Math 4510 Midterm Project

Numerical Differential Equations Project

Fred Hohman
March 18, 2014

Prompt: A magnetic rail-gun is used to launch a steel cylinder 25 cm in diameter and weighing 2 tons from a naval installation in Pearl Harbor, HI at a hostile carrier group 250 miles due west.
1. Develop a differential equation describing the flight of the projectile.
2. Implement a shooting method to solve your ODE in *Mathematica* to solve for a launch speed which will cause the cylinder to land on target assuming that you are given a launch angle of θ.
3. Determine by experiment the permissible window of angles and speeds which will result in a hit.
4. What is the kinetic energy of the projectile on impact? How does this compare to the chemical energy of an equivalent weight of TNT?

Notes:
1. Include air resistance and the varying gravitational field of the Earth in the model.
2. Assume the hostile carrier has the dimensions of the U.S.S. Nimitz.

Thanks: Eric Lybrand, Mallory Burks, Irma Stevens.

Notebook Evaluation Time: ~30 seconds.

---

0. Set Up.

Before we start, we need to make some assumptions. Most importantly, let's assume the earth is a perfect sphere of radius 6,378.1 km. Let's also assume that all physical scenarios will use equations and relationships from classical mechanics, i.e., assume our projectile is *not* moving at relativistic speeds. We will be making more assumptions along the way.

For some early notation we will use the abbreviations of PH to represent Pearl Harbor and TG to represent our target.

In[1]:=
```
EarthRadius = 6378.1 * 10^3;
```

Latitude and Longitude.

To begin, let's first determine our stationary location and our target location. To do this, I used Google Maps to obtain the approximate latitude and longitude of Pearl Harbor, Hawaii: {21.3679°N, 157.9771°W}

In[2]:=
```
PHLatLong = {21.3679, 157.9771};
```

Now we are told our target is 250 miles due west, i.e., coordinates 21.3679°N (the same as Pearl Harbor, since we aren't moving the in north or south direction), and 157.9771°W + "some unknown

angle". Since both points are located on a **small circle** of the earth, to find the latitude and longitude of the target we can consider our 250 miles of arc length in meters and find the angle that displaces our two points.

Note: every calculation used throughout this notebook will be in **SI units**.

So we need to convert 250 miles to meters.

In[3]:=
```
ArcLength = N[QuantityMagnitude[UnitConvert[Quantity[250, "Miles"], "Meters"]]]
```

Out[3]=
```
402 337.
```

Using the Arc Length Formula, ArcLength = θ*r, we can now calculate θ between the two points.

In[4]:=
```
Flatten[Solve[ArcLength == BoatAngle * EarthRadius * (Pi / 180)], 2]
```

Out[4]=
```
{BoatAngle → 3.61427}
```

In[5]:=
```
BoatAngle = BoatAngle /. {BoatAngle → 3.61427}
```

Out[5]=
```
3.61427
```

Now we can express our enemy's position in latitude and longitude, as given below.

In[6]:=
```
TGLatLong = {21.3679, 157.9771 + BoatAngle}
```

Out[6]=
```
{21.3679, 161.591}
```

Spherical Coordinates.

Recalling our assumption that the earth is a perfect sphere, we can now write these latitude and longitude values in spherical coordinates. To construct these points, recall that spherical coordinates are in the form {r, θ, ϕ}, where r = EarthRadius, θ = 180° - Longitude, and ϕ = 90° - Latitude. We will add the Degree function in our construction to convert these values to radians—we will need radians for future calculations.

In[7]:=
```
PHSpherical = {EarthRadius, (180 - 157.9771) Degree, (90 - 21.3679) Degree}
TGSpherical = {EarthRadius, (180 - (157.9771 + BoatAngle)) Degree, (90 - 21.3679) Degree}
```

Out[7]=
```
{6.3781 × 10^6, 0.384372, 1.19786}
```

Out[8]=
```
{6.3781 × 10^6, 0.321291, 1.19786}
```

Cartesian Coordinates.

Now that we have two spherical coordinate points, we can perform another coordinate transformation to go from spherical to Cartesian. Recall the construction of Cartesian coordinates form spherical coordinates below.
x = r*cos(θ)*sin(ϕ)
y = r*sin(θ)*sin(ϕ)
z = r*cos(ϕ)
So, following these formulas, we can construct the two Cartesian coordinate points below.

