Undergraduate Classical Mechanics PHGN350
Solutions to Numerical Homework II
P. David Flammer
September 2011

Physical variables:

t = time
θ = latitude ("polar" angle defined as zero at equator)
r = radial coordinate
θd=time derivative of θ
rd=time derivative of r
θdd=time derivative of θd
rdd=time derivative of rd
vr = radial velocity
vθ = angular velocity
v = Sqrt$\left[\text{vr}^2 + \text{v}\theta^2\right]$
ar = radial acceleration
aθ = angular acceleration
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
r0 = scale height of density of Martian atmosphere
h = initial height of probe above surface
v0 = initial velocity of probe (in e_θ direction)
fGrav = gravitational force
fθGrav = angular component of gravitational force
frAtmo = radial component of atmospheric force
fθAtmo = angular component of atmospheric force
cd = dimensionless drag coefficient
area = cross-sectional area of probe
gravConst = graviational constant
t = time

Variables for loop:

i = integer iterator for loop
dt = time step size
iMax = maximum number of steps allowed in loop

Make some definitions that will help us in our calculation

```
v = Sqrt[rd^2 + r^2 θd^2] ;
ρ = ρ0 Exp[-(r - rMars) / r0];
fAtmo = -1/2 cd ρ area v^2;
fGrav = -(gravConst mMars mProbe)/r^2; (*has units of force*)
frAtmo = -1/2 cd ρ area v rd; (* This has units of force *)
fθAtmo = -1/2 cd ρ area v r θd; (*This has units of force *)
area = π rProbe^2;
```

Using Newton's second law taking into account that the

```
rdd = 1/mProbe (frAtmo + fGrav) + r θd^2
θdd = 1/mProbe fθAtmo / r - 2 * rd * θd / r
```

$$r\,\theta d^2 + \frac{-\frac{\text{gravConst}\,\text{mMars}\,\text{mProbe}}{r^2} - \frac{1}{2}\,\text{cd}\,e^{\frac{-r+\text{rMars}}{r0}}\,\pi\,\text{rd}\,\text{rProbe}^2\sqrt{\text{rd}^2 + r^2\,\theta d^2}\;\rho 0}{\text{mProbe}}$$

$$-\frac{2\,\text{rd}\,\theta d}{r} - \frac{\text{cd}\,e^{\frac{-r+\text{rMars}}{r0}}\,\pi\,\text{rProbe}^2\,\theta d\sqrt{\text{rd}^2 + r^2\,\theta d^2}\;\rho 0}{2\,\text{mProbe}}$$

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.

```
params = {cd → 0.2, gravConst → 6.673 * 10^-11, mMars → 6.419 * 10^23,
    mProbe → 10, r0 → 11.1 * 10^3, ρ0 → 0.02, rProbe → 5, rMars → 3.39 * 10^6};
```

I obtain Martian planetary constants and descriptions from http://nssdc.gsfc.nasa.gov/planetary/factsheet/marsfact.html.

So that we don't recalculate these constants every time through the loop, go ahead and redefine them here. Make sure that whatever you have in the params at this point won't vary as part of your study when you do this.

```
rdd2[r_, θ_, rd_, θd_] = rdd /. params
θdd2[r_, θ_, rd_, θd_] = θdd /. params
```

$$r\,\theta d^2 + \frac{1}{10}\left(-\frac{4.2834\times 10^{14}}{r^2} - 0.15708\,e^{0.0000900901\,(3.39\times 10^6 - r)}\,\text{rd}\sqrt{\text{rd}^2 + r^2\,\theta d^2}\right)$$

$$-\frac{2\,\text{rd}\,\theta d}{r} - 0.015708\,e^{0.0000900901\,(3.39\times 10^6 - r)}\,\theta d\sqrt{\text{rd}^2 + r^2\,\theta d^2}$$

