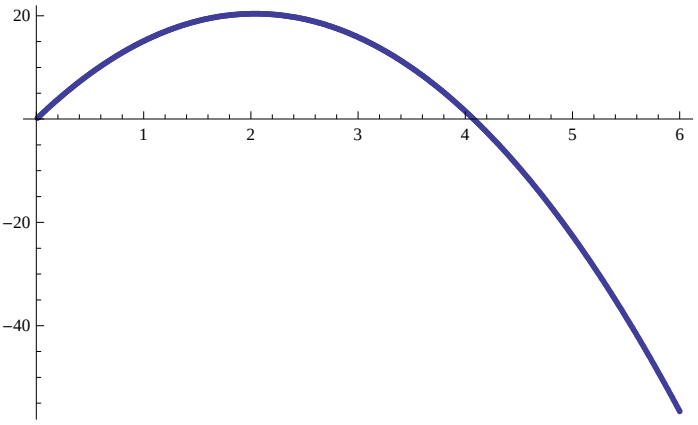
It is an easy matter to solve these equations numerically. Setting our initial velocity as 20 m/s, we write :

In[736]:=
```
Clear[v, y]
v[0] = 20; y[0] = 0; h = 0.01; g = 9.81;
v[n_] := v[n] = v[n - 1] - g h
y[n_] := y[n] = y[n - 1] + h (v[n] + v[n - 1]) / 2
```

We are by now very familiar with using the ListPlot[Table ... command to plot out various data, but suppose we want only to graph data while the object is in the air. How can we determine, within the structure of the program, how many data points we should plot. For instance, if we plot the first 600 points, we get :

In[741]:=
```
ListPlot[Table[{n h, y[n]}, {n, 600}]]
```

Out[741]=

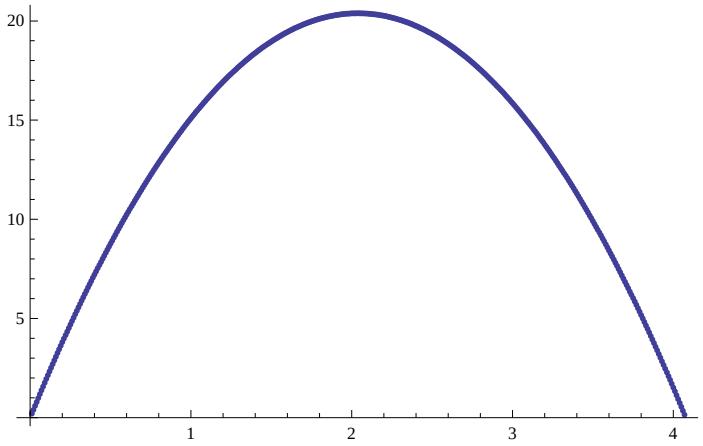


This looks like the proper graph, but we realize that we have tried to plot points for times after the object has returned to the ground. We could estimate from the graph above that the object will hit the ground in approximately 4.1 s, or we could use the equations of motion to recall that the object will return to earth in a time of 2 v0/g = 4.08 s. However, in more complicated physical situations, it would be useful to be able to compute this during the execution of the program. We can do this by utilizing the Mathematica functions Catch and Throw. Let' s rewrite this program making use of

these functions :

In[754]:=
```
Clear[v, y]
v[0] = 20; y[0] = 0; h = 0.01; g = 9.81;
v[n_] := v[n] = v[n - 1] - g h
y[n_] := y[n] = y[n - 1] + h (v[n] + v[n - 1]) / 2
nterms = Catch[Do[If[y[n] < 0, Throw[n - 1]], {n, 1, 1 000 000}]];
ListPlot[Table[{n h, y[n]}, {n, nterms}]]
```

Out[759]=



The first four lines of the program are typical of programs you have written before. The fifth line defines a new variable, nterms, which will represent the number of iterations we calculate until the object hits the ground. You can see in the last line that the limits of our plot are from n = 1 to this value of n = nterms, and that we obtain our desired result, namely, a graph that truncates when the object hits the ground.

Our question now is how does the code in line 4 accomplish this?