In[9]:=
```
PHCartesian = EarthRadius * {Cos[PHSpherical[[2]]] * Sin[PHSpherical[[3]]],
    Sin[PHSpherical[[2]]] * Sin[PHSpherical[[3]]], Cos[PHSpherical[[3]]]}
TGCartesian = EarthRadius * {Cos[TGSpherical[[2]]] * Sin[TGSpherical[[3]]],
    Sin[TGSpherical[[2]]] * Sin[TGSpherical[[3]]], Cos[TGSpherical[[3]]]}
```

Out[9]= $\{5.50628 \times 10^6, 2.22724 \times 10^6, 2.32389 \times 10^6\}$

Out[10]= $\{5.63573 \times 10^6, 1.8757 \times 10^6, 2.32389 \times 10^6\}$

Since these points lie on the surface of the earth, we can check that our coordinate transformations are correct by comparing the magnitudes of our Cartesian vectors with EarthRadius and verifying that they are equal.

In[11]:=
```
If[
 Norm[PHCartesian] == Norm[TGCartesian] == EarthRadius,
 Print["Math works!"],
 Print["Check your calculations."]]
```

```
Math works!
```

Great Circle.

Now that we have our two points defined in the standard Cartesian coordinate system, we need to pick a direction to shoot in. In particular, we want to shoot along the shortest path. We could shoot along the small circle (single meridian) that our two points are located on, but by considering a **great circle** of the earth we can minimize our projectile's path to shorten the time it takes to hit the target. Given any two non-trivial points, we can construct a great circle that splits the earth into two symmetric hemispheres by considering the intersection of a plane which passes though the center point of the sphere, Pearl Harbor, and our target position. We can choose our sphere's center to be the point (0, 0, 0) to simplify calculations. By constructing this great circle, we are finding the geodesic of the sphere, i.e., the shortest path between two points that lie on a sphere.

To do this, we can use the Haversine formula (http://en.wikipedia.org/wiki/Haversine_formula). This formula calculates great circle distances between two points on a sphere given latitude and longitude values. Borrowing from numerically stable java-script code from http://andrew.hedges.name/experiments/haversine/, I wrote a module that will calculate the great circle distance between our two points.

Recall that this distance is the minimum distance between two points on a sphere, so our result should be smaller than 250 miles.

In[12]:=
```
GreatCircleArcLength[LatLongPair1_, LatLongPair2_] := Module[{ΔLat, ΔLong, a, c},
  ΔLat = LatLongPair1[[1]] - LatLongPair2[[1]];
  ΔLong = LatLongPair1[[2]] - LatLongPair2[[2]];

  a = (Sin[ΔLat / 2 Degree]^2 + Cos[LatLongPair1[[1]] Degree] *
       Cos[LatLongPair2[[1]] Degree] * Sin[ΔLong / 2 Degree]^2);
  c = 2 ArcTan[Sqrt[a] / Sqrt[1 - a]];

  EarthRadius * c
 ]
```

So now let's find our specific great circle distance.

In[13]:=
```
GreatCircleArcLength[PHLatLong, TGLatLong]
```

Out[13]=
```
374 672.
```

Now let's recall our original ArcLength and compare distances to verify that our GreatCircleArcLength is smaller than our original ArcLength displacement.

In[14]:=
```
If[
 GreatCircleArcLength[PHLatLong, TGLatLong] < ArcLength,
 Print["Math works again!"],
 Print["Check your calculations."]]
```

```
Math works again!
```

To make things *even* clearer, let's convert this new distance into miles.

In[15]:=
```
N[QuantityMagnitude[UnitConvert[
   Quantity[GreatCircleArcLength[PHLatLong, TGLatLong], "Meters"], "Miles"]]]
```

Out[15]=
```
232.81
```

Just as we predicted, our great circle distance is slightly smaller than our original arc length separation (which is 250 miles).

Great Circle Coordinate System and Visualization.

It's time to visualize these numbers.
To begin, let's first define an entirely new coordinate system that we will place on the great circle. This new coordinate system will be 2-dimensional, and it's intersection with the Earth will be a circle centered at the origin with radius = EarthRadius, since great circles have the same radius as the sphere they are derived from.