I again use the shooting method, as described in the solutions to Numerical HW I. I make a module that has the following steps outlined in teh comments

```
FillData[tMax_, dt_, h_, vFromSat_, print_] := Module[{i, v0, lastI, iMax},
    (*Set how many steps this will use*)
```

```
iMax = Floor[tMax / dt + 1];
vSat = Sqrt[gravConst * mMars / (rMars + h)] /. params;
v0 = vSat + vFromSat;
(*Initialize the tables, since I didn't include them in the local variables,
they will be accessible globally*)
rTab = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
θTab = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
rdTab = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
θdTab = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
vθTab = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
vTab = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
xyTab = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
rddTab = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
θddTab = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
arTab = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
aθTab = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
aTab = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
centATab = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
angExtraATab = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
(*Get initial values*)
rMars2 = rMars /. params; (*The only reason I do this here is I want to
 limit the operations that I perform in the loop to make it run faster*)
rTab[[1, 2]] = rMars + h /. params;
θdTab[[1, 2]] = v0 / (rMars + h) /. params;
vθTab[[1, 2]] = v0 /. params;
(*Go through and do the calculation*)
lastI = 1;
Do[{
  lastI = i;
  (*Get the x/y position at the current time*)
  xyTab[[i]] =
   {rTab[[i, 2]] * Cos[θTab[[i, 2]]], rTab[[i, 2]] * Sin[θTab[[i, 2]]]};
     vθTab[[i, 2]] = rTab[[i, 2]] * θdTab[[i, 2]];
  vTab[[i, 2]] = √(rdTab[[i, 2]]^2 + vθTab[[i, 2]]^2 );
  centATab[[i, 2]] = rTab[[i, 2]] * θdTab[[i, 2]]^2;
  angExtraATab[[i, 2]] = -rdTab[[i, 2]] * θdTab[[i, 2]] / rTab[[i, 2]];
  (*Get the accelerations at the current time*)
  rddTab[[i, 2]] =
   rdd2[rTab[[i, 2]], θTab[[i, 2]], rdTab[[i, 2]], θdTab[[i, 2]]];
  θddTab[[i, 2]] = θdd2[rTab[[i, 2]], θTab[[i, 2]],
    rdTab[[i, 2]], θdTab[[i, 2]]];
  arTab[[i, 2]] = rddTab[[i, 2]] - rTab[[i, 2]] * θdTab[[i, 2]]^2;
  aθTab[[i, 2]] = rTab[[i, 2]] * θddTab[[i, 2]] + rdTab[[i, 2]] * θdTab[[i, 2]];
  aTab[[i, 2]] = Sqrt[arTab[[i, 2]]^2 + aθTab[[i, 2]]^2];
  (*Get the velocities at the next time*)
  rdTab[[i + 1, 2]] = rdTab[[i, 2]] + rddTab[[i, 2]] * dt;
```

```
    θdTab[[i + 1, 2]] = θdTab[[i, 2]] + θddTab[[i, 2]] * dt;
    (*Get the positions at the next time*)
    rTab[[i + 1, 2]] = rTab[[i, 2]] + rdTab[[i, 2]] * dt + 0.5 * rddTab[[i, 2]] * dt^2;
    θTab[[i + 1, 2]] = θTab[[i, 2]] + θdTab[[i, 2]] * dt + 0.5 * θddTab[[i, 2]] * dt^2;
    If[rTab[[i + 1, 2]] < rMars2,
     If[print, Print["Final i: ", i];
      Print["Final time: ", dt * i];
      Print["Final θ/π: ", θTab[[i, 2]] / Pi];
      Print["Final r: ", rTab[[i, 2]]];
      Print["Final v: ", vTab[[i, 2]]];
      Print["Final a: ", aTab[[i, 2]]];];
     Break[]];
   }, {i, 1, iMax - 1}];
  θTab[[lastI, 2]]
];
```

