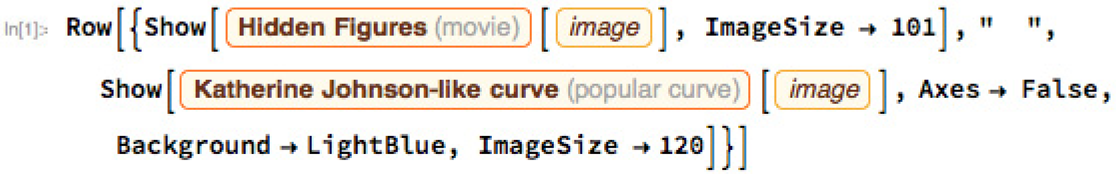
WOLFRAM
COMPUTATION MEETS KNOWLEDGE

Hidden Figures: Modern Approaches to Orbit and Reentry Calculations

February 24, 2017

The movie *Hidden Figures* was released in theaters recently and has been getting good reviews. It also deals with an important time in US history, touching on a number of topics, including civil rights and the Space Race. The movie details the hidden story of Katherine Johnson and her coworkers (Dorothy Vaughan and Mary Jackson) at NASA during the Mercury missions and the United States' early explorations into manned space flight. The movie focuses heavily on the dramatic civil rights struggle of African American women in NASA at the time, and these struggles are set against the number-crunching ability of Johnson and her coworkers. Computers were in their early days at this time, so Johnson and her team's ability to perform complicated navigational orbital mechanics problems without the use of a computer provided an important sanity check against the early computer results.



```
In[1]:= Row[{Show[Hidden Figures (movie) [image], ImageSize → 101], "   ",
   Show[Katherine Johnson-like curve (popular curve) [image], Axes → False,
    Background → LightBlue, ImageSize → 120]}]
```

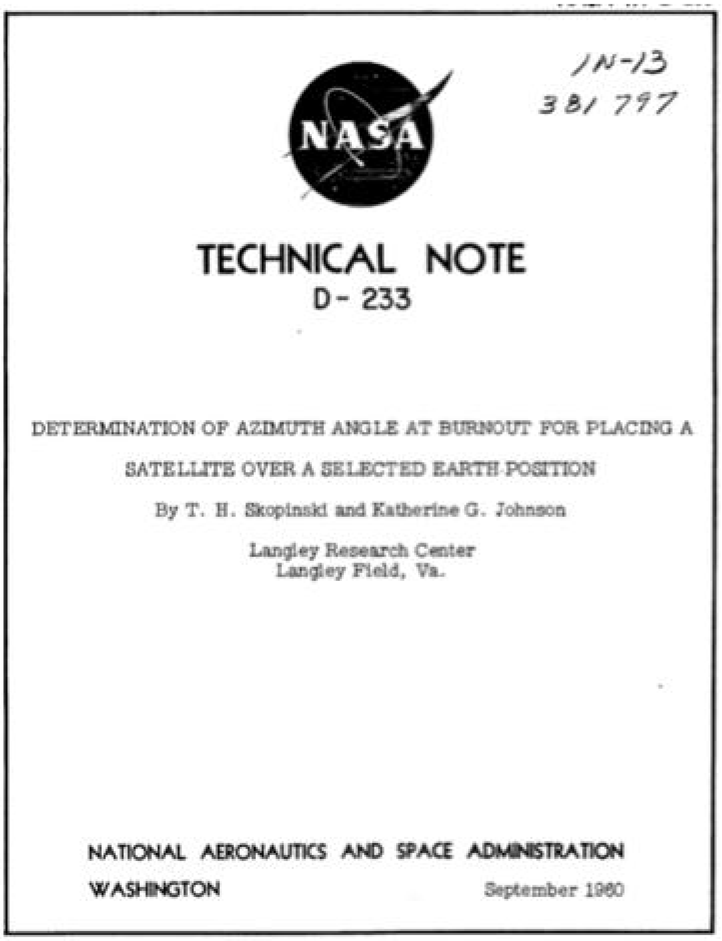
Out[1]=

I will touch on two aspects of her scientific work that were mentioned in the film: orbit calculations and reentry calculations. For the orbit calculation, I will first exactly follow what Johnson did and then compare with a more modern, direct approach utilizing an array of tools made available with the Wolfram Language. Where the movie mentions the solving of differential equations using Euler's method, I will compare this method with more modern ones in an important problem of rocketry: computing a reentry trajectory from the rocket equation and drag terms (derived using atmospheric model data obtained directly from within the Wolfram Language).

The movie doesn't focus much on the math details of the types of problems Johnson and her team dealt with, but for the purposes of this blog, I hope to provide at least a flavor of the approaches one might have used in Johnson's day compared to the present.

Placing a Satellite over a Selected Position

One of the earliest papers that Johnson coauthored, "Determination of Azimuth Angle at Burnout for Placing a Satellite over a Selected Earth Position," deals with the problem of making sure that a satellite can be placed over a specific Earth location after a specified number of orbits, given a certain starting position (e.g. Cape Canaveral, Florida) and orbital trajectory. The approach that Johnson's team used was to determine the azimuthal angle (the angle formed by the spacecraft's velocity vector at the time of engine shutoff with a fixed reference direction, say north) to fire the rocket in, based on other orbital parameters. This is an important step in making sure that an astronaut is in the correct location for reentry to Earth.



IN-13
381 797

NASA

TECHNICAL NOTE
D- 233

DETERMINATION OF AZIMUTH ANGLE AT BURNOUT FOR PLACING A SATELLITE OVER A SELECTED EARTH POSITION

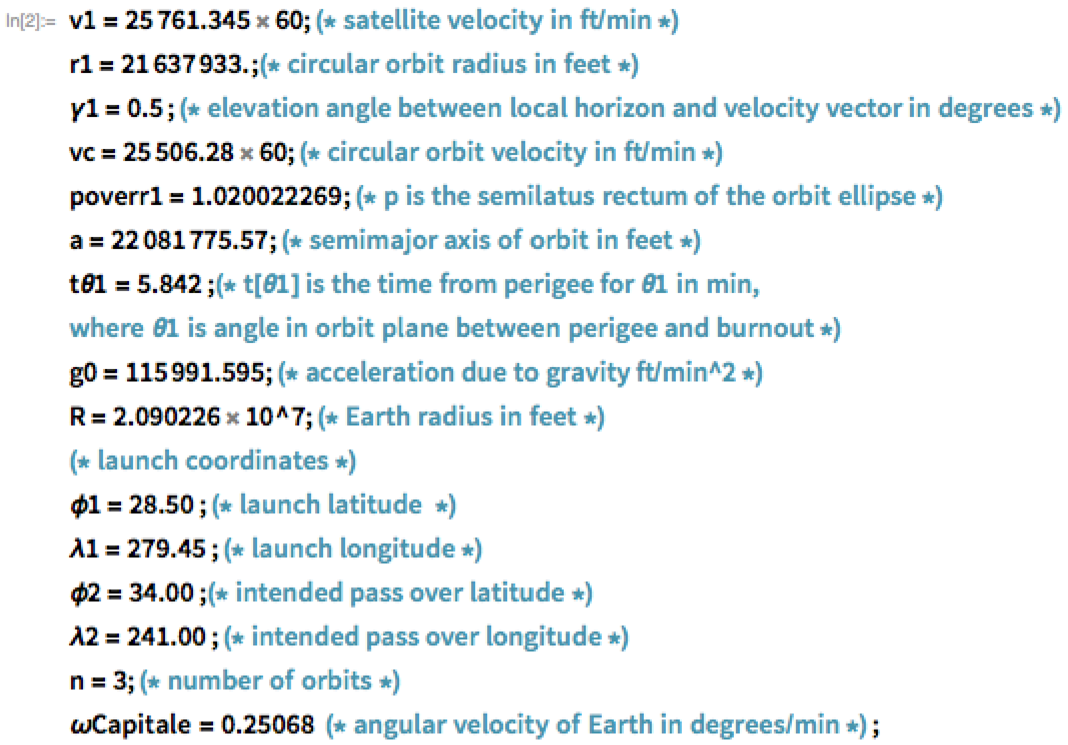
By T. H. Skopinski and Katherine G. Johnson

Langley Research Center
Langley Field, Va.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
WASHINGTON September 1960

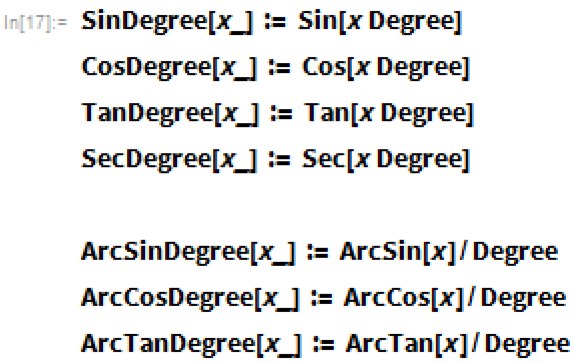
Constants and Initial Processing

In the paper, Johnson defines a number of constants and input parameters needed to solve the problem at hand. One detail to explain is the term "burnout," which refers to the shutoff of the rocket engine. After burnout, orbital parameters are essentially "frozen," and the spacecraft moves solely under the Earth's gravity (as determined, of course, through Newton's laws). In this section, I follow the paper's unit conventions as closely as possible.

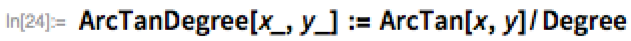


```
In[2]:= v1 = 25 761.345 × 60; (* satellite velocity in ft/min *)
        r1 = 21 637 933.;(* circular orbit radius in feet *)
        γ1 = 0.5; (* elevation angle between local horizon and velocity vector in degrees *)
        vc = 25 506.28 × 60; (* circular orbit velocity in ft/min *)
        poverr1 = 1.020022269; (* p is the semilatus rectum of the orbit ellipse *)
        a = 22 081 775.57; (* semimajor axis of orbit in feet *)
        tθ1 = 5.842 ;(* t[θ1] is the time from perigee for θ1 in min,
        where θ1 is angle in orbit plane between perigee and burnout *)
        g0 = 115 991.595; (* acceleration due to gravity ft/min^2 *)
        R = 2.090226 × 10^7; (* Earth radius in feet *)
        (* launch coordinates *)
        ϕ1 = 28.50 ; (* launch latitude  *)
        λ1 = 279.45 ; (* launch longitude *)
        ϕ2 = 34.00 ;(* intended pass over latitude *)
        λ2 = 241.00 ; (* intended pass over longitude *)
        n = 3; (* number of orbits *)
        ωCapitale = 0.25068 (* angular velocity of Earth in degrees/min *) ;
```

For convenience, some functions are defined to deal with angles in degrees instead of radians. This allows for smoothly handling time in angle calculations:



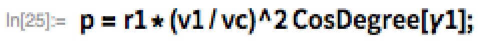
```
In[17]:= SinDegree[x_] := Sin[x Degree]
         CosDegree[x_] := Cos[x Degree]
         TanDegree[x_] := Tan[x Degree]
         SecDegree[x_] := Sec[x Degree]

         ArcSinDegree[x_] := ArcSin[x] / Degree
         ArcCosDegree[x_] := ArcCos[x] / Degree
         ArcTanDegree[x_] := ArcTan[x] / Degree
```



```
In[24]:= ArcTanDegree[x_, y_] := ArcTan[x, y] / Degree
```

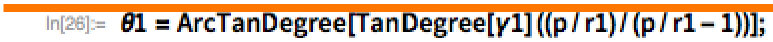
Johnson goes on to describe several other derived parameters, though it's interesting to note that she sometimes adopted values for these rather than using the values returned by her formulas. Her adopted values were often close to the values obtained by the formulas. For simplicity, the values from the formulas are used here.

Semilatus rectum of the orbit ellipse:



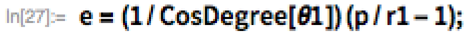
```
In[25]:= p = r1 * (v1 / vc)^2 CosDegree[γ1];
```

Angle in orbit plane between perigee and burnout point:



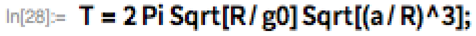
```
In[26]:= θ1 = ArcTanDegree[TanDegree[γ1] ((p / r1) / (p / r1 – 1))];
```

Orbit eccentricity:



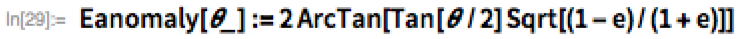
```
In[27]:= e = (1 / CosDegree[θ1]) (p / r1 – 1);
```

Orbit period:



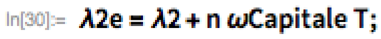
```
In[28]:= T = 2 Pi Sqrt[R / g0] Sqrt[(a / R)^3];
```

Eccentric anomaly:



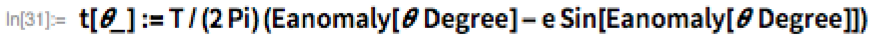
```
In[29]:= Eanomaly[θ_] := 2 ArcTan[Tan[θ / 2] Sqrt[(1 – e) / (1 + e)]]
```

To describe the next parameter, it's easiest to quote the original paper: "The requirement that a satellite with burnout position $\varphi 1$, $\lambda 1$ pass over a selected position $\varphi 2$, $\lambda 2$ after the completion of n orbits is equivalent to the requirement that, during the first orbit, the satellite pass over an equivalent position with latitude $\varphi 2$ the same as that of the selected position but with longitude $\lambda 2e$ displaced eastward from $\lambda 2$ by an amount sufficient to compensate for the rotation of the Earth during the n complete orbits, that is, by the polar hour angle $n\ \omega_E\ T$. The longitude of this equivalent position is thus given by the relation":



```
In[30]:= λ2e = λ2 + n ωCapitale T;
```