So let's plot Pearl Harbor and our Target on this new coordinate system. We now have some freedom in placing Pearl Harbor, so let's define it's coordinate as

In[16]:=
```
PHGreatCircle = {EarthRadius, 0};
```

Now we need to find our target's position in our new coordinate system. To do this, we need to find the central angle between our two Cartesian points measured from the origin. Notice that this angle lies in the plane of the great circle, and hence our new coordinate system.

To find the central angle, recall a particular dot product property:
$\alpha \cdot \beta$ = |α|*|β|*cos(θ), where α and β are two Euclidean vectors and θ is the angle between them (our central angle).
So let's have *Mathematica* calculate the angle.

In[17]:=
```
CentralAngle =
 ArcCos[PHCartesian.TGCartesian / (Norm[PHCartesian] * Norm[TGCartesian])]
```

Out[17]=
```
0.0587435
```

For a circle centered at the origin, we know a point on that circle has coordinates {r*cos(θ), r*sin(θ)}, so let's now find our target's position.

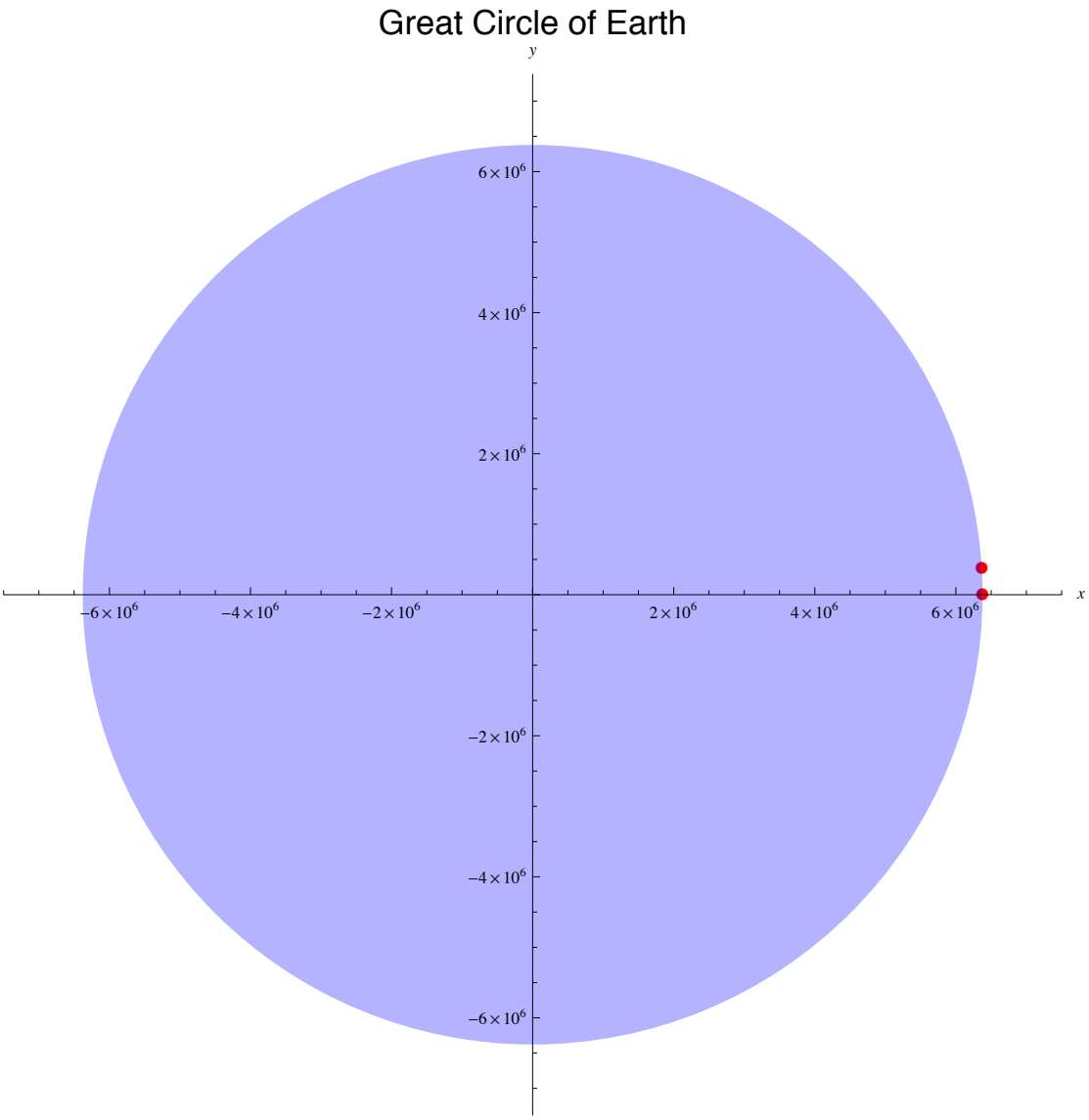
In[18]:=
```
TGGreatCircle = EarthRadius {Cos[CentralAngle], Sin[CentralAngle]}
```