This is if you just let it go, to test to see if it gives us similar results to last time. They of course do. You didn't need to do this for points.

```
FillData[16 000, 1, 1*^6, -Sqrt[gravConst * mMars / (rMars + 1*^6)] /. params, True]
```

```
Final i: 2165

Final time: 2165

Final θ/π: 0.

Final r: 3.39001 × 10^6

Final v: 15.4304

Final a: 0.0106782
```

```
0.
```

```
maxA = Max[Table[aTab[[i, 2]], {i, 1, Length[aTab]}]]
```

```
89.6838
```

```
maxV = Max[Table[vTab[[i, 2]], {i, 1, Length[aTab]}]]
```

```
2210.08
```

These create some plots that we can use to examine stuff as we calculate it

```
pr := ListPlot[rTab, AxesLabel → {"time", "radius (m)"}, PlotRange → All]
pθ := ListPlot[θTab, AxesLabel → {"time", "angle (rad)"}, PlotRange → All]
pvr :=
 ListPlot[rdTab, AxesLabel → {"time", "radial velocity (m/s)"}, PlotRange → All]
pθd := ListPlot[θdTab, AxesLabel → {"time", "Angular velocity (rad/s)"},
  PlotRange → All]
pvθ := ListPlot[vθTab, AxesLabel → {"time", "velocity in angular direction (m/s)"},
  PlotRange → All]
par := ListPlot[rddTab, AxesLabel → {"time", "radial acceleration (m/s^2)"},
  PlotRange → All]
paθ := ListPlot[θddTab, AxesLabel → {"time", "angular acceleration (rad/s^2)"},
  PlotRange → All]
pCentAθ := ListPlot[centATab, AxesLabel →
   {"time", "centripetal acceleration (m/s^2)"}, PlotRange → All]
myCirclePlot := ParametricPlot[{rMars Sin[a], rMars Cos[a]} /. params,
   {a, 0, 2 π}, PlotStyle → {Thick, Red}];
porbit := Show[ListPlot[xyTab, AxesLabel → {"x-position (m)", "y-position (m)"},
   PlotRange → All], myCirclePlot, AspectRatio → Automatic]
```

I didn't give the probe enough initial velocity to get it to orbit the planet even once. We can see this by combining the orbit plot with a circle representing the (red!) planet:

Graded Material (on top of code that you wrote)

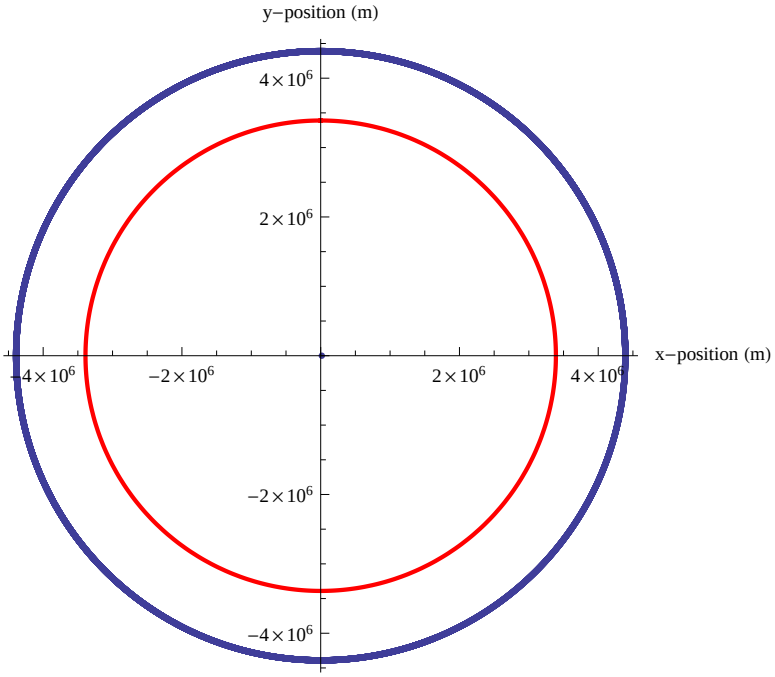
- **(a) This is the case where you just let it go. You should get a circular orbit**

```
FillData[16 000, 1, 1*^6, 0, True]
```

```
11.3839
```