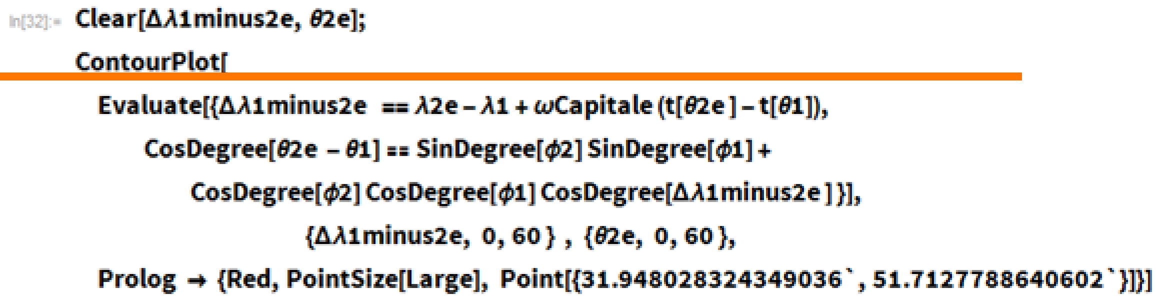
Time from perigee for angle θ:



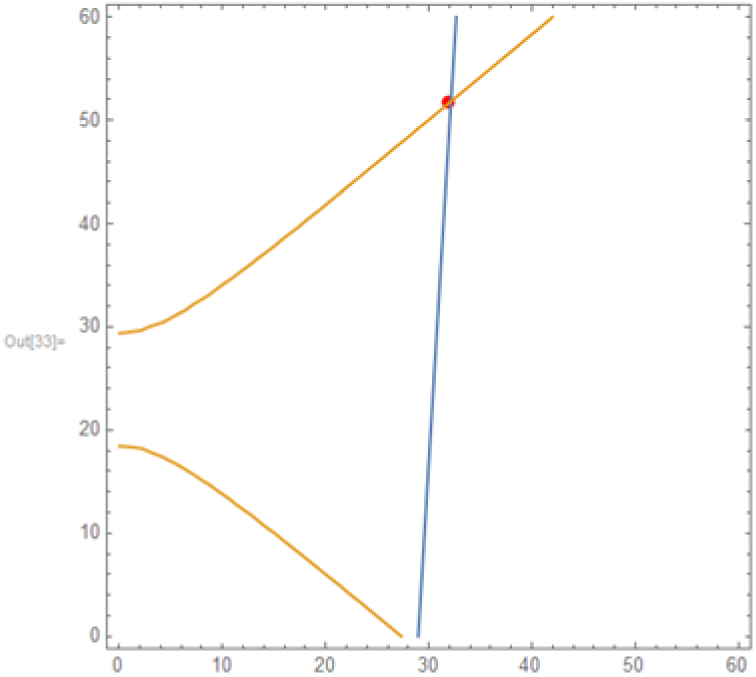
```
In[31]:= t[θ_] := T / (2 Pi) (Eanomaly[θ Degree] – e Sin[Eanomaly[θ Degree]])
```

Computation

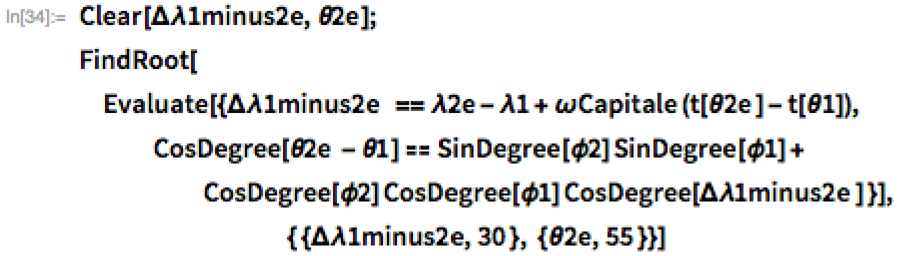
Part of the final solution is to determine values for intermediate parameters $\delta\lambda_{1\text{-}2e}$ and θ_{2e}. This can be done in a couple of ways. First, I can use **ContourPlot** to obtain a graphical solution via equations 19 and 20 from the paper:



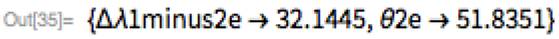
```
In[32]:= Clear[Δλ1minus2e, θ2e];
         ContourPlot[
           Evaluate[{Δλ1minus2e  == λ2e - λ1 + ωCapitale (t[θ2e] - t[θ1]),
               CosDegree[θ2e  - θ1] == SinDegree[ϕ2] SinDegree[ϕ1] +
                   CosDegree[ϕ2] CosDegree[ϕ1] CosDegree[Δλ1minus2e ]}],
                           {Δλ1minus2e, 0, 60 } , {θ2e, 0, 60 },
           Prolog → {Red, PointSize[Large], Point[{31.948028324349036` , 51.7127788640602` }]}]
```



FindRoot can be used to find the solutions numerically:

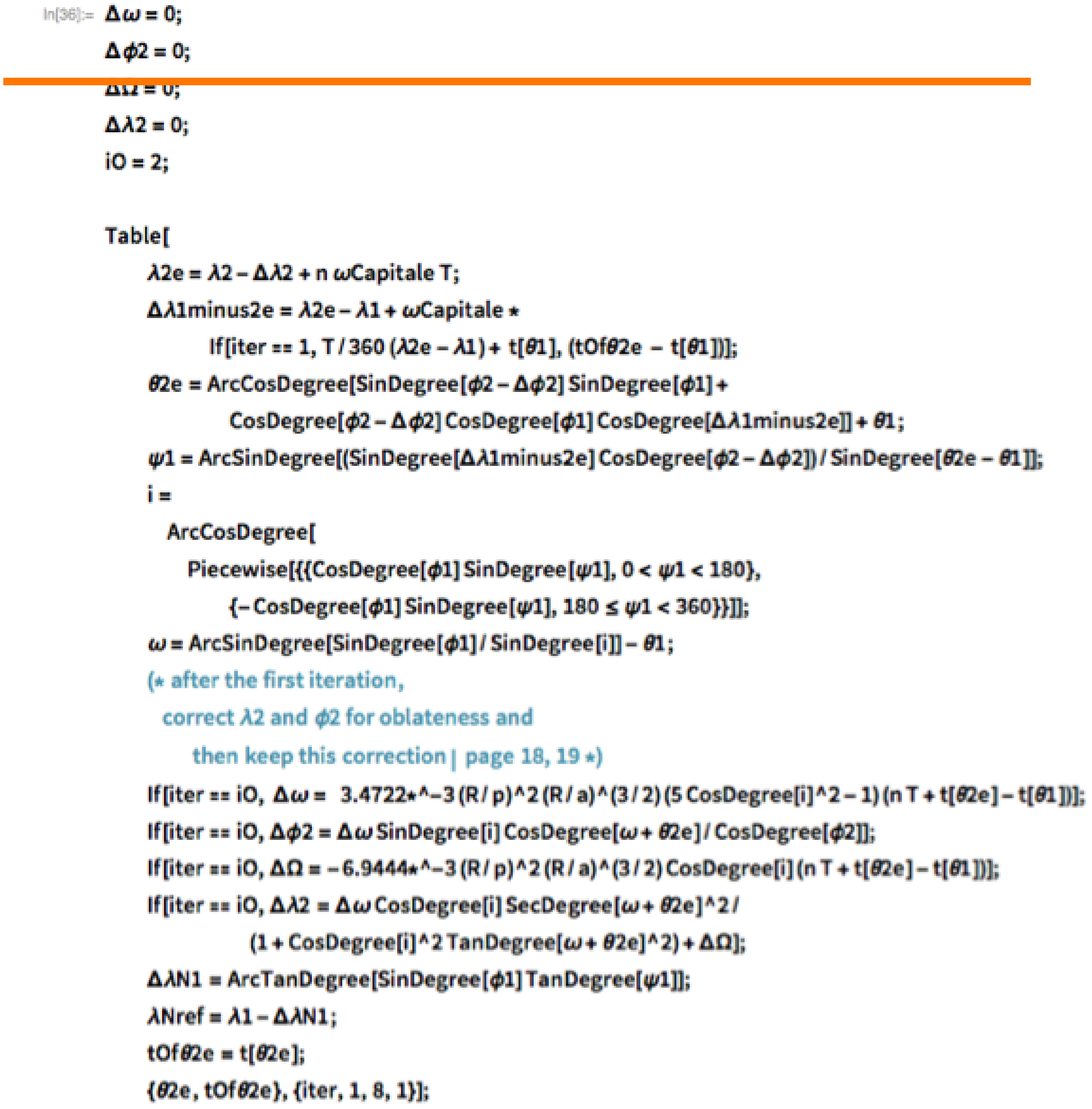


```
In[34]:= Clear[Δλ1minus2e, θ2e];
         FindRoot[
           Evaluate[{Δλ1minus2e  == λ2e - λ1 + ωCapitale (t[θ2e] - t[θ1]),
               CosDegree[θ2e  - θ1] == SinDegree[ϕ2] SinDegree[ϕ1] +
                   CosDegree[ϕ2] CosDegree[ϕ1] CosDegree[Δλ1minus2e ]}],
                         {{Δλ1minus2e, 30}, {θ2e, 55 }}]
```



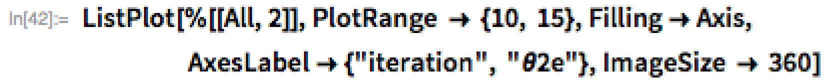
```
Out[35]= {Δλ1minus2e → 32.1445, θ2e → 51.8351}
```

Of course, Johnson didn't have access to **ContourPlot** or **FindRoot**, so her paper describes an iterative technique. I translated the technique described in the paper into the Wolfram Language, and also solved for a number of other parameters via her iterative method. Because the base computations are for a spherical Earth, corrections for oblateness are included in her method:



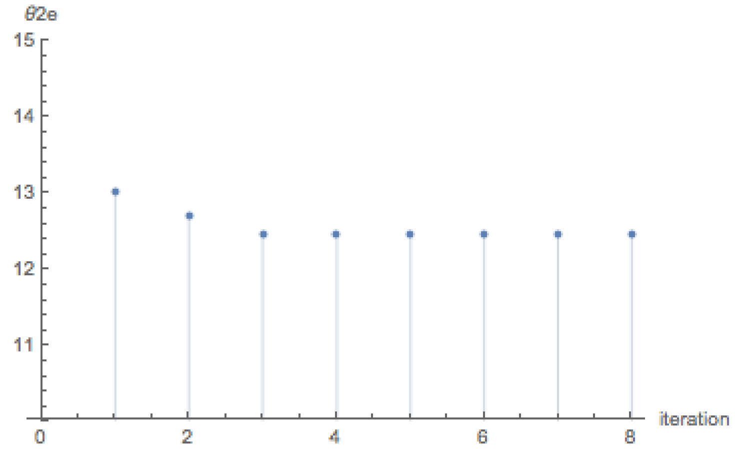
```
In[36]:= Δω = 0;
         Δϕ2 = 0;
         ΔΩ = 0;
         Δλ2 = 0;
         iO = 2;

         Table[
           λ2e = λ2 − Δλ2 + n ωCapitale T;
           Δλ1minus2e = λ2e − λ1 + ωCapitale *
                If[iter == 1, T / 360 (λ2e − λ1) + t[θ1], (tOfθ2e − t[θ1])];
           θ2e = ArcCosDegree[SinDegree[ϕ2 − Δϕ2] SinDegree[ϕ1] +
                   CosDegree[ϕ2 − Δϕ2] CosDegree[ϕ1] CosDegree[Δλ1minus2e]] + θ1;
           ψ1 = ArcSinDegree[(SinDegree[Δλ1minus2e] CosDegree[ϕ2 − Δϕ2]) / SinDegree[θ2e − θ1]];
           i =
             ArcCosDegree[
               Piecewise[{{CosDegree[ϕ1] SinDegree[ψ1], 0 < ψ1 < 180},
                   {−CosDegree[ϕ1] SinDegree[ψ1], 180 ≤ ψ1 < 360}}]];
           ω = ArcSinDegree[SinDegree[ϕ1] / SinDegree[i]] − θ1;
           (* after the first iteration,
             correct λ2 and ϕ2 for oblateness and
                then keep this correction | page 18, 19 *)
           If[iter == iO, Δω =  3.4722*^-3 (R / p)^2 (R / a)^(3 / 2) (5 CosDegree[i]^2 − 1) (n T + t[θ2e] − t[θ1])];
           If[iter == iO, Δϕ2 = Δω SinDegree[i] CosDegree[ω + θ2e] / CosDegree[ϕ2]];
           If[iter == iO, ΔΩ = −6.9444*^-3 (R / p)^2 (R / a)^(3 / 2) CosDegree[i] (n T + t[θ2e] − t[θ1])];
           If[iter == iO, Δλ2 = Δω CosDegree[i] SecDegree[ω + θ2e]^2 /
                    (1 + CosDegree[i]^2 TanDegree[ω + θ2e]^2) + ΔΩ];
           ΔλN1 = ArcTanDegree[SinDegree[ϕ1] TanDegree[ψ1]];
           λNref = λ1 − ΔλN1;
           tOfθ2e = t[θ2e];
           {θ2e, tOfθ2e}, {iter, 1, 8, 1}];
```

Graphing the value of θ2e for the various iterations shows a quick convergence:



```
In[42]:= ListPlot[%[[All, 2]], PlotRange → {10, 15}, Filling → Axis,
               AxesLabel → {"iteration", "θ2e"}, ImageSize → 360]
```

Out[42]=



```
In[43]:= ksol = {
           "Δλ1minus2e" → Δλ1minus2e, "Δλ2" → Δλ2, "θ2e" → θ2e , "ψ1" → ψ1, "ω" → ω,
           "Δω" → Δω, "ΔΩ" → ΔΩ, "i" → N@i, "Δϕ2" → Δϕ2}

Out[43]= {Δλ1minus2e → 31.0612, Δλ2 → 1.02742, θ2e → 50.9335, ψ1 → 70.5551,
          ω → 34.5578, Δω → 1.96276, ΔΩ → −1.33792, i → 34.0355, Δϕ2 → 0.0841712}
```