Out[18]=
```
{6.3671 × 10^6, 374 457.}
```

Now we can plot our two points on our new coordinate system to see exactly where they are in relation to each other and the rest of the world. The transparent blue circle represents the great circle of the earth, and the two points represent Pearl Harbor and our target location.

In[19]:=
```
Show[{
  ListPlot[{PHGreatCircle, TGGreatCircle}, PlotMarkers → {Automatic, 10},
   AxesOrigin → {0, 0}, AxesLabel → {x, y}, PlotStyle → Red],
  Graphics[{Blue, Opacity[0.3], Disk[{0, 0}, EarthRadius]}]},
 PlotRange → {{-EarthRadius - 10^6, EarthRadius + 10^6},
   {-EarthRadius - 10^6, EarthRadius + 10^6}},
 AspectRatio → Automatic, ImageSize → Large,
 PlotLabel → Style["Great Circle of Earth", FontSize → 18, FontFamily → "Helvetica"]]
```

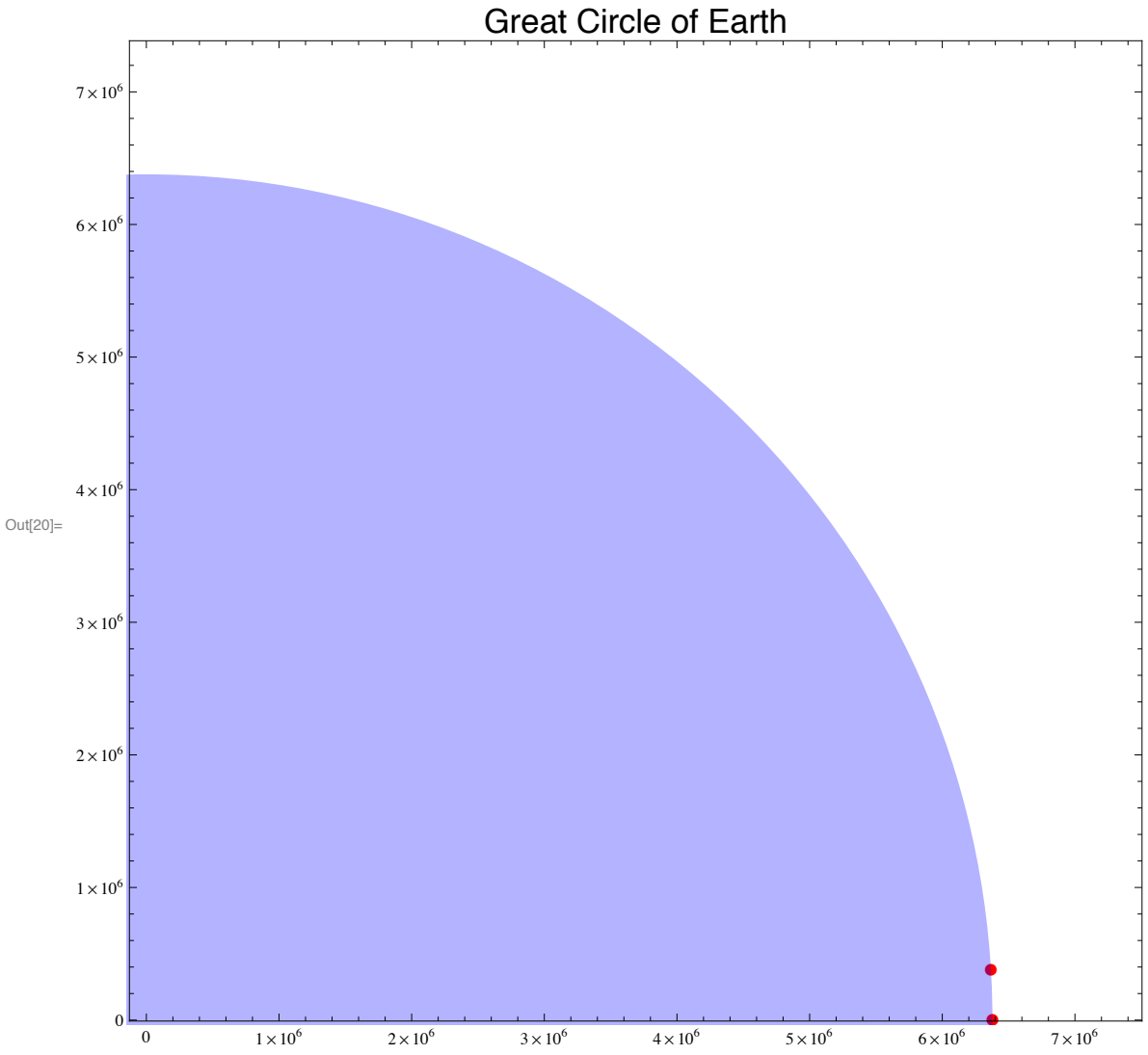
Out[19]=



Let's zoom in on the first quadrant to get a better look.

In[20]:=
```
Show[{
  ListPlot[{PHGreatCircle, TGGreatCircle}, PlotMarkers → {Automatic, 10},
   AxesOrigin → {0, 0}, AxesLabel → {x, y}, PlotStyle → Red],
  Graphics[{Blue, Opacity[0.3], Disk[{0, 0}, EarthRadius]}]},
 PlotRange → {{0, EarthRadius + 10^6}, {0, EarthRadius + 10^6}},
 AspectRatio → Automatic, ImageSize → Large,
 PlotLabel → Style["Great Circle of Earth", FontSize → 18, FontFamily → "Helvetica"],
 Frame → True]
```

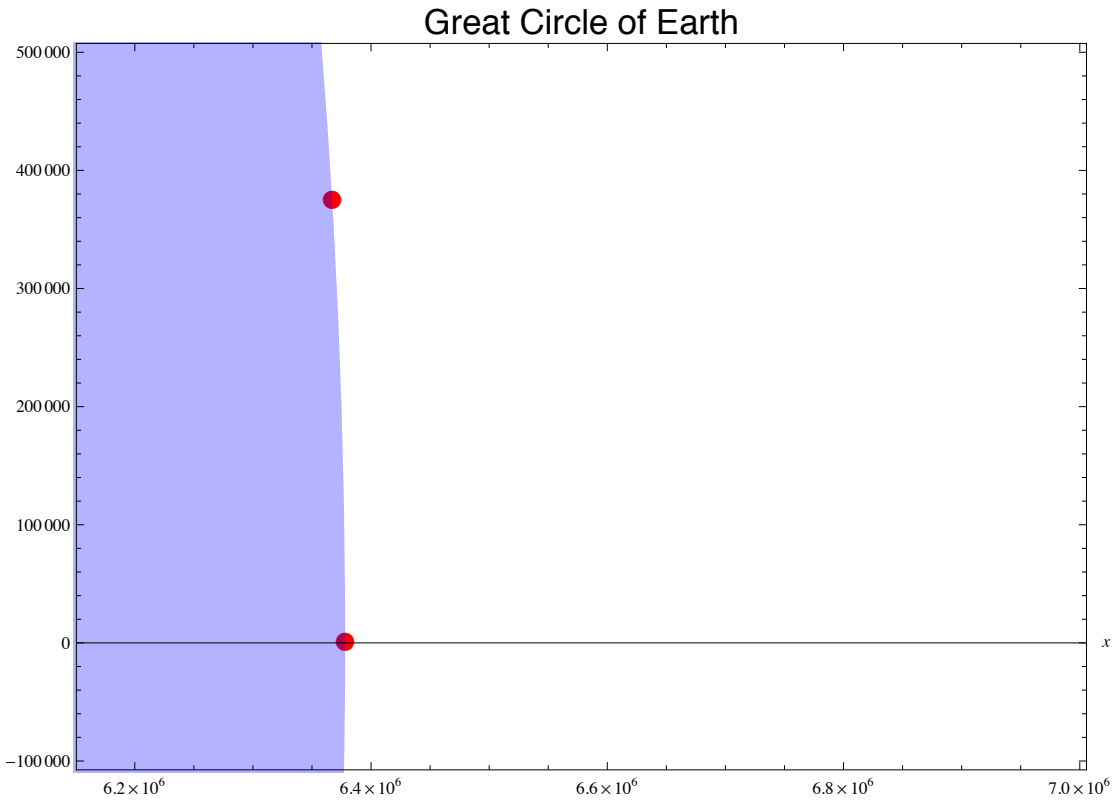
Out[20]=



Finally, let's zoom in even further by considering just a portion of the first quadrant. These close-up frame dimensions will be useful when we plot our projectile's path.

