Undergraduate Classical Mechanics PHGN350
Solutions to Numerical Homework I
August 2011

- **In good programming practice, you must define all your variables in comments. This will be required in future notebooks.**

t = time
y = distance between center of mass of probe and surface
v = velocity
a = acceleration
r = distance between center of mass of probe and center of mass of Mars
rMars = radius of Mars
rProbe = radius of probe
mMars = mass of Mars
mProbe = mass of probe
ρ = density of Martian atmosphere
ρ0 = surface density of Martian atmosphere
y0 = scale height of density of Martian atmosphere
h = initial height of probe
fAtmo = atmospheric force (scaled to probe mass in equations below)
fGrav = gravitational force
cd = dimensionless drag coefficient
area = cross-sectional area of probe
gravConst = graviational constant
t = time
yData = {t,distance dropped}
vData = {t, velocity}
aData = {t, acceleration}
j = integer iterator for loop
stepSize = time step size
maxSteps = maximum number of steps allowed in loop
thisAcceleration = storage variable in loop

Let's set up the equations first.

- **Make some overall definitions, and set our parameter list. These should be things that WILL NOT change from study to study. Stuff that will change from study to study is better included in an input list for your modules.**

In[37]:=
```
fAtmo = 1/2 cd ρ area v^2;
ρ = ρ0 Exp[-y / y0];
fGrav = -(gravConst mMars mProbe)/r^2;
area = π rProbe^2;
r = rMars + y;
```

Using Newton's second law

In[42]:=
```
a1 = 1/mProbe (fAtmo + fGrav)
```

Out[42]= $$\frac{-\frac{\text{gravConst mMars mProbe}}{(\text{rMars}+\text{y})^2}+\frac{1}{2}\,\text{cd}\,e^{-\frac{\text{y}}{\text{y0}}}\,\pi\,\text{rProbe}^2\,\text{v}^2\,\rho\text{0}}{\text{mProbe}}$$

In[43]:=
```
a2 = Simplify[a1]
```

Out[43]= $$-\frac{\text{gravConst mMars}}{(\text{rMars}+\text{y})^2}+\frac{\text{cd}\,e^{-\frac{\text{y}}{\text{y0}}}\,\pi\,\text{rProbe}^2\,\text{v}^2\,\rho\text{0}}{2\,\text{mProbe}}$$

Although I have defined symbols with equal signs, they are only defined in terms of other symbols. I use a parameter list to define symbols as numbers.

In[44]:=
```
params = {cd → 0.2, gravConst → 6.673 * 10^-11, mMars → 6.419 * 10^23,
    mProbe → 10, y0 → 11.1 * 10^3, ρ0 → 0.02, rProbe → 5, rMars → 3390 * 10^3};
```

Martian constants came from the NASA website, using the Mars Fact Sheet: http://nssdc.gsfc.nasa.gov/planetary/factsheet/mars-fact.html

As a test, if I substitute these parameters into the acceleration, I get a function only of y and v. If anything else is left, it will not give me numbers when I plug it into my loop.

In[45]:=
```
a2 /. params
```

Out[45]= $$0.015708\,e^{-0.0000900901\,\text{y}}\,\text{v}^2-\frac{4.2834\times 10^{13}}{(3\,390\,000+\text{y})^2}$$

As I mentioned in lecture several time, in numerical methods one discretizes the governing equations. In the following solution, I discretize the equations in time: $v_{n+1} = v_n + a_n$ dt, $y_{n+1} = y_n + v_n$ dt+ a_n dt^2,, and $t_{n+1} = t_n$ +dt. The grid over t runs from n=1 to some large N which will correspond to when the probe hits the planet. The stepsize is the time resolution, dt. An initial condition is required for the following method, called a *shooting method*. The shooting method is good for ordinary differential equations in one dimension.