I can convert the method in a **FindRoot** command as follows (this takes the oblateness effects into account in a fully self-consistent manner and calculates values for all nine variables involved in the equations):

```
In[44]:= fr := FindRoot[Evaluate[
            {Δλ1minus2e  == (λ2 − Δλ2 + n ωCapitale T) − λ1 + ωCapitale * (t[θ2e] − t[θ1]),
              CosDegree[θ2e − θ1] == SinDegree[ϕ2 − Δϕ2] SinDegree[ϕ1] +
                  CosDegree[ϕ2 − Δϕ2] CosDegree[ϕ1] CosDegree[Δλ1minus2e],
              SinDegree[ψ1] == (SinDegree[Δλ1minus2e] CosDegree[ϕ2 − Δϕ2]) / SinDegree[θ2e − θ1],
              CosDegree[i] == CosDegree[ϕ1] SinDegree[ψ1],
              SinDegree[ω + θ1] == SinDegree[ϕ1] / SinDegree[i],
              Δω ==  3.4722*^−3 (R / p)^2 (R / a)^(3 / 2) (5 CosDegree[i]^2 − 1) (n T + t[θ2e] − t[θ1]),
              Δϕ2 == Δω SinDegree[i] CosDegree[ω + θ2e] / CosDegree[ϕ2 − Δϕ2],
              ΔΩ == −6.9444*^−3 (R / p)^2 (R / a)^(3 / 2) CosDegree[i] (n T + t[θ2e] − t[θ1]),
              Δλ2 ==
                Δω CosDegree[i] SecDegree[ω + θ2e]^2 / (1 + CosDegree[i]^2 TanDegree[ω + θ2e]^2) + ΔΩ}],
          {{Δλ1minus2e, 31}, {Δλ2, 1}, {θ2e, 51}, {ψ1, 70}, {ω, 34}, {Δω, 2}, {ΔΩ, −1.3}, {i, 34}, {Δϕ2, 0.1}}]

In[45]:= Clear[Δλ1minus2e, Δλ2, θ2e, ψ1, ω, Δω, ΔΩ, i, Δϕ2];
         wlsol = fr

Out[46]= {Δλ1minus2e → 31.062, Δλ2 → 1.02664, θ2e → 50.9332, ψ1 → 70.5964,
          ω → 34.61, Δω → 1.96527, ΔΩ → −1.33778, i → 34.0139, Δϕ2 → 0.102921}
```

Interestingly, even the iterative root-finding steps of this more complicated system converge quite quickly:

```
In[47]:= (OwnValues[fr] /. HoldPattern[FindRoot[args__]] :→ Reap[FindRoot[args, StepMonitor :→ Sow[
                           {Δλ1minus2e, Δλ2, θ2e, ψ1, ω, Δω, ΔΩ, i, Δϕ2}]]])[[1, 2]][[2]]

Out[47]= {{{31.0619, 1.02671, 50.9332, 70.5861, 34.6025, 1.96521, −1.33777, 34.0149, 0.102885},
           {31.062, 1.02664, 50.9332, 70.5964, 34.61, 1.96527, −1.33778, 34.0139, 0.102922},
           {31.062, 1.02664, 50.9332, 70.5964, 34.61, 1.96527, −1.33778, 34.0139, 0.102921},
           {31.062, 1.02664, 50.9332, 70.5964, 34.61, 1.96527, −1.33778, 34.0139, 0.102921}}}
```

Plotting

With the orbital parameters determined, it is desirable to visualize the solution. First, some critical parameters from the previous solutions need to be extracted:

```
In[48]:= {θ2e, ω, i} = {"θ2e", "ω", "i"} /. ksol;
```

Next, the latitude and longitude of the satellite as a function of azimuth angle need to be derived:

```
In[49]:= ΔλNs[θs_] := ArcTanDegree[CosDegree[ω + θs], CosDegree[- i] SinDegree[ω + θs]]
```

```
In[50]:= λN[θs_] := λNref - ωCapitale (t[θs] - t[θ1]) + ΔλNs[θs]
```

*φ*s and *λ*s are the latitudes and longitudes as a function of *θ*s:

```
In[51]:= ϕs[θs_] := ArcSinDegree[SinDegree[i] SinDegree[ω + θs]]
```

```
In[52]:= λs[θs_, n_] := Mod[λNref - ωCapitale (n T + t[θs] - t[θ1]) + ΔλNs[θs], 360]
```

The satellite ground track can be constructed by creating a table of points:

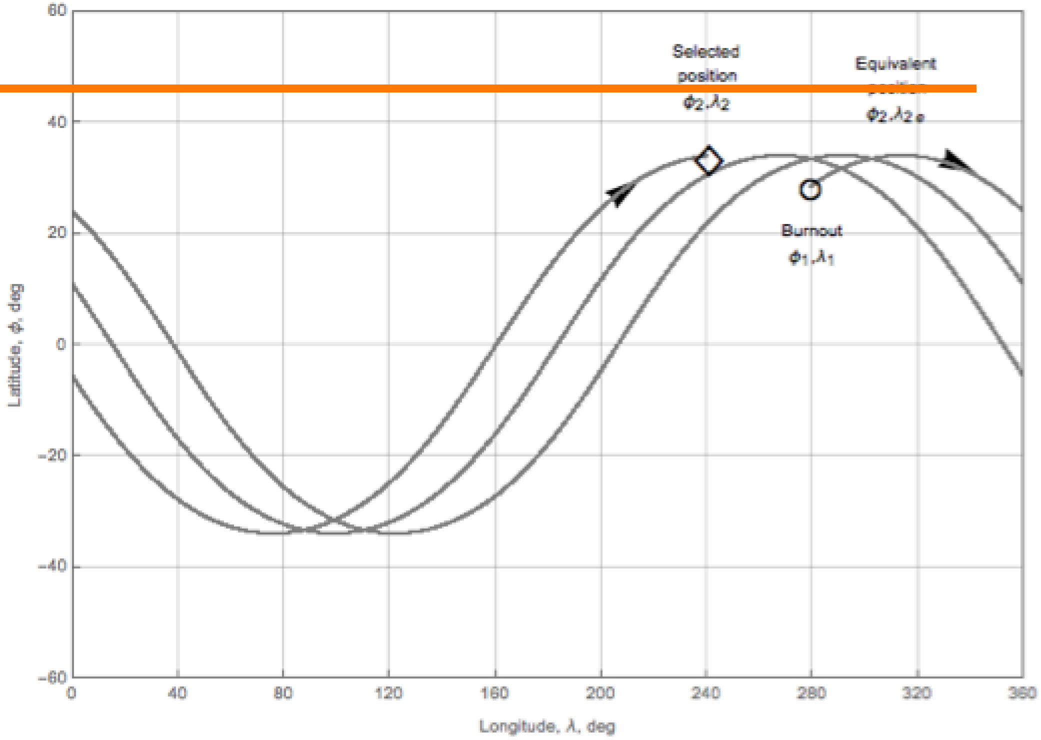
```
In[53]:= satpts[n_] := Table[{ϕs[θs], λs[θs, n]}, {θs, 0, 360, .01}];
```

Johnson’s paper presents a sketch of the orbital solution including markers showing the burnout, selected and equivalent positions. It’s easy to reproduce a similar plain diagram here:

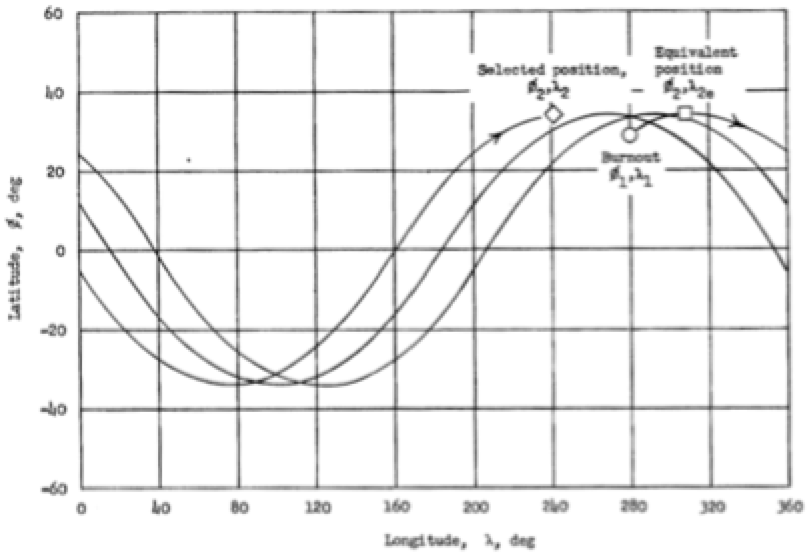
```
In[54]:= points = {Table[{λs[θs, 0], ϕs[θs]}, {θs, θ1, 180, .1}],
           Table[{λs[θs, 1], ϕs[θs]}, {θs, 0, 360, .1}],
           Table[{λs[θs, 2], ϕs[θs]}, {θs, 0, 360, .1}],
           Table[{λs[θs, 3], ϕs[θs]}, {θs, -180, θ2e, .1}]};
```

```
In[55]:= ListPlot[points,
           PlotStyle → Gray, PlotRange → {{0, 360}, {-60, 60}},
           Epilog → {Black, Text[Style[#1, 32, Bold], #2] &@@@
                     {{"○", {λ1, ϕ1}}, {"◇", {λ2, ϕ2}}, {"□", {λ2e, ϕ2}}}},
           Prolog → {Arrow[{{333, 33}, {342, 31}}], Arrow[{{212, 29}, {215, 30}}],
                      Text[Style["Burnout\nϕ1,λ1", 10], {280, 18}],
                      Text[Style["Selected\nposition\nϕ2,λ2", 10], {240, 48}],
                      Text[Style["Equivalent\nposition\nϕ2,λ2e", 10], {312, 46}]},
           GridLines → {Table[λ, {λ, 0, 360, 40}], Table[ϕ, {ϕ, -60, 60, 20}]},
           Axes → False, Frame → True, ImageSize → 600, AspectRatio → 0.7,
           FrameLabel → {"Longitude, λ, deg", "Latitude, ϕ, deg"},
           FrameTicks → {{Table[{ϕ, ϕ}, {ϕ, -60, 60, 20}], None}, {Table[{λ, λ}, {λ, 0, 360, 40}], None}}]
```

Out[55]=



For comparison, here is her original diagram:



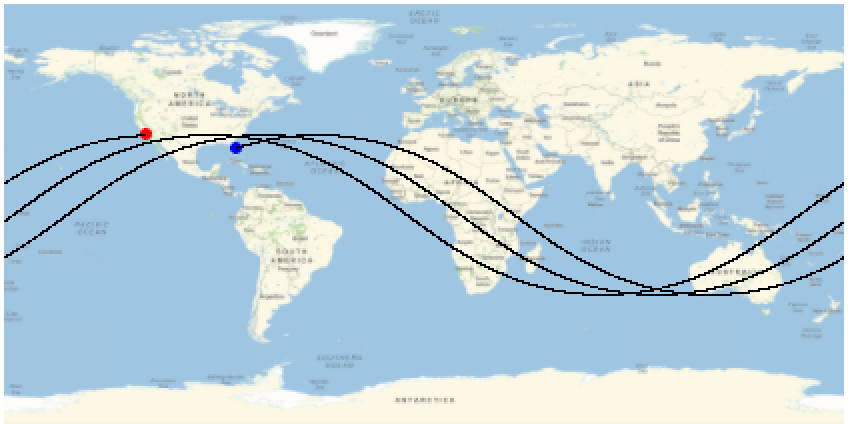
```
In[56]:= path = {#1,  QuantityMagnitude[GeodesyData["Krassovsky",
                    {"FromGeocentricLatitude", #2}]]} & @@@ Flatten[points, 1];
```

A more visually useful version can be constructed using **GeoGraphics**, taking care to convert the geocentric coordinates into geodetic coordinates:

```
In[56]:= path = {#1,  QuantityMagnitude[GeodesyData["Krassovsky",
                    {"FromGeocentricLatitude", #2}]]} & @@@ Flatten[points, 1];
```

```
In[57]:= GeoGraphics[{Blue, PointSize[0.015], Point[GeoPosition[{ϕ1, λ1}]], Red,
           Point[GeoPosition[{ϕ2, λ2}]],
           Black, PointSize[0.002], Point[GeoPosition[Reverse /@ path]]}, GeoRange → "World",
         GeoZoomLevel → 3]
```

Out[57]=



How to Calculate Orbits Today

Today, virtually every one of us has, within immediate reach, access to computational resources far more powerful than those available to the entirety of NASA in the 1960s. Now, using only a desktop computer and the Wolfram Language, you can easily find direct numerical solutions to problems of orbital mechanics such as those posed to Katherine Johnson and her team. While perhaps less taxing of our ingenuity than older methods, the results one can get from these explorations are no less interesting or useful.