The 11.3839 is the angle in radians it went around. It never hit, so it never stopped. Let's look at the orbit.

```
porbit
```

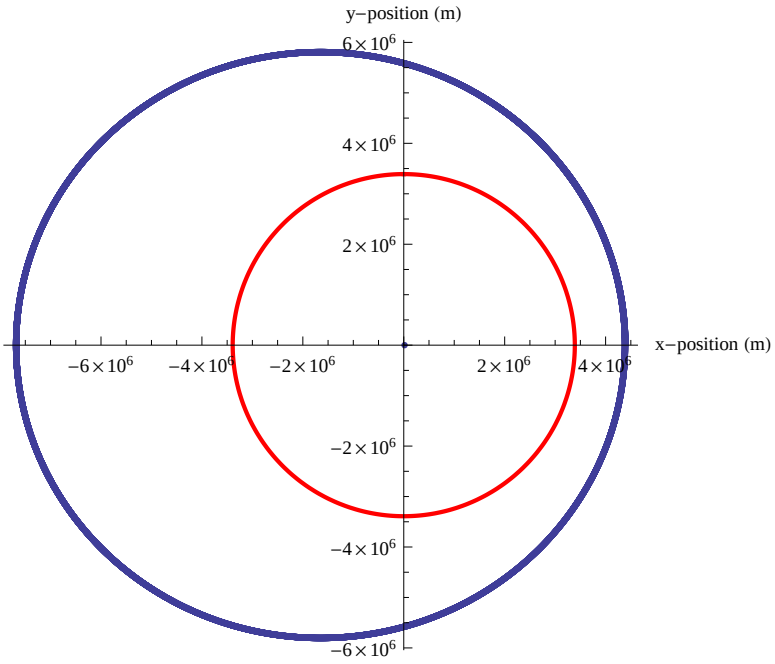


- **(b) This is the case where you launch it in the wrong direction at 400m/s**

```
FillData[16000, 1, 1*^6, 400, True]
```

```
7.54866
```

```
porbit
```



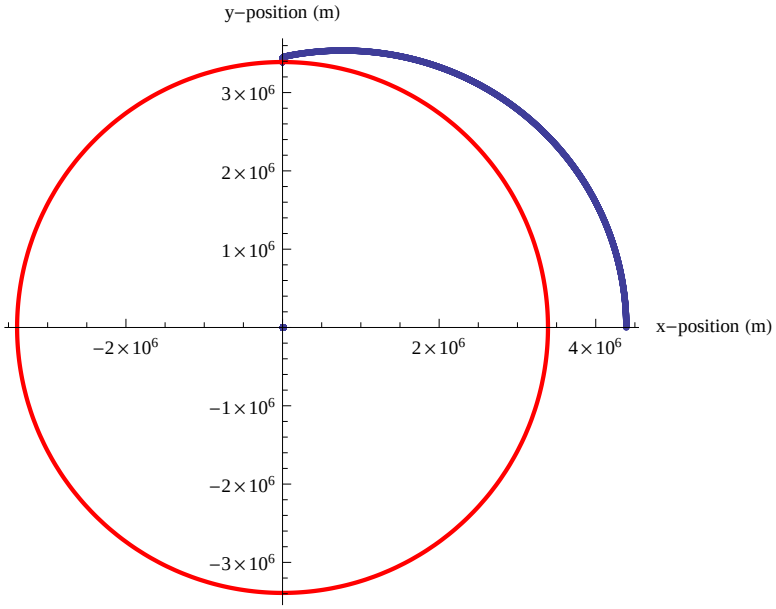
- **(c) Now to hit the north pole. The velocity I got (using 1 second as my step was about 352 (or 353). I did this by trial and error. There are more elegant ways of finding the zero of course.**

```
FillData[16000, 1, 1*^6, -352, True]
```

```
Final i: 3500
Final time: 3500
Final θ/π: 0.500139
Final r: 3.39×10^6
Final v: 15.4289
Final a: 0.0106762
```

```
1.57123
```

```
porbit
```