- **1) Make the function that will calculate**

```
In[46]:= loadData[tMax_, dt_, h_, print_] := Module[{i, iMax, lastI},
   (*Get the maximum number of iterations possible*)
   iMax = Floor[tMax / dt] + 1;
   aFunc[y_, v_] = a2 /. params;
   (*Initialize the tables that will be solved for,
   and fill them with the associated times*)
   yData = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
   vData = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
   aData = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
   (*Set the initial values*)
   yData[[1, 2]] = h;
   (*Run the calculation in a loop. I prefer Do loops in mathematica,
   but While, and For are just as good. While is honestly the best in this case,
   because it allows you to break out of
    the loop without using the Break command.*)
   lastI = 1;
   Do[{
     lastI = i;
     (*Get the acceleration at the current time*)
     aData[[i, 2]] = aFunc[yData[[i, 2]], vData[[i, 2]]];
     (*Get the velocity and position at the next time*)
     vData[[i + 1, 2]] = vData[[i, 2]] + aData[[i, 2]] * dt;
     yData[[i + 1, 2]] = yData[[i, 2]] + vData[[i, 2]] * dt + 0.5 aData[[i, 2]] * dt^2;
     (*Check to see if you're done*)
     If[yData[[i + 1, 2]] < 5,
      If[print, Print["Final i: ", i];
       Print["Final time: ", dt * i];
       Print["Final y: ", yData[[i, 2]]];
       Print["Final v: ", vData[[i, 2]]];
       Print["Final a: ", aData[[i, 2]]];];
      (*At this point I strip off the extra elements of the table. You don't
       have to do this, but it will make the plots look a little nicer. You
       won't have to deal with all the zeros that were left over.*)
      yData = Drop[yData, -(iMax - i - 1)];
      vData = Drop[vData, -(iMax - i - 1)];
      aData = Drop[aData, -(iMax - i - 1)];
      Break[]];
    }, {i, 1, iMax - 1}];
   (lastI - 1)
  ];
```

Now I define some plots that I am going to use later. I do it with the := so that it doesn’t evaluate them right now, so I can just use their variable to evaluate whatever yData or vData happend to be when I evaluate them. Be sure to include labels as well as units in the labels!

```
In[47]:= py := ListPlot[yData, AxesLabel → {"time (s)", "height above surface (m)"}];
         pv := ListPlot[vData, AxesLabel → {"time (s)", "velocity (m/s)"}, PlotRange → All];
         pa := ListPlot[aData,
            AxesLabel → {"time (s)", "acceleration (m/s^2)"}, PlotRange → All];
```

- **Done Programming, now answer questions!**

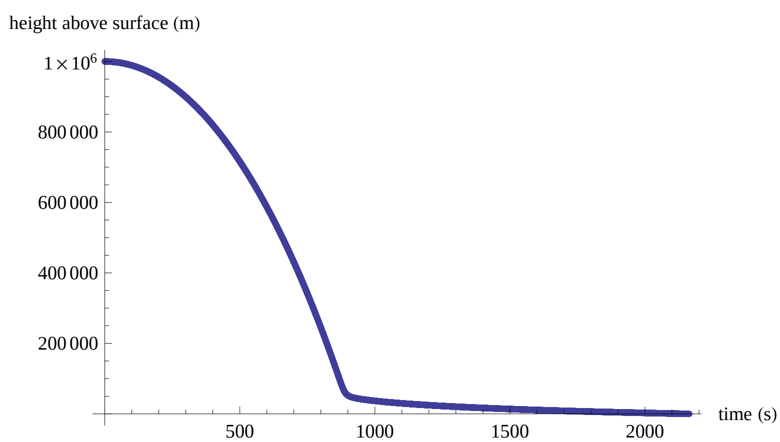
Now I can just solve the problem by running my module. If you want the module to return something in particular, such as the time, you just need to put it in at the end of the Module without a semicolon. I did this with the final i, so I can use that to get different final variables out (see the limiting case below). I already printed out the answers to what the final position, time, velocity and acceleration are.

```
In[50]:= loadData[3000, 0.5, 1*^6, True]

Final i: 4327

Final time: 2163.5

Final y: 10.6284

Final v: -15.4334

Final a: 0.0106862

Out[50]= 4326
```