In[21]:=
```
Show[{
  ListPlot[{PHGreatCircle, TGGreatCircle}, PlotMarkers → {Automatic, 15},
   AxesOrigin → {0, 0}, AxesLabel → {x, y}, PlotStyle → Red],
  Graphics[{Blue, Opacity[0.3], Disk[{0, 0}, EarthRadius]}]},
 PlotRange → {{EarthRadius - 10^5, EarthRadius + 5 * 10^5}, {-10^5, 5 * 10^5}},
 AspectRatio → Automatic, ImageSize → Large,
 PlotLabel → Style["Great Circle of Earth", FontSize → 18, FontFamily → "Helvetica"],
 Frame → True]
```

Out[21]=



1. Defining the Differential Equation.

It's time we create our differential equation to model the flight of our projectile being launched. Consider Newton's second Law of Motion: **F** = m*__a__, where **F** and **a** are vectors. In this project we will be including two major forces, namely gravity and air drag, as most others are negligible when compared to these dominant forces.
So we can write

F / m = **a**

⇒ **a** = d**v** / dt = (**F**_gravity + **F**_air) / m

⇒ d**v** / dt = {x''(t), y''(t)}.

Using the classical model of gravity and the air drag model from the given paper "High-altitude free fall," we are able to construct the components of acceleration as

In[22]:=
```
xDoublePrime[x_, y_, xPrime_, yPrime_] := If[
  xPrime > 0,
  (1 / m) * (((G * EarthMass * m) / Norm[{x, y}]^2) * (-x / Norm[{x, y}]) -
     xPrime^2 * k0 * Exp[-(Norm[{x, y}] - EarthRadius) / λ]),
  (1 / m) * (((G * EarthMass * m) / Norm[{x, y}]^2) * (-x / Norm[{x, y}]) +
     xPrime^2 * k0 * Exp[-(Norm[{x, y}] - EarthRadius) / λ])
 ]

yDoublePrime[x_, y_, xPrime_, yPrime_] :=
 (1 / m) * (((G * EarthMass * m) / Norm[{x, y}]^2) * (-y / Norm[{x, y}]) -
    yPrime^2 * k0 * Exp[-(Norm[{x, y}] - EarthRadius) / λ])
```

Notice that xDoublePrime is a piecewise function—this accounts for the sign change in our air drag model, since at some point during the flight the velocity in the x-direction will change from positive to negative.

With these components defined, we can now write down our final system of differential equations X[t, **x**] below, where **x** is the vector {Position, Velocity} = {x, y, xPrime, yPrime}.

In[24]:=
```
X[t_, {x_, y_, xPrime_, yPrime_}] := {xPrime, yPrime,
  xDoublePrime[x, y, xPrime, yPrime], yDoublePrime[x, y, xPrime, yPrime]}
```

Symbolically, this function X[t, **x**] now has the (rather complicated and "useless") form

In[25]:=
```
X[t, {x, y, xPrime, yPrime}]
```

Out[25]=
$$\left\{\text{xPrime},\ \text{yPrime},\ \text{If}\left[\text{xPrime} > 0,\ \frac{1}{\text{m}}\left(\frac{(\text{G EarthMass m})\ (-\text{x})}{\text{Norm}[\{\text{x, y}\}]^2\ \text{Norm}[\{\text{x, y}\}]} - \text{xPrime}^2\ \text{k0}\ \text{Exp}\left[-\frac{\text{Norm}[\{\text{x, y}\}] - \text{EarthRadius}}{\lambda}\right]\right),\right.$$
$$\left.\frac{1}{\text{m}}\left(\frac{(\text{G EarthMass m})\ (-\text{x})}{\text{Norm}[\{\text{x, y}\}]^2\ \text{Norm}[\{\text{x, y}\}]} + \text{xPrime}^2\ \text{k0}\ \text{Exp}\left[-\frac{\text{Norm}[\{\text{x, y}\}] - \text{EarthRadius}}{\lambda}\right]\right)\right],$$
$$\left.\frac{-e^{\frac{6.3781\times10^6 - \sqrt{\text{Abs[x]}^2 + \text{Abs[y]}^2}}{\lambda}}\ \text{k0}\ \text{yPrime}^2 - \frac{\text{EarthMass G m y}}{\left(\text{Abs[x]}^2 + \text{Abs[y]}^2\right)^{3/2}}}{\text{m}}\right\}$$