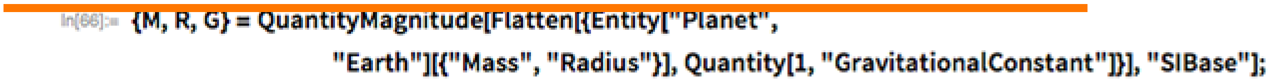
To solve for the azimuthal angle ψ using more modern methods, let's set up parameters for a simple circular orbit beginning after burnout over Florida, assuming a spherically symmetric Earth (I'll not bother trying to match the orbit of the Johnson paper precisely, and I'll redefine certain quantities from above using the modern SI system of units). Starting from the same low-Earth orbit altitude used by Johnson, and using a little spherical trigonometry, it is straightforward to derive the initial conditions for our orbit:

```
In[58]:= ϕ1 = 28.50 Degree (* latitude of burnout point*);
         λ1 = (279.45 – 360) Degree (* longitude of burnout point *);
```

```
In[60]:= (* assuming a circular orbit, and leaving azimuth angle ψ undetermined *)
         r0 = QuantityMagnitude[Quantity[21 637 933., "Feet"], "Meters"] (* radius of the orbit *);
         v0 = Sqrt[G M / r0] (* initial velocity of the circular orbit *);
         ℛ = RotationTransform[λ1, {0, 0, 1}] (* used to rotate initial conditions *);
         {x0, y0, z0} = ℛ[{r0 Cos[ϕ1], 0, r0 Sin[ϕ1]}] (* initial x,y,z at burnout point *);
         {vx0, vy0, vz0} = ℛ[v0 {– Cos[ϕ1] Sin[γ] – Cos[γ] Cos[ψ] Sin[ϕ1],
                 Cos[γ] Sin[ψ], Cos[γ] Cos[ϕ1] Cos[ψ] – Sin[γ] Sin[ϕ1]}](* initial velocity at the burnout point *)
```

```
In[65]:= initCond[ψ_, γ_] = {x[0] == x0, y[0] == y0, z[0] == z0,
                              x'[0] == vx0, y'[0] == vy0, z'[0] == vz0};
```

The relevant physical parameters can be obtained directly from within the Wolfram Language:



```
In[66]:= {M, R, G} = QuantityMagnitude[Flatten[{Entity["Planet",
                         "Earth"][{"Mass", "Radius"}], Quantity[1, "GravitationalConstant"]}], "SIBase"];
```

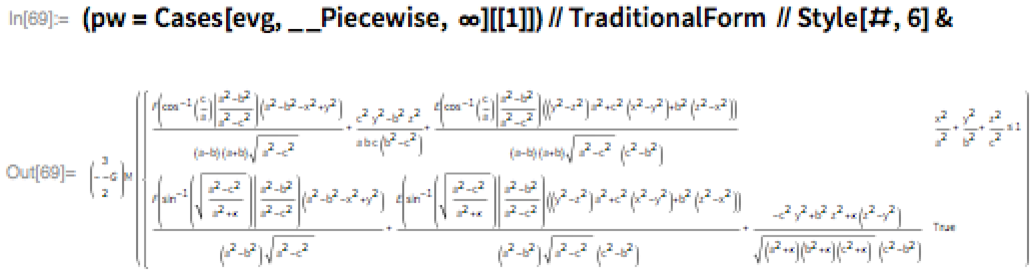
Next, I obtain a differential equation for the motion of our spacecraft, given the gravitational field of the Earth. There are several ways you can model the gravitational potential near the Earth. Assuming a spherically symmetric planet and utilizing a Cartesian coordinate system throughout, the potential is merely:

```
In[67]:= pot[{x_, y_, z_}] := -G M / Sqrt[x^2 + y^2 + z^2];
```

Alternatively, you can use a more realistic model of Earth's gravity, where the planet's shape is taken to be an oblate ellipsoid of revolution. The exact form of the potential from such an ellipsoid (assuming constant mass-density over ellipsoidal shells), though complicated (containing multiple elliptic integrals), is available through **EntityValue**:

```
In[68]:= evg = EntityValue[ massive triaxial ellipsoid (physical system) , "GravitationalPotential"];
```

For a general homogeneous triaxial ellipsoid, the potential contains piecewise functions:



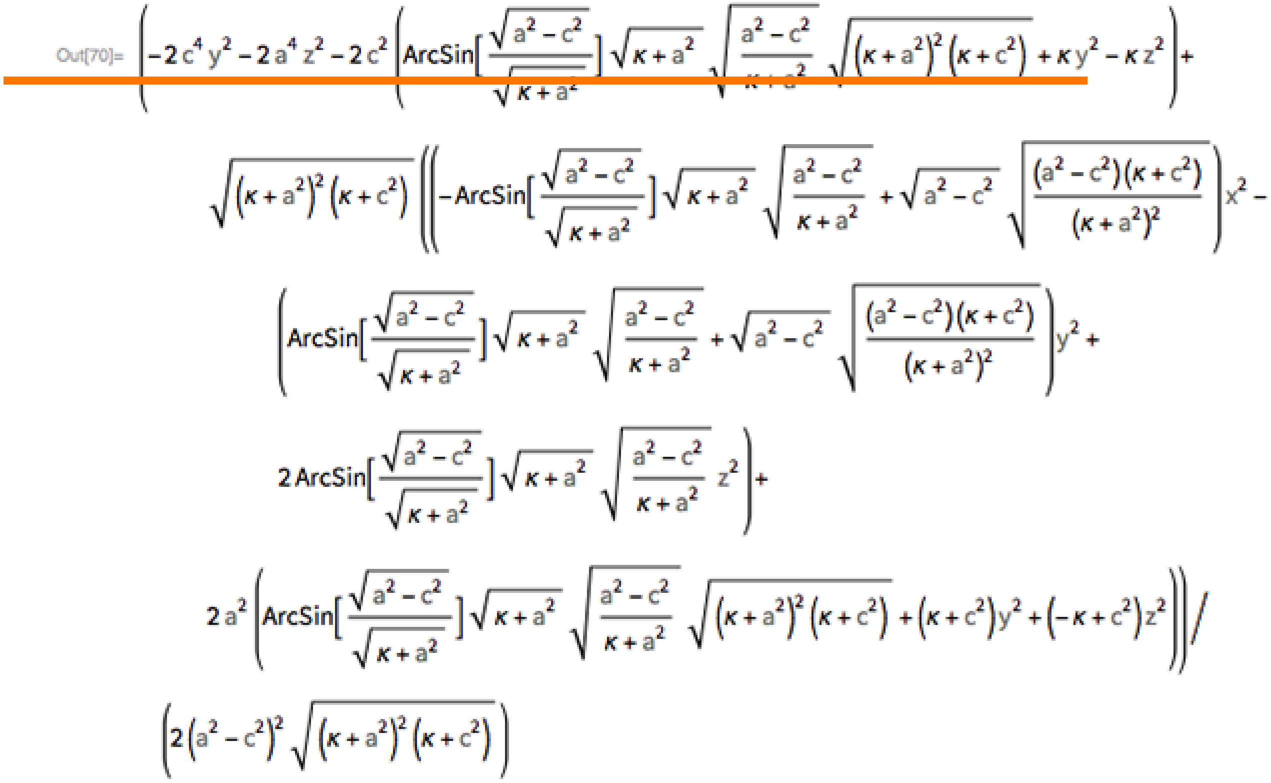
```
In[69]:= (pw = Cases[evg, __Piecewise, ∞][[1]]) // TraditionalForm // Style[#, 6] &
```

Out[69]=

$$\left(-\frac{3}{2}G\right)M\left(\begin{cases} \frac{F\left(\cos^{-1}\left(\frac{c}{a}\right)\middle|\frac{a^2-b^2}{a^2-c^2}\right)\left(a^2-b^2-x^2+y^2\right)}{(a-b)(a+b)\sqrt{a^2-c^2}}+\frac{c^2y^2-b^2z^2}{abc\left(b^2-c^2\right)}+\frac{E\left(\cos^{-1}\left(\frac{c}{a}\right)\middle|\frac{a^2-b^2}{a^2-c^2}\right)\left(\left(y^2-z^2\right)a^2+c^2\left(x^2-y^2\right)+b^2\left(z^2-x^2\right)\right)}{(a-b)(a+b)\sqrt{a^2-c^2}\left(c^2-b^2\right)} & \frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}\le 1 \\ \frac{F\left(\sin^{-1}\left(\sqrt{\frac{a^2-c^2}{a^2+\kappa}}\right)\middle|\frac{a^2-b^2}{a^2-c^2}\right)\left(a^2-b^2-x^2+y^2\right)}{\left(a^2-b^2\right)\sqrt{a^2-c^2}}+\frac{E\left(\sin^{-1}\left(\sqrt{\frac{a^2-c^2}{a^2+\kappa}}\right)\middle|\frac{a^2-b^2}{a^2-c^2}\right)\left(\left(y^2-z^2\right)a^2+c^2\left(x^2-y^2\right)+b^2\left(z^2-x^2\right)\right)}{\left(a^2-b^2\right)\sqrt{a^2-c^2}\left(c^2-b^2\right)}+\frac{-c^2y^2+b^2z^2+\kappa\left(z^2-y^2\right)}{\sqrt{\left(a^2+\kappa\right)\left(b^2+\kappa\right)\left(c^2+\kappa\right)}\left(c^2-b^2\right)} & \text{True} \end{cases}\right)$$

Here, *κ* is the largest root of $x^2/(a^2+\kappa)+y^2/(b^2+\kappa)+z^2/(c^2+\kappa)=1$. In the case of an oblate ellipsoid, the previous formula can be simplified to contain only elementary functions…

```
In[70]:= Limit[pw[[1, -1]], b -> a] // FullSimplify
```



Out[70]=

$$\Bigg(-2c^4y^2-2a^4z^2-2c^2\left(\mathrm{ArcSin}\left[\frac{\sqrt{a^2-c^2}}{\sqrt{\kappa+a^2}}\right]\sqrt{\kappa+a^2}\sqrt{\frac{a^2-c^2}{\kappa+a^2}}\sqrt{\left(\kappa+a^2\right)^2\left(\kappa+c^2\right)}+\kappa y^2-\kappa z^2\right)+$$

$$\sqrt{\left(\kappa+a^2\right)^2\left(\kappa+c^2\right)}\Bigg(\left(-\mathrm{ArcSin}\left[\frac{\sqrt{a^2-c^2}}{\sqrt{\kappa+a^2}}\right]\sqrt{\kappa+a^2}\sqrt{\frac{a^2-c^2}{\kappa+a^2}}+\sqrt{a^2-c^2}\sqrt{\frac{\left(a^2-c^2\right)\left(\kappa+c^2\right)}{\left(\kappa+a^2\right)^2}}\right)x^2-$$

$$\left(\mathrm{ArcSin}\left[\frac{\sqrt{a^2-c^2}}{\sqrt{\kappa+a^2}}\right]\sqrt{\kappa+a^2}\sqrt{\frac{a^2-c^2}{\kappa+a^2}}+\sqrt{a^2-c^2}\sqrt{\frac{\left(a^2-c^2\right)\left(\kappa+c^2\right)}{\left(\kappa+a^2\right)^2}}\right)y^2+$$

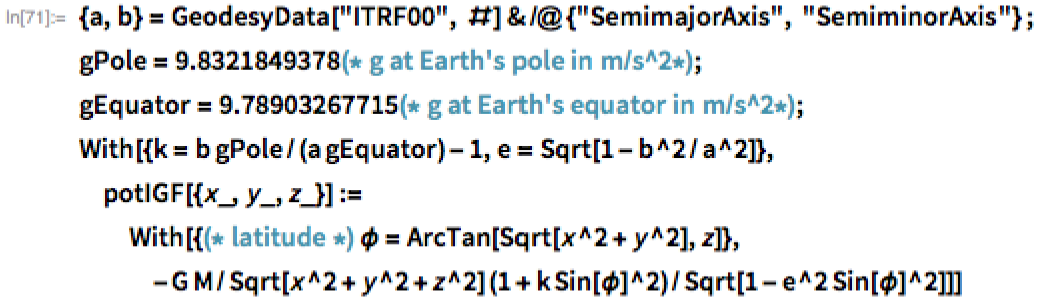
$$2\,\mathrm{ArcSin}\left[\frac{\sqrt{a^2-c^2}}{\sqrt{\kappa+a^2}}\right]\sqrt{\kappa+a^2}\sqrt{\frac{a^2-c^2}{\kappa+a^2}}\,z^2\Bigg)+$$

$$2a^2\left(\mathrm{ArcSin}\left[\frac{\sqrt{a^2-c^2}}{\sqrt{\kappa+a^2}}\right]\sqrt{\kappa+a^2}\sqrt{\frac{a^2-c^2}{\kappa+a^2}}\sqrt{\left(\kappa+a^2\right)^2\left(\kappa+c^2\right)}+\left(\kappa+c^2\right)y^2+\left(-\kappa+c^2\right)z^2\right)\Bigg)\Big/$$

$$\left(2\left(a^2-c^2\right)^2\sqrt{\left(\kappa+a^2\right)^2\left(\kappa+c^2\right)}\right)$$

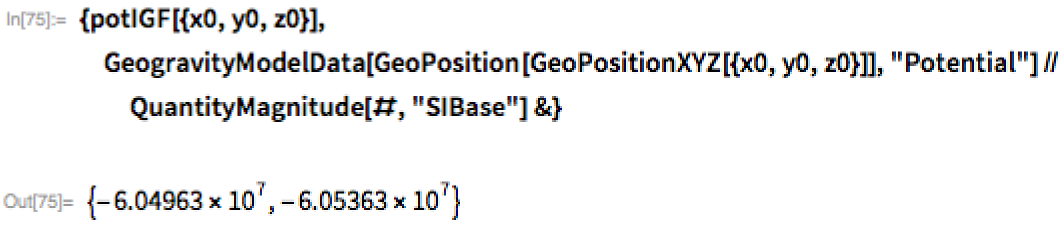
… where $\kappa=((2\,z^2\,(a^2–c^2+x^2+y^2)+(-a^2+c^2+x^2+y^2)^2+z^4)^{1/2}–a^2–c^2+x^2+y^2+z^2)/2$.