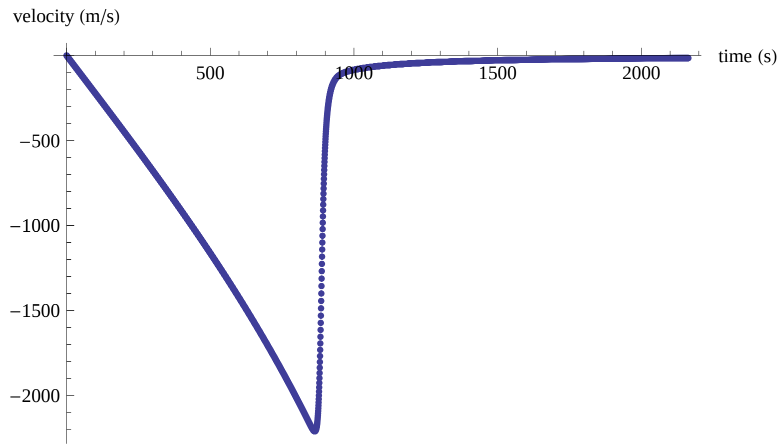
- **2) Plot the acceleration, velocity and position as functions of time**

```
In[51]:= py
```

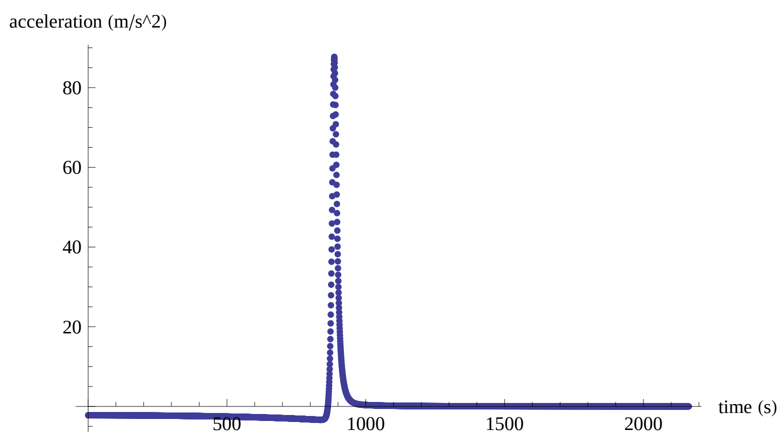
Out[51]=



In[52]:= **pv**

Out[52]=


In[53]:= **pa**

Out[53]=


There is a great deal of physics to see here. One can see the "bounce" on the atmosphere.

Note that since I didn’t make it to my max time, my problem really ends at

- **3) The answers are actually printed out where I solved it above. I resolve it here to show it again.**

In[54]:=
```
loadData[3000, 0.1, 1*^6, True];
```

```
Final i: 21628
Final time: 2162.8
Final y: 6.51592
Final v: -15.4306
Final a: 0.0106852
```

- **4) What are the maximum values of a and v as it falls?**

In[55]:=
```
maxV = Max[Table[-vData[[i, 2]], {i, 1, Length[vData]}]]
```

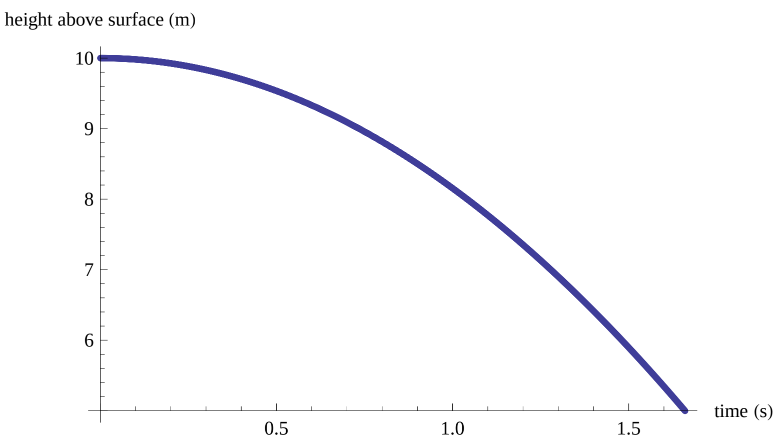
Out[55]= 2208.89

In[56]:=
```
maxA = Max[Table[aData[[i, 2]], {i, 1, Length[aData]}]]
```
Out[56]= 86.1212