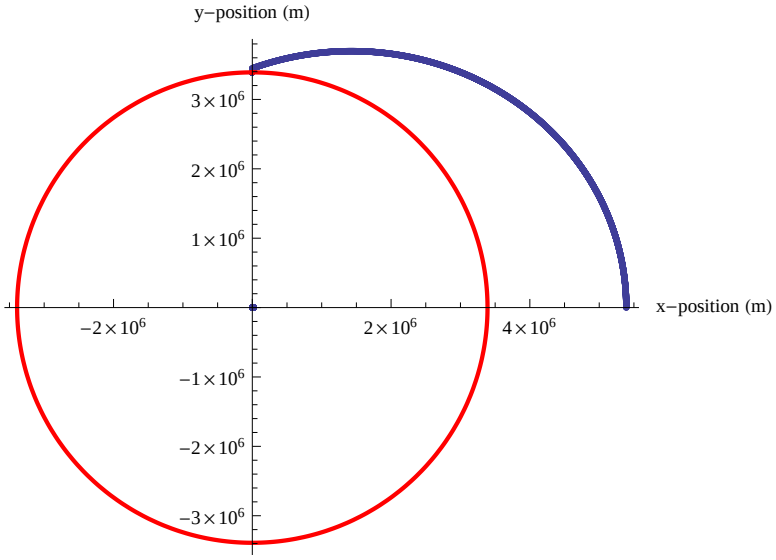
- **(d) At 2000km elevation starting. The velocity I got (using 1 second as my step was about 352 (or 353). Again by trial and error.**

```
FillData[16000, 1, 2*^6, -564, True]
```

```
Final i: 4110
Final time: 4110
Final θ/π: 0.500127
Final r: 3.39×10^6
Final v: 15.4274
Final a: 0.0106742
```

```
1.5712
```

porbit



- **(e) At 4000km elevation starting. The velocity I got (using 1 second as my step was about 352 (or 353). Again by trial and error.**

```
FillData[16 000, 1, 4*^6, -763, True]
```

```
Final i: 5518
```

```
Final time: 5518
```

```
Final θ/π: 0.500673
```

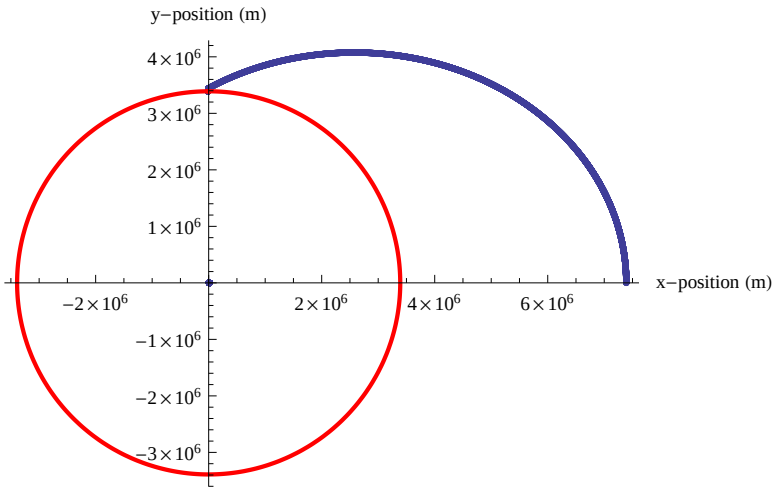
```
Final r: 3.39×10^6
```

```
Final v: 15.4288
```

```
Final a: 0.0106761
```

```
1.57291
```

porbit



- **(f) If my launcher jams at -400, then where should I launch?**

```
FillData[16000, 1, 1*^6, -400, True]
```

```
Final i: 3340
Final time: 3340
Final θ/π: 0.455219
Final r: 3.39001×10^6
Final v: 15.4357
Final a: 0.0106856
```