A simpler form that is widely used in the geographic and space science community, and that I will use here, is given by the so-called International Gravity Formula (IGF). The IGF takes into account differences from a spherically symmetric potential up to second order in spherical harmonics, and gives numerically indistinguishable results from the exact potential referenced previously. In terms of four measured geodetic parameters, the IGF potential can be defined as follows:



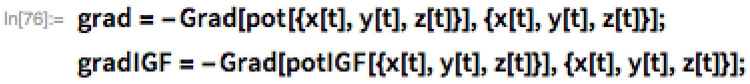
```
In[71]:= {a, b} = GeodesyData["ITRF00", #] &/@{"SemimajorAxis", "SemiminorAxis"};
         gPole = 9.8321849378(* g at Earth's pole in m/s^2*);
         gEquator = 9.78903267715(* g at Earth's equator in m/s^2*);
         With[{k = b gPole / (a gEquator) - 1, e = Sqrt[1 - b^2 / a^2]},
           potIGF[{x_, y_, z_}] :=
             With[{(* latitude *) ϕ = ArcTan[Sqrt[x^2 + y^2], z]},
               -G M / Sqrt[x^2 + y^2 + z^2] (1 + k Sin[ϕ]^2) / Sqrt[1 - e^2 Sin[ϕ]^2]]]
```

I could easily use even better values for the gravitational force through **GeogravityModelData**. For the starting position, the IGF potential deviates only 0.06% from a high-order approximation:

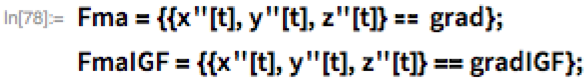


```
In[75]:= {potIGF[{x0, y0, z0}],
           GeogravityModelData[GeoPosition[GeoPositionXYZ[{x0, y0, z0}]], "Potential"] //
             QuantityMagnitude[#, "SIBase"] &}

Out[75]= {-6.04963 × 10^7, -6.05363 × 10^7}
```

With these functional forms for the potential, finding the orbital path amounts to taking a gradient of the potential to get the gravitational field vector and then applying Newton's third law. Doing so, I obtain the orbital equations of motion for the two gravity models:

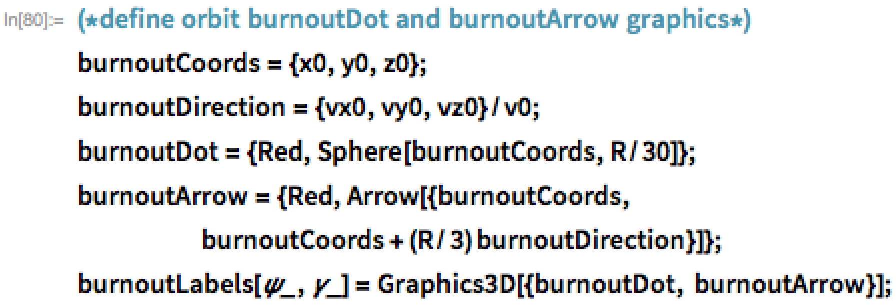


```
In[76]:= grad = -Grad[pot[{x[t], y[t], z[t]}], {x[t], y[t], z[t]}];
         gradIGF = -Grad[potIGF[{x[t], y[t], z[t]}], {x[t], y[t], z[t]}];
```

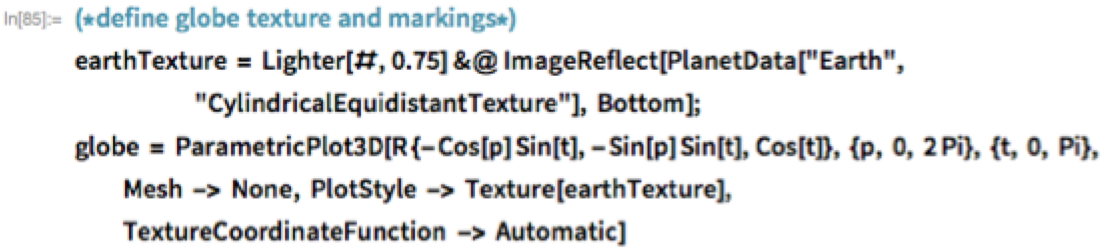


```
In[78]:= Fma = {{x''[t], y''[t], z''[t]} == grad};
         FmaIGF = {{x''[t], y''[t], z''[t]} == gradIGF};
```

I am now ready to use the power of **NDSolve** to compute orbital trajectories. Before doing this, however, it will be nice to display the orbital path as a curve in three-dimensional space. To give these curves context, I will plot them over a texture map of the Earth's surface, projected onto a sphere. Here I construct the desired graphics objects:

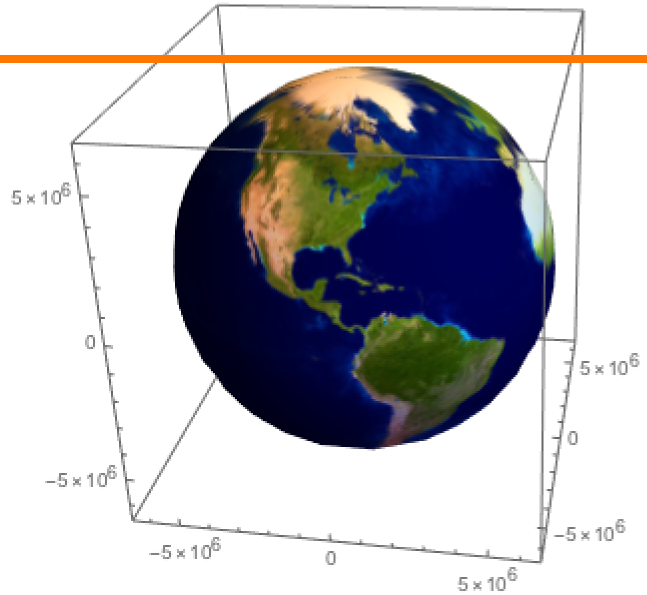


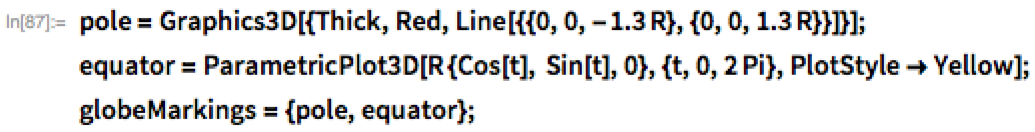
```
In[80]:= (*define orbit burnoutDot and burnoutArrow graphics*)
         burnoutCoords = {x0, y0, z0};
         burnoutDirection = {vx0, vy0, vz0} / v0;
         burnoutDot = {Red, Sphere[burnoutCoords, R / 30]};
         burnoutArrow = {Red, Arrow[{burnoutCoords,
                  burnoutCoords + (R / 3) burnoutDirection}]};
         burnoutLabels[ψ_, γ_] = Graphics3D[{burnoutDot, burnoutArrow}];
```



```
In[85]:= (*define globe texture and markings*)
         earthTexture = Lighter[#, 0.75] &@ ImageReflect[PlanetData["Earth",
                 "CylindricalEquidistantTexture"], Bottom];
         globe = ParametricPlot3D[R {-Cos[p] Sin[t], -Sin[p] Sin[t], Cos[t]}, {p, 0, 2 Pi}, {t, 0, Pi},
             Mesh -> None, PlotStyle -> Texture[earthTexture],
             TextureCoordinateFunction -> Automatic]
```

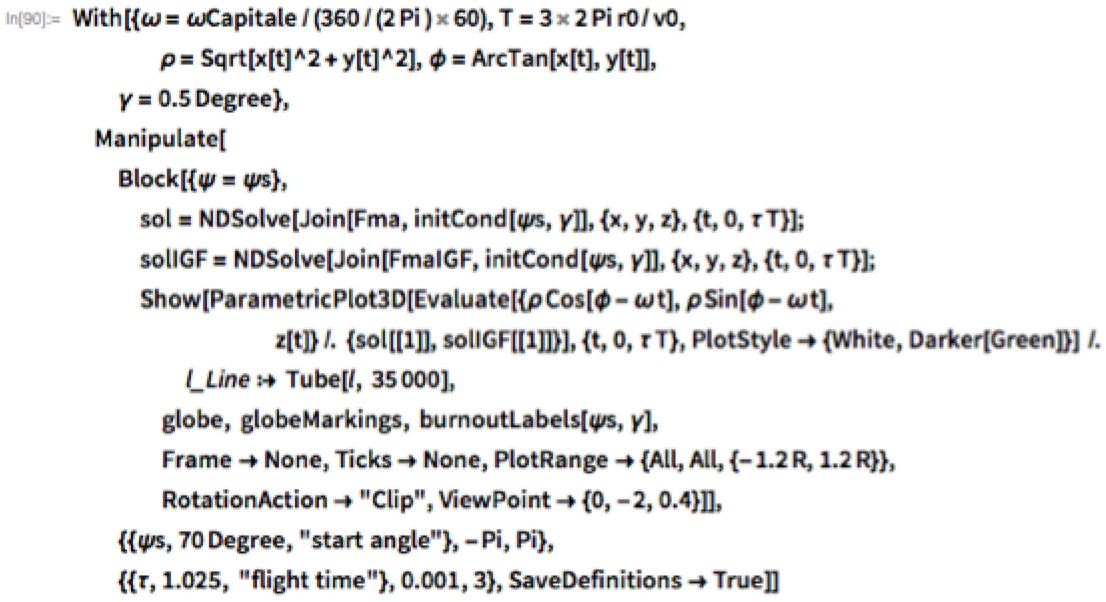
Out[86]=





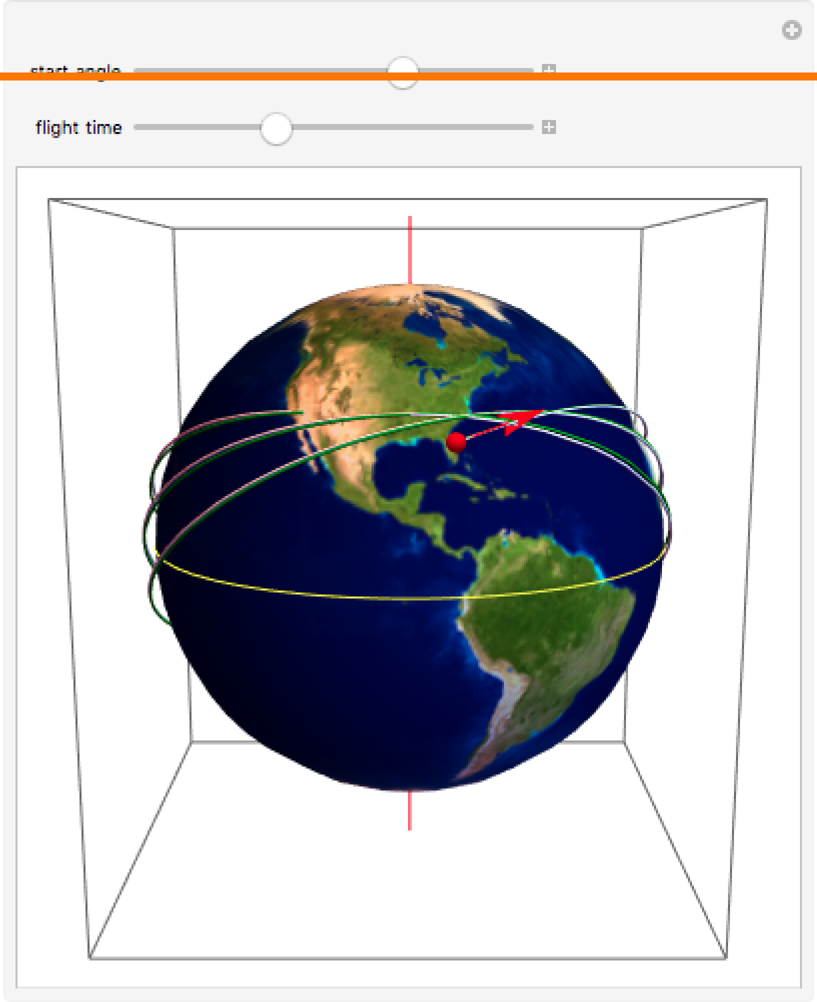
```
In[87]:= pole = Graphics3D[{Thick, Red, Line[{{0, 0, -1.3 R}, {0, 0, 1.3 R}}]}];
         equator = ParametricPlot3D[R {Cos[t], Sin[t], 0}, {t, 0, 2 Pi}, PlotStyle → Yellow];
         globeMarkings = {pole, equator};
```

While the orbital path computed in an inertial frame forms a periodic closed curve, when you account for the rotation of the Earth, it will cause the spacecraft to pass over different points on the Earth's surface during each subsequent revolution. I can visualize this effect by adding an additional rotation term to the solutions I obtain from **NDSolve**. Taking the number of orbital periods to be three (similar to John Glenn's flight) for visualization purposes, I construct the following **Manipulate** to see how the orbital path is affected by the azimuthal launch angle ψ, similar to the study in Johnson's paper. I'll plot both a path assuming a spherical Earth (in white) and another path using the IGF (in green) to get a sense of the size of the oblateness effect (note that the divergence of the two paths increases with each orbit):



```
In[90]:= With[{ω = ωCapitale / (360 / (2 Pi) × 60), T = 3 × 2 Pi r0 / v0,
             ρ = Sqrt[x[t]^2 + y[t]^2], ϕ = ArcTan[x[t], y[t]],
           γ = 0.5 Degree},
          Manipulate[
           Block[{ψ = ψs},
            sol = NDSolve[Join[Fma, initCond[ψs, γ]], {x, y, z}, {t, 0, τ T}];
            solIGF = NDSolve[Join[FmaIGF, initCond[ψs, γ]], {x, y, z}, {t, 0, τ T}];
            Show[ParametricPlot3D[Evaluate[{ρ Cos[ϕ - ω t], ρ Sin[ϕ - ω t],
                        z[t]} /. {sol[[1]], solIGF[[1]]}], {t, 0, τ T}, PlotStyle → {White, Darker[Green]}] /.
               l_Line :→ Tube[l, 35 000],
             globe, globeMarkings, burnoutLabels[ψs, γ],
             Frame → None, Ticks → None, PlotRange → {All, All, {-1.2 R, 1.2 R}},
             RotationAction → "Clip", ViewPoint → {0, -2, 0.4}]],
           {{ψs, 70 Degree, "start angle"}, -Pi, Pi},
           {{τ, 1.025, "flight time"}, 0.001, 3}, SaveDefinitions → True]]
```

Out[90]=



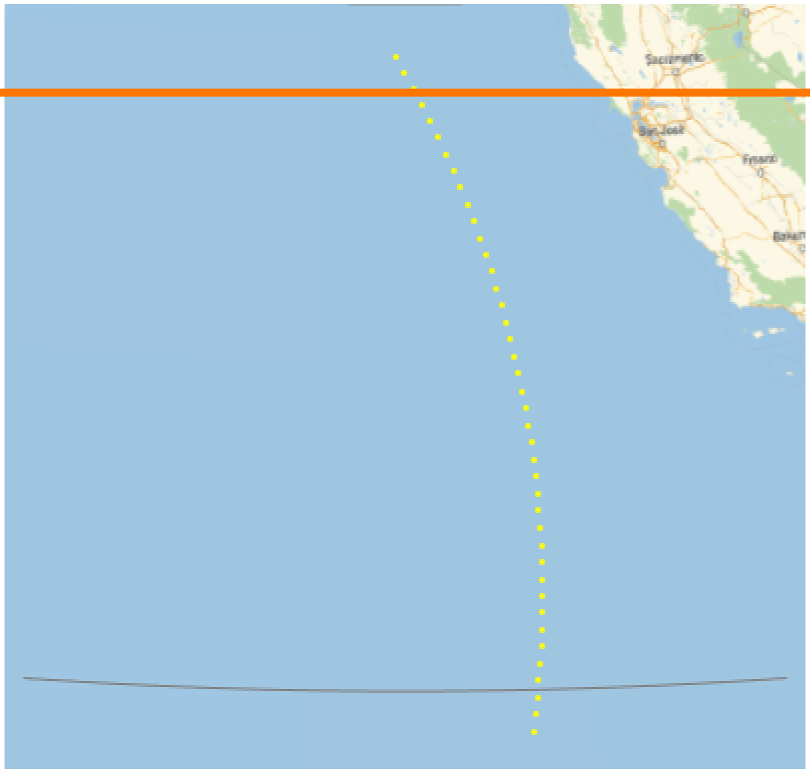
In the notebook attached to this blog, you can see this **Manipulate** in action, and note the speed at which each new solution is obtained. You would hope that Katherine Johnson and her colleagues at NASA would be impressed!