2. Shooting Method.

With our equation created, it's time to define some constants. Remember that all constants are defined with appropriate SI units.

In[26]:=
```
EarthMass = 5.97219 × 10^24; (*Mass of the Earth*)
G = 6.67384 × 10^(-11); (*Gravitational constant*)
m = 1814.37; (*Mass of our cylindrical projectile*)
k0 = 0.21; (*Coefficient of air resistance at sea level of our cylinder*)
λ = 7.4621 * 10^3; (*Characteristic hieght of the atmosphere*)
```

Now let's define our solver. We will be using the standard Runge-Kutta 4 method that we coded earlier in the semester. Since our differential equation is a system of linear ODEs, RK4 is able to handle the calculations well. Notice that our input into RK4 is a function f, an initial condition x0, a starting time t0, an ending time t1, and the number of intervals n.

In[31]:=
```
RK4[f_, x0_, t0_, t1_, n_] := Module[{α, β, γ, δ, ϵ, ϕ, w1, w2, w3, w4, h, RK4Step},

   h = (t1 - t0) / (n);
   {α, β, γ, δ, ϵ, ϕ} = {1 / 2, 1 / 2, 1 / 2, 1 / 2, 1, 1};
   {w1, w2, w3, w4} = {1 / 6, 1 / 3, 1 / 3, 1 / 6};

   RK4Step[{t_, x_, h_}] := Module[{k1, k2, k3, k4},
     k1 = h * f[t, x];
     k2 = h * f[t + α * h, x + β * k1];
     k3 = h * f[t + γ * h, x + δ * k2];
     k4 = h * f[t + ϵ * h, x + ϕ * k3];
     {t + h, x + w1 * k1 + w2 * k2 + w3 * k3 + w4 * k4, h}
    ];
    NestList[RK4Step, { t0, x0, h}, n][[All, 1 ;; 2]]
  ];
```

3. Results and Discussion Questions.

It's time to run our solver. Let's pass RK4 our system of differential equations X with the initial condition {x-position, y-position, x-velocity, and y-velocity} = {EarthRadius, 0, v_x, v_y} (where we will experiment with and vary v_x and v_y), a starting time of 0 seconds, an ending time of 325 seconds (this will be slightly longer than the flight time, since our path will enter into the Earth), and a sufficiently high amount of intervals to accurately solve the system.

From physics we know that on a flat surface with gravity as our only force, a launch of angle 45° will achieve the farthest distance. Knowing that the ocean we are shooting over isn't *exactly* flat and we now have air resistance, let's still set v_x = v_y as a guess so that we are launching at 45° from Pearl Harbor. After experimenting with various values of the initial velocity let's pick $\sqrt{2}$ * 3115 ≃ 4405 m/s (I have the foreknowledge/intuition to pick such a value after running the code multiple times!)

In[32]:=
```
WorkingSolutionListX = N[RK4[X, {EarthRadius, 0.0, 3115, 3115}, 0, 325, 10 000]]
```

Out[32]=

A very large output was generated. Here is a sample of it:

```
{{0., {6.3781 × 10^6, 0., 3115., 3115.}},
 {0.0325, {6.3782 × 10^6, 100.652, 3078.85, 3079.16}},
 {0.065, {6.3783 × 10^6, 200.159, 3043.98, 3044.6}},
 {0.0975, {6.3784 × 10^6, 298.564, 3010.32, 3011.25}}, <<9993>>,
 {324.903, {6.35769 × 10^6, 381 482., -192.785, 108.479}},
 {324.935, {6.35769 × 10^6, 381 485., -192.639, 108.313}},
 {324.968, {6.35768 × 10^6, 381 489., -192.492, 108.147}},
 {325., {6.35767 × 10^6, 381 492., -192.347, 107.981}}}
```

Show Less | Show More | Show Full Output | Set Size Limit...