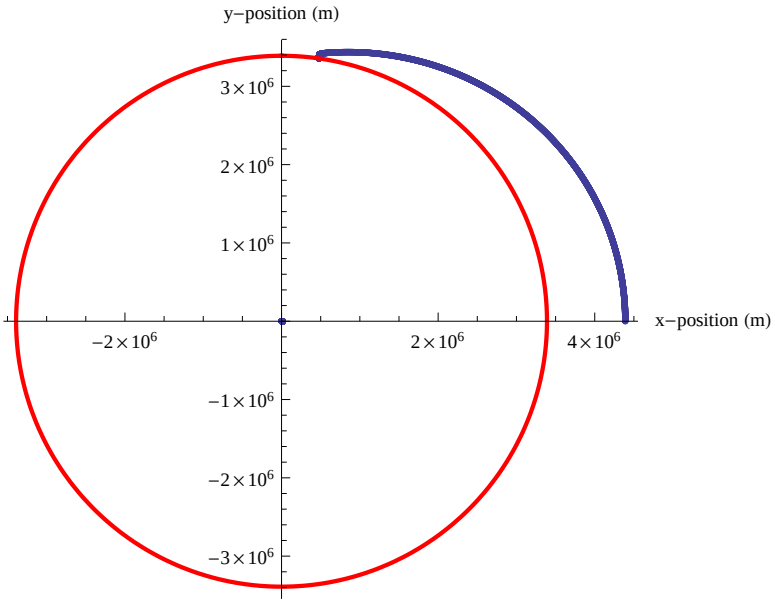
```
1.43011
```

```
90 - 0.4555219 * Pi * 180 / Pi
```

```
8.00606
```

This means that I will hit 8 degrees shy of the north pole, so I should launch when I am at 8 degrees north.

```
porbit
```



- **(g) Plot the angular distance as a function of time step.**

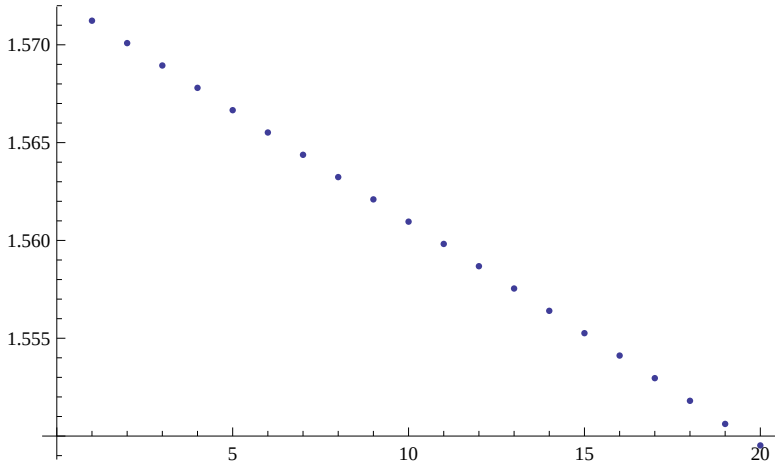
```
dtTestTable = Table[{1 * i, 0}, {i, 1, 20}]
```

```
{{1, 0}, {2, 0}, {3, 0}, {4, 0}, {5, 0}, {6, 0}, {7, 0}, {8, 0}, {9, 0}, {10, 0}, {11, 0},
 {12, 0}, {13, 0}, {14, 0}, {15, 0}, {16, 0}, {17, 0}, {18, 0}, {19, 0}, {20, 0}}
```

```
Do[
 {
  dtTestTable[[i, 2]] = FillData[16000, dtTestTable[[i, 1]], 1*^6, -352, False];
 }, {i, 1, Length[dtTestTable]}]
```