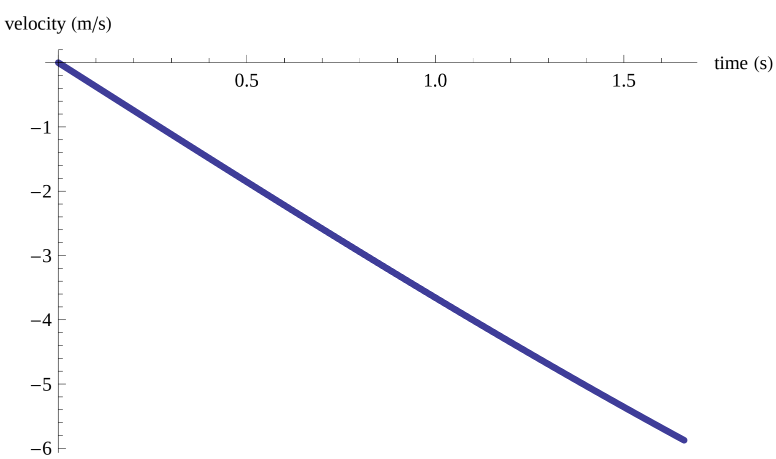
- **5) Now do it where h=10m. Notice I decreased my tMax and my dt.**

In[57]:=
```
iFinal = loadData[30, 0.001, 10, True]
```
```
Final i: 1660
Final time: 1.66
Final y: 5.00267
Final v: -5.87188
Final a: -3.18589
```
Out[57]= 1659

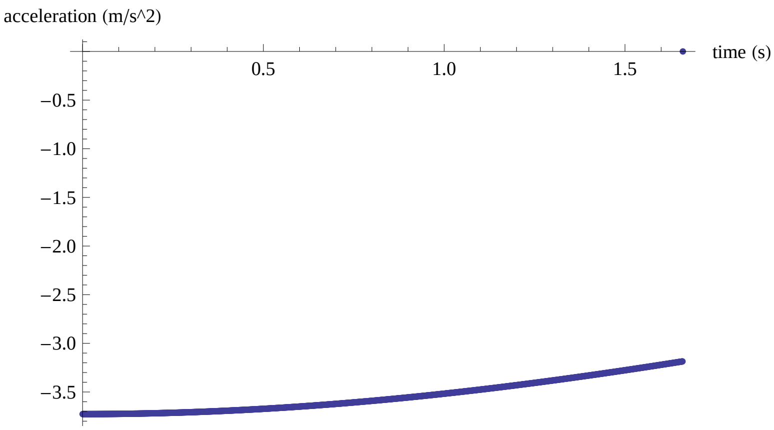
In[58]:=
```
py
```
Out[58]=



In[59]:=
```
pv
```
Out[59]=



In[60]:= **pa**

Out[60]=


We are still hitting drag, since the acceleration is decreasing. However, the height is nearly quadratic and the velocity is nearly linear, so it should be close to the no-drag calculation.

Now compare this to the constant gravitational field, no drag prediction. For that, we need g for Mars at the surface. Let's calculate it using our constants.

In[61]:= **gMars = gravConst * mMars / rMars^2 /. params**

Out[61]= 3.72725

In[62]:= $\mathbf{s1 = Solve\left[5 == 10 - \frac{1}{2}\, gMars\, time^2 \;/.\; params,\ time\right]}$

Out[62]= {{time → −1.63797}, {time → 1.63797}}

In[63]:= **approxTime = time /. s1[[2]]**

Out[63]= 1.63797

Calculate percent error

In[64]:= $\mathbf{percentError[x1\_,\ x2\_] = 100 * Abs\left[\frac{x1 - x2}{\frac{1}{2}\,(x1 + x2)}\right];}$

Percent error in time is

In[65]:= **myT = yData[[iFinal, 1]];**
percentError[myT, approxTime]

Out[66]= 1.21554

The error is about 1.2%. We can also check velocity and acceleration.

In[67]:= **approxV = −gMars approxTime /. params**

Out[67]= −6.10512

In[68]:= **myV = vData[[iFinal, 2]];**
percentError[myV, approxV]

Out[69]= 3.94907

In[70]:=
```
approxV = time /. s1[[2]]
```
Out[70]= 1.63797

In[71]:=
```
myA = aData[[iFinal, 2]];
percentError[myA, -gMars /. params]
```
Out[72]= 15.6435

The velocity and acceleration yield 3.95% and 15.6% errors, respectively.

Remark:
This implementation of the shooting method is only good for very small time steps, as the error is proportional to dt. Higher order methods will have error proportional to dt^2 or even dt^5.