Now, varying the angle *ψ* at burnout time, it is straightforward to calculate the position of the spacecraft after, say, three revolutions:

```
In[91]:= posis = Block[{γ = 0.5 Degree, ω = ωCapitale / (360 / (2 Pi ) × 60), T = 3.05 × 2 Pi r0 / v0},
           Table[Map[(Most[#] - {0, ω T 360 / (2 Pi)}) &, GeoPosition[GeoPositionXYZ[{x[T], y[T], z[T]} /.
                   NDSolve[Join[FmalGF, initCond[ψ, γ]], {x, y, z}, {t, 0, T}][[1]]]]],
            {ψ, 50 Degree, 90 Degree, 1 Degree}]];
```

```
In[92]:= GeoGraphics[{Gray, GeoPath[{"Parallel", 28, {-135, -120}}],
           Yellow, Point[posis]}, GeoRange → Quantity[450, "Miles"],
          GeoBackground → "StreetMap", ImageSize → 400]
```

Out[92]=



Modeling the Reentry of a Satellite

The movie also mentions Euler's method in connection with the reentry phase. After the initial problem of finding the azimuthal angle has been solved, as done in the previous sections, it's time to come back to Earth. Rockets are fired to slow down the orbiting body, and a complex set of events happens as the craft transitions from the vacuum of space to an atmospheric environment. Changing atmospheric density, rapid deceleration and frictional heating all become important factors that must be taken into account in order to safely return the astronaut to Earth. Height, speed and acceleration as a function of time are all problems that need to be solved. This set of problems can be solved with Euler's method, as done by Katherine Johnson, or by using the differential equation-solving functionality in the Wolfram Language.

For simple differential equations, one can get a detailed step-by-step solution with a specified quadrature method. An equivalent of Newton's famous $F = m\,a$ for a time-dependent mass $m(t)$ is the so-called ideal rocket equation (in one dimension)…

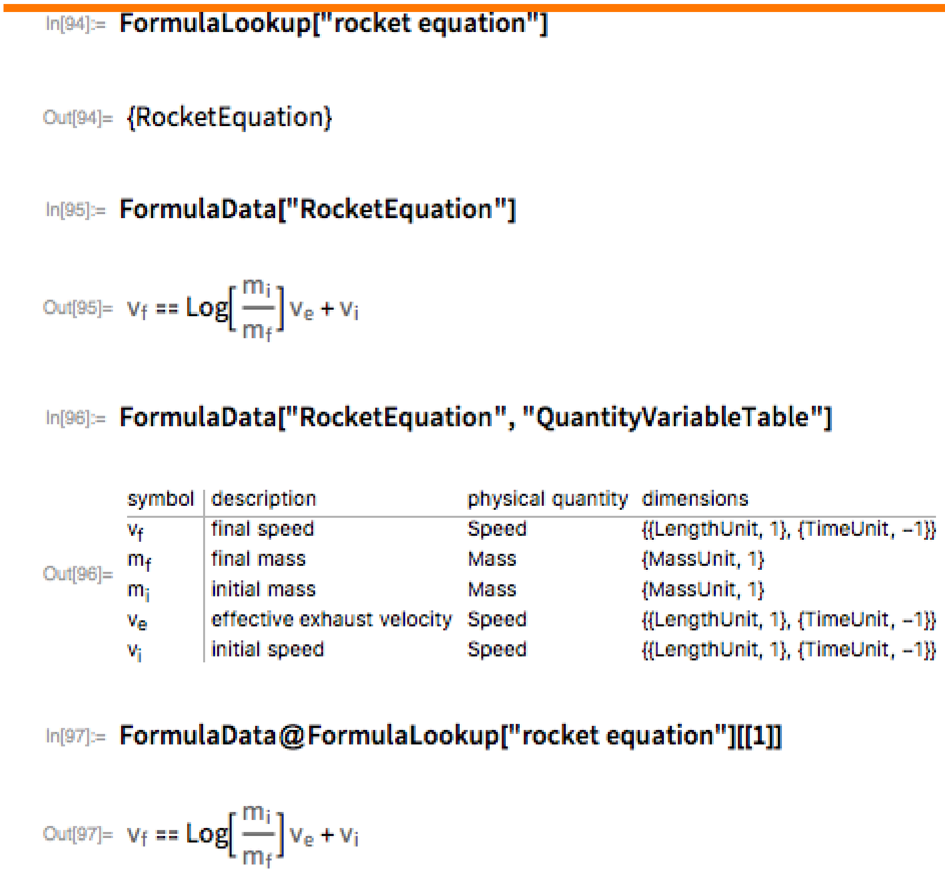
$$m(t)\,v'(t) == -v_e\,m_p'(t),$$

… where $m(t)$ is the rocket mass, v_e the engine exhaust velocity and $m'_p(t)$ the time derivative of the propellant mass. Assuming a constant $m'_p(t)$, the structure of the equation is relatively simple and easily solvable in closed form:

```
In[93]:= DSolve[{(m0 − β t) v'[t] == vₑₓ β, v[0] == vi}, v[t], t]
```

```
Out[93]= {{v[t] → vi + Log[m0] vₑₓ − Log[m0 − t β] vₑₓ}}
```

With initial and final conditions for the mass, I get the celebrated rocket equation (Tsiolkovsky 1903):



```
In[94]:= FormulaLookup["rocket equation"]

Out[94]= {RocketEquation}

In[95]:= FormulaData["RocketEquation"]
```

Out[95]= $v_f == \text{Log}\left[\frac{m_i}{m_f}\right] v_e + v_i$

```
In[96]:= FormulaData["RocketEquation", "QuantityVariableTable"]
```

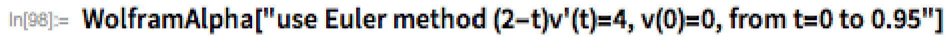
Out[96]=

symbol	description	physical quantity	dimensions
v_f	final speed	Speed	{{LengthUnit, 1}, {TimeUnit, −1}}
m_f	final mass	Mass	{MassUnit, 1}
m_i	initial mass	Mass	{MassUnit, 1}
v_e	effective exhaust velocity	Speed	{{LengthUnit, 1}, {TimeUnit, −1}}
v_i	initial speed	Speed	{{LengthUnit, 1}, {TimeUnit, −1}}

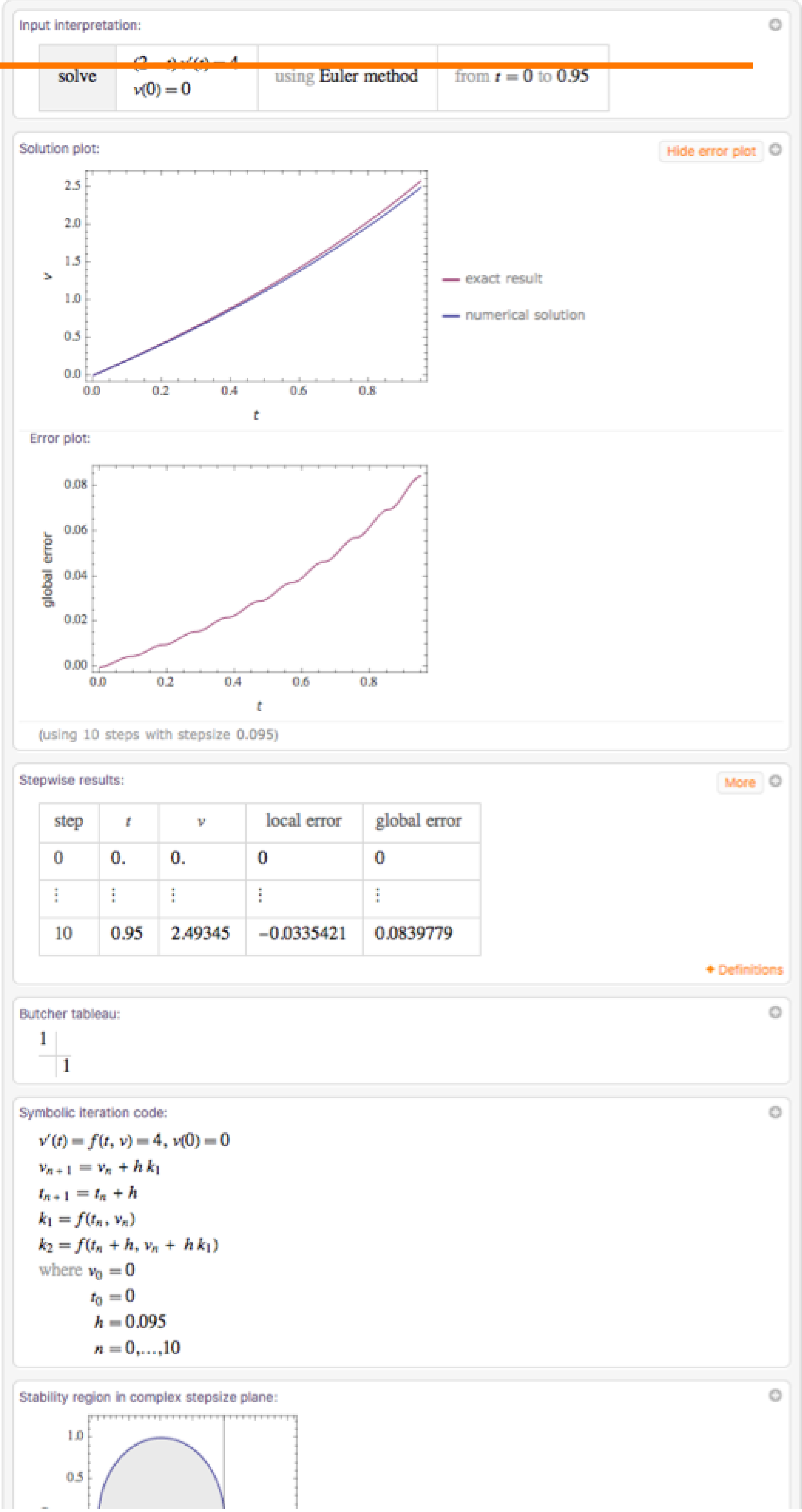
```
In[97]:= FormulaData@FormulaLookup["rocket equation"][[1]]
```

Out[97]= $v_f == \text{Log}\left[\frac{m_i}{m_f}\right] v_e + v_i$

The details of solving this equation with concrete parameter values and e.g. with the classical Euler method I can get from Wolfram|Alpha. Here are those details together with a detailed comparison with the exact solution, as well as with other numerical integration methods:



```
In[98]:= WolframAlpha["use Euler method (2–t)v'(t)=4, v(0)=0, from t=0 to 0.95"]
```



Input interpretation:

solve	$(2-t)\,v'(t)=4$ $v(0)=0$	using Euler method	from $t=0$ to 0.95

Solution plot:

Error plot:

(using 10 steps with stepsize 0.095)

Stepwise results:

step	t	v	local error	global error
0	0.	0.	0	0
⋮	⋮	⋮	⋮	⋮
10	0.95	2.49345	−0.0335421	0.0839779

Butcher tableau:

$$\begin{array}{c|c} 1 & \\ \hline & 1 \end{array}$$

Symbolic iteration code:

$v'(t)=f(t,v)=4,\ v(0)=0$

$v_{n+1}=v_n+h\,k_1$

$t_{n+1}=t_n+h$

$k_1=f(t_n,v_n)$

$k_2=f(t_n+h,v_n+\ h\,k_1)$

where $v_0=0$

$t_0=0$

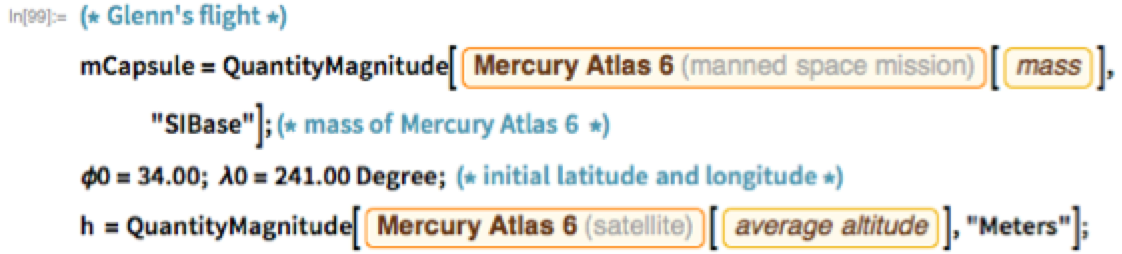
$h=0.095$

$n=0,\ldots,10$

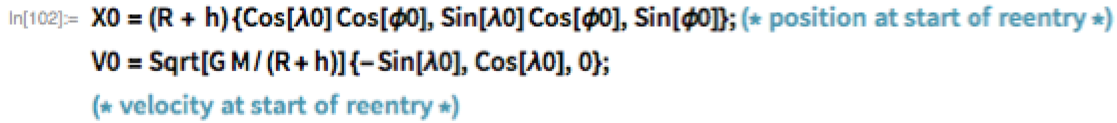
Out[98]=

Stability region in complex stepsize plane:

Following the movie plot, I will now implement a minimalistic ODE model of the reentry process. I start by defining parameters that mimic Glenn's flight:



```
In[99]:= (* Glenn's flight *)
         mCapsule = QuantityMagnitude[ Mercury Atlas 6 (manned space mission) [ mass ],
             "SIBase"]; (* mass of Mercury Atlas 6 *)
         ϕ0 = 34.00; λ0 = 241.00 Degree; (* initial latitude and longitude *)
         h = QuantityMagnitude[ Mercury Atlas 6 (satellite) [ average altitude ], "Meters"];
```



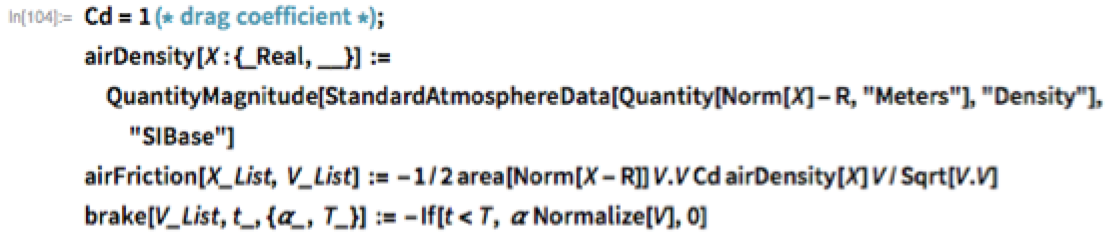
```
In[102]:= X0 = (R + h){Cos[λ0] Cos[ϕ0], Sin[λ0] Cos[ϕ0], Sin[ϕ0]}; (* position at start of reentry *)
          V0 = Sqrt[G M / (R + h)] {-Sin[λ0], Cos[λ0], 0};
          (* velocity at start of reentry *)
```

I assume that the braking process uses 1% of the thrust of the stage-one engine and runs, say, for 60 seconds. The equation of motion is:

$$m_{\text{caps}}\, \boldsymbol{v}'(t) == \boldsymbol{F}_{\text{grav}}(\boldsymbol{x}(t)) + \boldsymbol{F}_{\text{exhaust}}(t) + \boldsymbol{F}_{\text{friction}}(\boldsymbol{x}(t), \boldsymbol{v}(t))$$

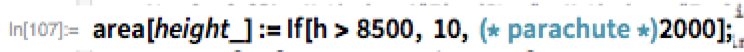
Here, $\boldsymbol{F}_{\text{grav}}$ is the gravitational force, $\boldsymbol{F}_{\text{exhaust}}(t)$ the explicitly time-dependent engine force and $\boldsymbol{F}_{\text{friction}}(\boldsymbol{x}(t),\boldsymbol{v}(t))$ the friction force. The latter depends via the air density explicitly on the position $\boldsymbol{x}(t)$ and via the friction law on $\boldsymbol{v}(t)$.

For the height-dependent air density, I can conveniently use the StandardAtmosphereData function. I also account for a height-dependent area because of the parachute that opened about 8.5 km above ground:



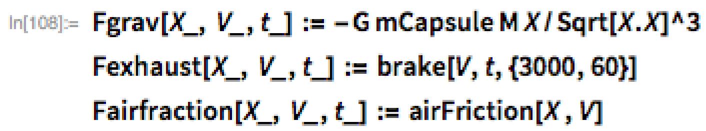
```
In[104]:= Cd = 1 (* drag coefficient *);
          airDensity[X : {_Real, __}] :=
            QuantityMagnitude[StandardAtmosphereData[Quantity[Norm[X] - R, "Meters"], "Density"],
              "SIBase"]
          airFriction[X_List, V_List] := -1/2 area[Norm[X - R]] V.V Cd airDensity[X] V/Sqrt[V.V]
          brake[V_List, t_, {α_, T_}] := -If[t < T, α Normalize[V], 0]
```

This gives the following set of coupled nonlinear differential equations to be solved. The last WhenEvent[...] specifies to end the integration when the capsule reaches the surface of the Earth. I use vector-valued position and velocity variables X and V:



```
In[107]:= area[height_] := If[h > 8500, 10, (* parachute *)2000];
```

With these definitions for the weight, exhaust and air friction force terms…



```
In[108]:= Fgrav[X_, V_, t_] := -G mCapsule M X / Sqrt[X.X]^3
          Fexhaust[X_, V_, t_] := brake[V, t, {3000, 60}]
          Fairfraction[X_, V_, t_] := airFriction[X, V]
```

… total force can be found via:

```
In[111]:= Ftotal[X_, V_, t_] := Fgrav[X, V, t] + Fexhaust[X, V, t] + Fairfraction[X, V, t]
```

```
In[112]:= odeSystem = {X'[t] == V[t],
                        mCapsule V'[t] == Ftotal[X[t], V[t], t],
                        WhenEvent[Norm[X[t]] - R == 0, "StopIntegration"]};
```

In this simple model, I neglected the Earth's rotation, intrinsic rotations of the capsule, active flight angle changes, supersonic effects on the friction force and more. The explicit form of the differential equations in coordinate components is the following. The equations that Katherine Johnson solved would have been quite similar to these:

```
In[113]:= (odeSystem /. {X → ({x[#], y[#], z[#]} &), V → ({vx[#], vy[#], vz[#]} &)} /.
                w_WhenEvent :> Evaluate //@ w) // Column[#, Dividers → Center] &
```

Out[113]=

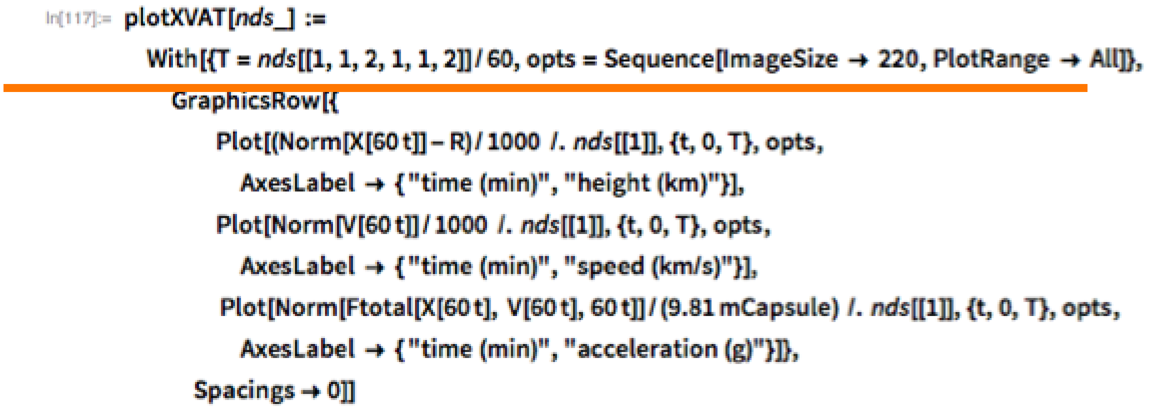
$$\begin{array}{l}
\{x'[t], y'[t], z'[t]\} == \{vx[t], vy[t], vz[t]\} \\
\hline
\{1352.0\, vx'[t], 1352.0\, vy'[t], 1352.0\, vz'[t]\} == \\
\quad \Big\{-\text{If}[t < 60, 3000\, \text{Normalize}[\{vx[t], vy[t], vz[t]\}], 0] - 5\, \text{airDensity}[\{x[t], y[t], z[t]\}]\, vx[t] \\
\qquad \sqrt{vx[t]^2 + vy[t]^2 + vz[t]^2} - \frac{5.389\times10^{17}\, x[t]}{(x[t]^2+y[t]^2+z[t]^2)^{3/2}}, -\text{If}[t < 60, 3000\, \text{Normalize}[\{vx[t], vy[t], vz[t]\}], 0] - \\
\qquad 5\, \text{airDensity}[\{x[t], y[t], z[t]\}]\, vy[t] \sqrt{vx[t]^2 + vy[t]^2 + vz[t]^2} - \frac{5.389\times10^{17}\, y[t]}{(x[t]^2+y[t]^2+z[t]^2)^{3/2}}, \\
\quad -\text{If}[t < 60, 3000\, \text{Normalize}[\{vx[t], vy[t], vz[t]\}], 0] - \\
\qquad 5\, \text{airDensity}[\{x[t], y[t], z[t]\}]\, vz[t] \sqrt{vx[t]^2 + vy[t]^2 + vz[t]^2} - \frac{5.389\times10^{17}\, z[t]}{(x[t]^2+y[t]^2+z[t]^2)^{3/2}}\Big\} \\
\hline
\text{WhenEvent}\Big[-6.3710088\times10^6 + \sqrt{\text{Abs}[x[t]]^2 + \text{Abs}[y[t]]^2 + \text{Abs}[z[t]]^2} == 0, \text{StopIntegration}\Big]
\end{array}$$

Supplemented by the initial position and velocity, it is straightforward to solve this system of equations numerically. Today, this is just a simple call to **NDSolve**. I don't have to worry about the method to use, step size control, error control and more because the Wolfram Language automatically chooses values that guarantee meaningful results:

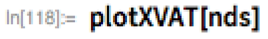
```
In[114]:= tMax = 5000;
          inits = {X[0] == X0, V[0] == V0};
          nds = NDSolve[Join[odeSystem, inits], {X, V}, {t, 0, 60, tMax}]
```

```
Out[116]= {{X → InterpolatingFunction[ ⊞  Domain: {{0., 1440.}}
                                           Output dimensions: {3} ],

            V → InterpolatingFunction[ ⊞  Domain: {{0., 1440.}}
                                           Output dimensions: {3} ]}}
```

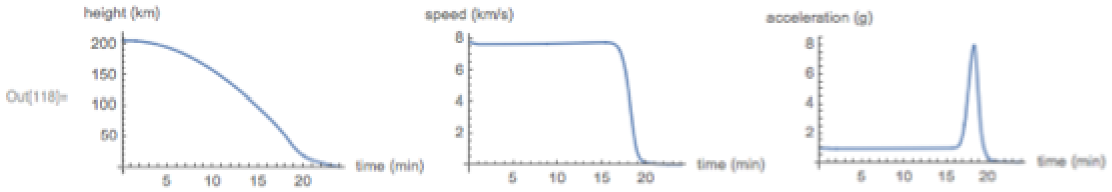
Here is a plot of the height, speed and acceleration as a function of time:



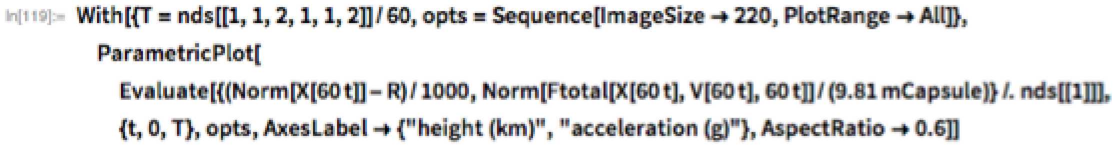
```
In[117]:= plotXVAT[nds_] :=
    With[{T = nds[[1, 1, 2, 1, 1, 2]] / 60, opts = Sequence[ImageSize → 220, PlotRange → All]},
      GraphicsRow[{
          Plot[(Norm[X[60 t]] – R) / 1000 /. nds[[1]], {t, 0, T}, opts,
            AxesLabel → {"time (min)", "height (km)"}],
          Plot[Norm[V[60 t]] / 1000 /. nds[[1]], {t, 0, T}, opts,
            AxesLabel → {"time (min)", "speed (km/s)"}],
          Plot[Norm[Ftotal[X[60 t], V[60 t], 60 t]] / (9.81 mCapsule) /. nds[[1]], {t, 0, T}, opts,
            AxesLabel → {"time (min)", "acceleration (g)"}]},
        Spacings → 0]]
```



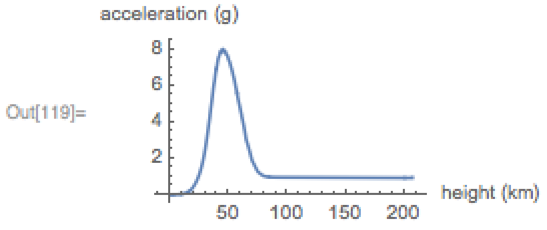
```
In[118]:= plotXVAT[nds]
```



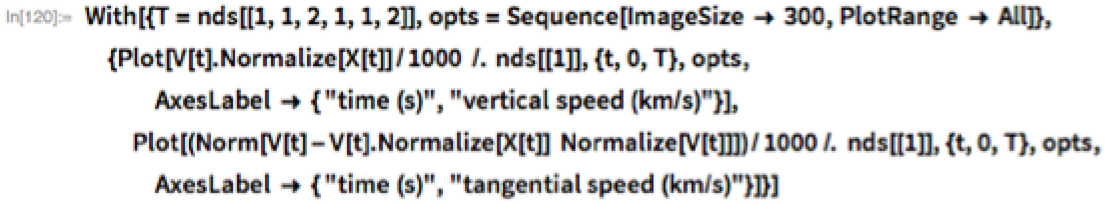
Plotting as a function of height instead of time shows that the exponential increase of air density is responsible for the high deceleration. This is not due to the parachute, which happens at a relatively low altitude. The peak deceleration happens at a very high altitude as the capsule goes from a vacuum to an atmospheric environment very quickly:



```
In[119]:= With[{T = nds[[1, 1, 2, 1, 1, 2]] / 60, opts = Sequence[ImageSize → 220, PlotRange → All]},
    ParametricPlot[
      Evaluate[{(Norm[X[60 t]] – R) / 1000, Norm[Ftotal[X[60 t], V[60 t], 60 t]] / (9.81 mCapsule)} /. nds[[1]]],
      {t, 0, T}, opts, AxesLabel → {"height (km)", "acceleration (g)"}, AspectRatio → 0.6]]
```