Clearly the value asymptotically approaches a number very close to the number at dt=1.

```
ListPlot[dtTestTable]
```



Since it looks linear, I can find the slope. And since the change in angle from dt=0 to dt=1 is the slope times the change in dt (which equals 1), that is the error in my angle. The percent error is also below. This is my best estimation for how close I am to the true solution (assuming we didn't make any systematic mistakes).

```
m = (dtTestTable[[2, 2]] - dtTestTable[[1, 2]]) /
   (dtTestTable[[2, 1]] - dtTestTable[[1, 1]])
```

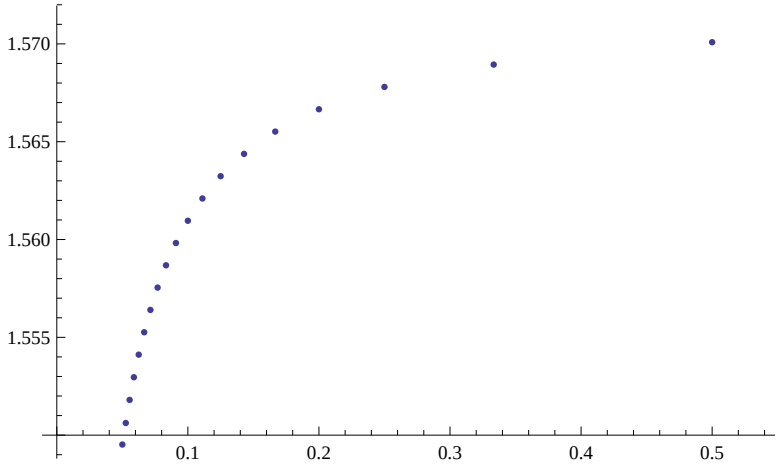
```
-0.00114534
```

```
percentErr = 100 * m / dtTestTable[[1, 2]]
```

```
-0.0728942
```

You can see the asymptotic behavior as well if you plot 1/dt versus the change in angle.

```
ListPlot[
 Table[{1 / dtTestTable[[i, 1]], dtTestTable[[i, 2]]}, {i, 1, Length[dtTestTable]}]]
```



■ Something a little extra.

The function below uses a rudimentary method to home in on the velocity I have to fire the probe from my satellite in order to hit a target angle. It takes a target (the angle you want to hit, Pi/2 for the north pole), all the same stuff that our last module took so it can pass that down to it, vGuessFromSat is our guess for a v relative to the satellite, vStep is the step to take as it tries to find the target, and maxIt is the maximum number of iterations it will use before it gives up. For what we're doing here, you can honestly just guess using the stuff above rather than make a root finder like this. If you wanted to do this the right way, you would use Newton's method to find where the zero occurs.

```
findVFromFinalAngle[target_, tMax_, dt_, h_, vGuessFromSat_, vStep_, maxIt_] :=
  Module[{vCur, fillDataLast, fillDataTest, i, direction},
   vCur = vGuessFromSat;
   Print["vCur=", vCur];
   fillDataLast = FillData[tMax, dt, h, vCur];
   direction = 1;
   Do[
    vCur = vCur + direction * vStep;
    Print["vCur=", vCur];
    Print["θCur=", fillDataLast];
    fillDataTest = FillData[tMax, dt, h, vCur];
    If[Sign[fillDataTest - target] ≠ Sign[fillDataLast - target],
     Print["Solution found v=", vCur];
     Break[];]; (*If the sign of the difference changed,
    you passed through the target, so quit*)
    If[Abs[(fillDataTest - target)] > Abs[(fillDataLast - target)],
     (*You're moving in the wrong direction, so flip the direction*)
     direction = -direction;];
    fillDataLast = fillDataTest;
    , {i, 1, maxIt}];
   vCur
  ];
```