Start with the If statement; our test condition is y[n] < 0, which means we are testing whether the object is still in the air (y > 0). If the calculated position of y[n] < 0, we know the object has hit the ground, therefore we would like to exit the Do loop. That is what the Throw statement does; as long as y ≥0, the program will iterate through the Do loop (up to a million iterations if necessary). However, once the value of y < 0, the program exits the Do loop and "throws" the current value of the argument (or arguments) inside Throw.

In this case, once y < 0 we exit the loop by throwing the current value of (n - 1). (Think about why we throw the value of n - 1 instead of the value of n.) We "Catch" this value of n and define it as nterms, such that we can use this value of nterms in later lines of the code. You must use Catch whenever you Throw values; this may seem superfluous but it is required by the structure of *Mathematica*.

Let’s see how we can use Catch and Throw to determine some other values. Suppose we want to find the maximum height of projectile. We know that the maximum height occurs at the apex of the trajectory, so we have a few ways we could find this value:

In[809]:=
```
Clear[v, y]
v[0] = 20; y[0] = 0; h = 0.01; g = 9.81;
v[n_] := v[n] = v[n - 1] - g h
y[n_] := y[n] = y[n - 1] + h (v[n] + v[n - 1]) / 2
Catch[
 Do[If[v[n] < 0, Throw[Print["The maximum altitude of the projectile =  ", y[n - 1]]]],
  {n, 1, 1 000 000}]]
Catch[Do[If[Abs[y[n] - y[n - 1]] < 0.001,
   Throw[Print["The maximum altitude of the projectile =  ", y[n - 1]]]], {n, 1, 1 000 000}]]
```
```
The maximum altitude of the projectile =  20.387

The maximum altitude of the projectile =  20.387
```

The first Catch/Throw statement determines when the velocity of the object becomes negative, since we know that the velocity of the object is positive in the ascent phase and zero at max altitude. The second Catch/Throw statement yields the same result, but checks for maximum altitude by finding when the height between successive values of y is a minimum. Notice what happens if you use Abs[y[n] - y[n - 1] < 0.0001. Why do you think you get this result?

We can throw the values of more than one variable, as in :

In[2106]:=
```
Clear[v, y]
v[0] = 20; y[0] = 0; h = 0.01; g = 9.81;
v[n_] := v[n] = v[n - 1] - g h
y[n_] := y[n] = y[n - 1] + h (v[n] + v[n - 1]) / 2
nterms = Catch[Do[If[y[n] < 0, Throw[n - 1]], {n, 1, 1 000 000}]];
solution = Catch[Do[If[y[n] < 0, Throw[{(n - 1) h, y[n - 1], v[n - 1]}]], {n, 1, 1 000 000}]]
Print["Time of flight = ", solution[[1]]]
Print["Height of final evaluation = ", solution[[2]]]
Print["Final evaluated velocity = ", solution[[3]]]

ListPlot[Table[{n h, y[n]}, {n, nterms}]]
```

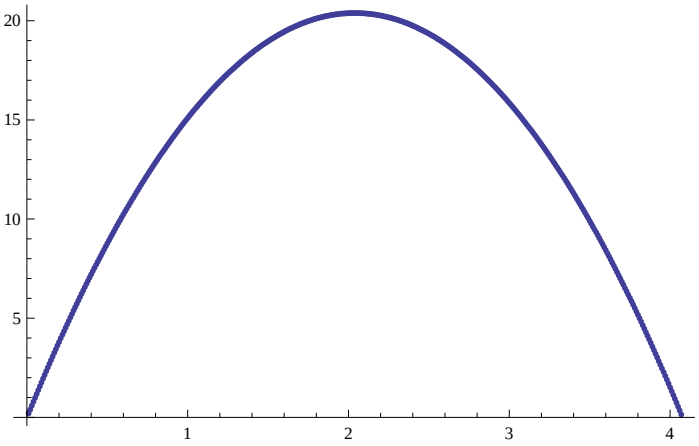
Out[2111]= {4.07, 0.149166, -19.9267}

```
Time of flight = 4.07

Height of final evaluation = 0.149166

Final evaluated velocity = -19.9267
```

Out[2115]=



Where the output statements print the time of trajectory for the object, the last height c alculated before impact, and the velocity at that height.