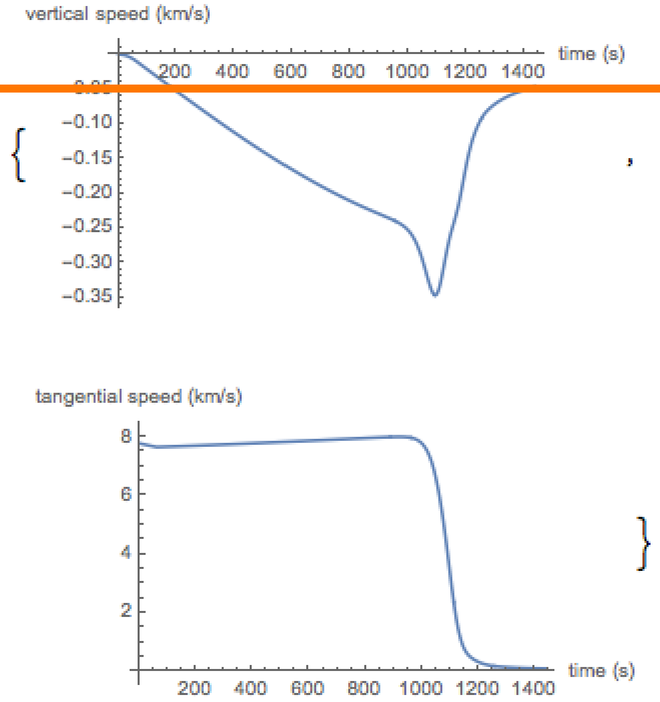


And here is a plot of the vertical and tangential speed of the capsule in the reentry process:



```
In[120]:= With[{T = nds[[1, 1, 2, 1, 1, 2]], opts = Sequence[ImageSize → 300, PlotRange → All]},
    {Plot[V[t].Normalize[X[t]] / 1000 /. nds[[1]], {t, 0, T}, opts,
        AxesLabel → {"time (s)", "vertical speed (km/s)"}],
      Plot[(Norm[V[t] – V[t].Normalize[X[t]] Normalize[V[t]]]) / 1000 /. nds[[1]], {t, 0, T}, opts,
        AxesLabel → {"time (s)", "tangential speed (km/s)"}]}]
```

Out[120]=



Now I repeat the numerical solution with a fixed-step Euler method:

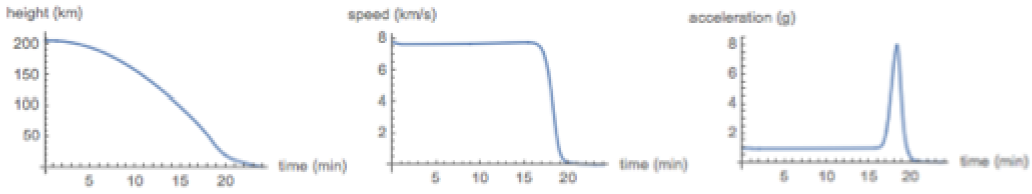
```
In[121]:= ndsEuler = NDSolve[Join[odeSystem, inits], {X, V}, {t, 0, tMax},
             Method → {"FixedStep", Method → "ExplicitEuler"}, StartingStepSize → 0.05]
```

```
Out[121]= {{X → InterpolatingFunction[ Domain: {{0., 1.44×10^3}}
                                       Output dimensions: {3} ],

             V → InterpolatingFunction[ Domain: {{0., 1.44×10^3}}
                                        Output dimensions: {3} ]}}
```

Qualitatively, the solution looks the same as the previous one:

```
In[122]:= plotXVAT[ndsEuler]
```

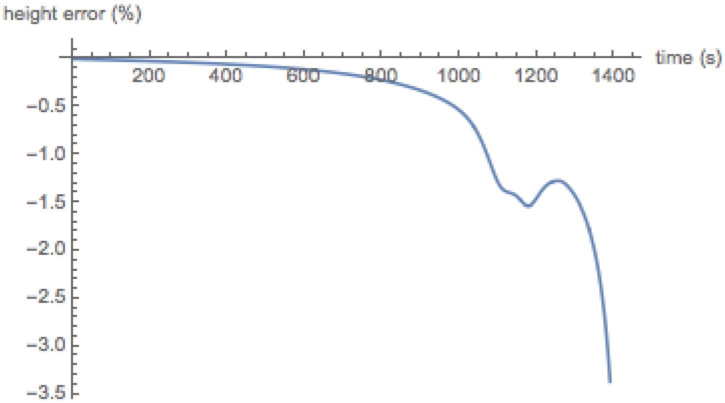
Out[122]=



For the used step size of the time integration, the accumulated error is on the order of a few percent. Smaller step sizes would reduce the error (see the previous Wolfram|Alpha output):

```
In[123]:= With[{T = nds[[1, 1, 2, 1, 1, 2]]},
            Plot[100 ((Norm[X[t]] – R /. nds[[1]])/
                          (Norm[X[t]] – R /. ndsEuler[[1]]) – 1), {t, 0, T},
                   AxesLabel → {"time (s)", "height error (%)"}]]
```

Out[123]=



Note that the landing time predicted by the Euler method deviates only 0.11% from the previous time. (For comparison, if I were to solve the equation with two modern methods, say **"BDF"** vs. **"Adams"**, the error would be smaller by a few orders of magnitude.)

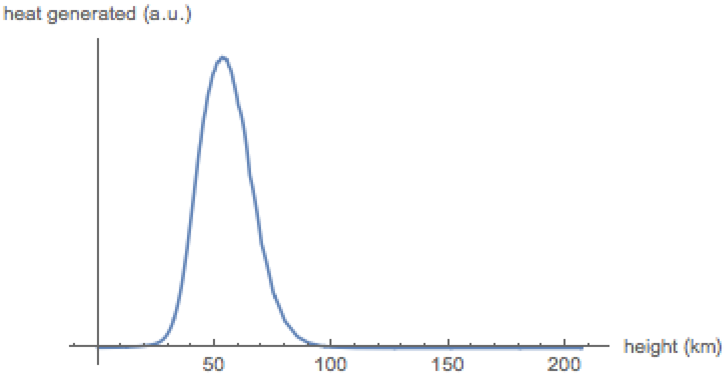
Now, the reentry process generates a lot of heat. This is where the heat shield is needed. At which height is the most heat per area *q* generated? Without a detailed derivation, I can, from purely dimensional grounds, conjecture $\dot{q} \sim \rho v^3$:

```
In[124]:= DimensionalCombinations[{"Speed", "MassDensity"}, "HeatFlux"]
```

```
Out[124]= {MassDensity Speed³}
```

```
In[125]:= With[{T = nds[[1, 1, 2, 1, 1, 2]]},
           ParametricPlot[Evaluate[{(Norm[X[t]] – R)/1000, airDensity[X[t]] Norm[V[t]]^3} /. nds[[1]]],
             {t, 0, T}, PlotRange → All, Ticks → {True, False},
                AxesLabel → {"height (km)", "heat generated (a.u.)"}, AspectRatio → 0.6]]
```

Out[125]=



Many more interesting things could be calculated (Hicks 2009), but just like the movie had to fit everything into two hours and seven minutes, I will now end my blog for the sake of time. I hope I can be pardoned for the statement that, with the Wolfram Language, the sky's the limit.

To download this post as a Computable Document Format (CDF) file, click here. New to CDF? Get your copy for free with this one-time download.

182 1590 2805

POSTED IN: Best of Blog | History | Mathematics

Jeff Bryant, Research Programmer, Wolfram|Alpha Scientific Content

Paco Jain, Research Programmer, Wolfram|Alpha Scientific Content

Michael Trott, Chief Scientist, Wolfram|Alpha Scientific Content

COMMENTS

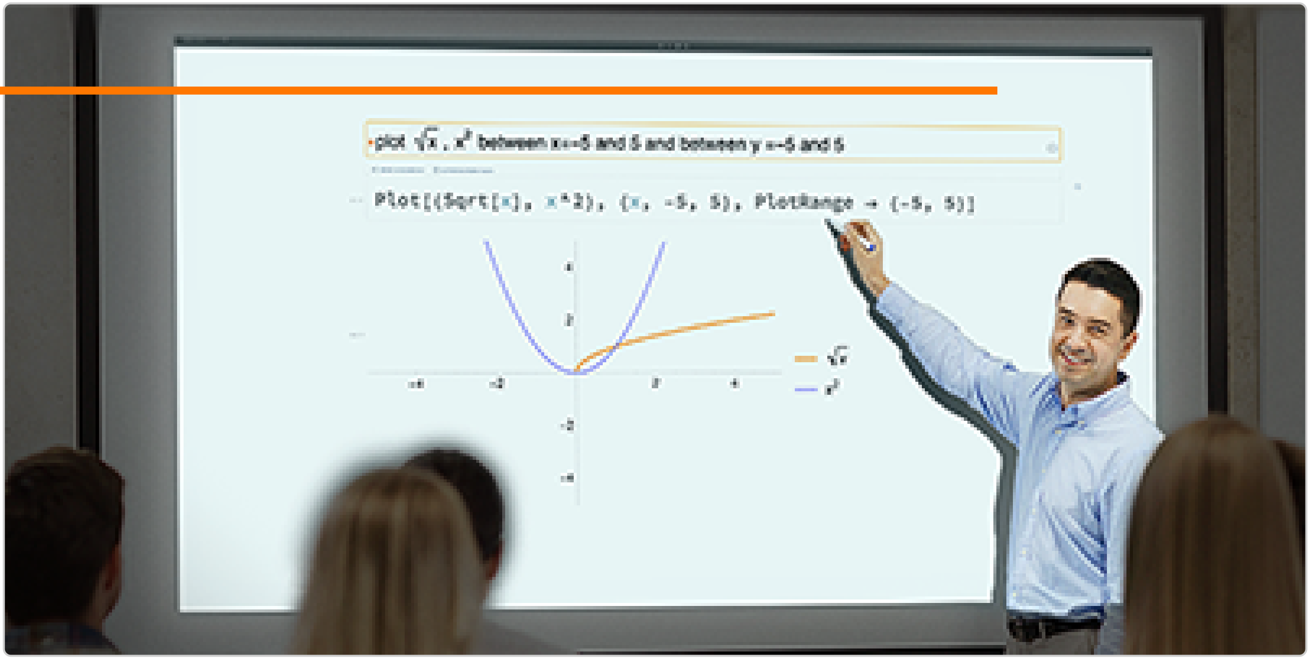
Join the discussion

+ **14 comments**

RELATED POSTS

BEST OF BLOG

Active Learning with Wolfram|Alpha Notebook Edition

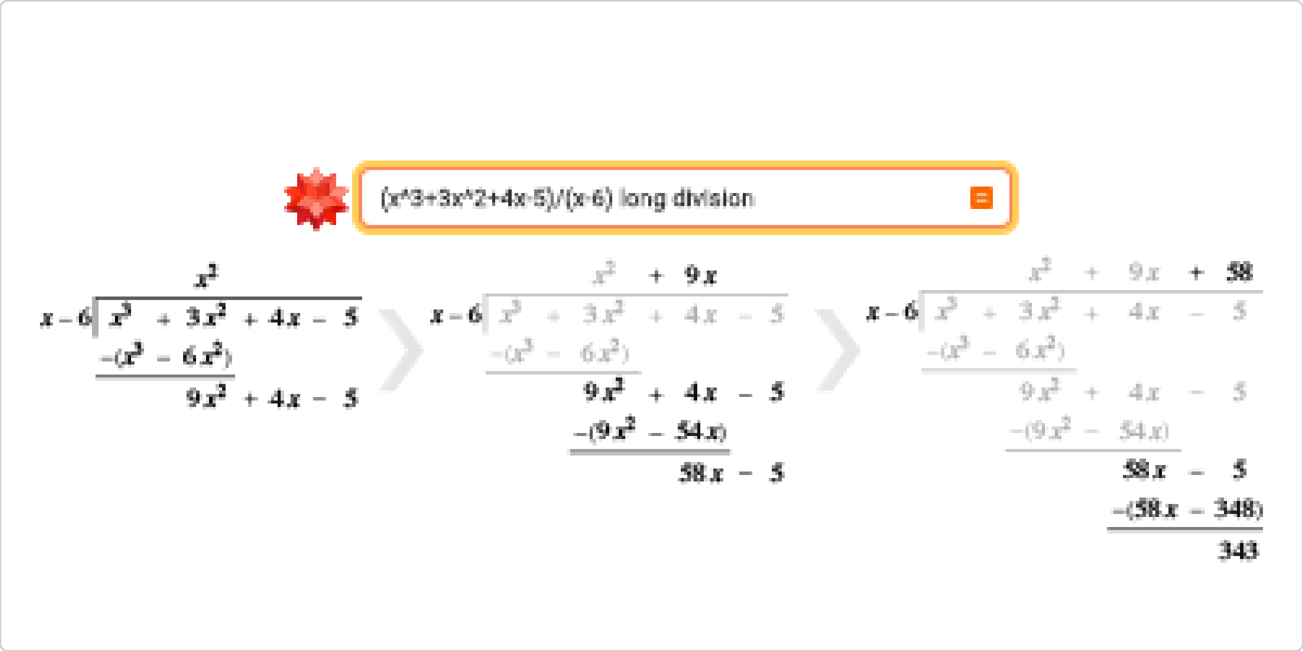


plot √x , x² between x=-5 and 5 and between y =-5 and 5

```
Plot[{Sqrt[x], x^2}, {x, -5, 5}, PlotRange → {-5, 5}]
```

BEST OF BLOG

Wolfram|Alpha Pro Teaches Step-by-Step Arithmetic for All Grade Levels



(x^3+3x^2+4x-5)/(x-6) long division

$$x-6\,\overline{)\,x^3+3x^2+4x-5}\qquad \text{quotient } x^2$$

$$-(x^3-6x^2)$$

$$9x^2+4x-5$$

$$x^2+9x$$

$$-(9x^2-54x)$$

$$58x-5$$

$$x^2+9x+58$$

$$-(58x-348)$$

$$343$$

BEST OF BLOG

Fractional Calculus in Wolfram Language 13.1

$f(x) = \frac{x^2}{2}$
$\frac{d^\alpha}{dx^\alpha} f(x) = \frac{x^{2-\alpha}}{\Gamma(3-\alpha)}$
$\alpha = 1$
$\alpha = 0.5$
$\alpha = 0$
$\alpha = -0.5$
$\alpha = -1$