Notice that the elements in our solution list are of the form {time, {x-position, y-position, v_x, v_y}}. Let's extract only the position coordinates from this list so we can plot our path on the great circle.

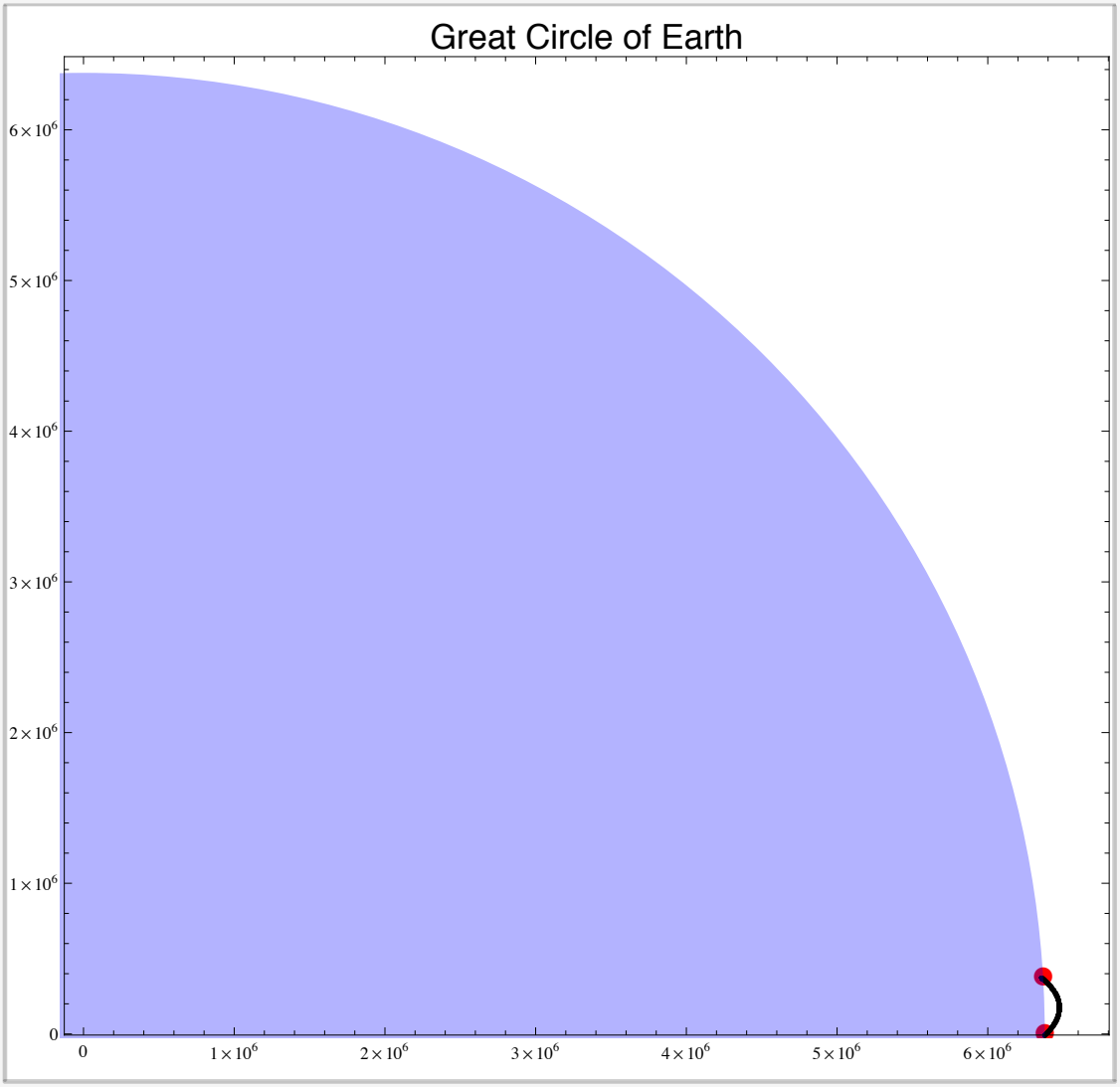
In[33]:=
```
PositionSolutionListX = WorkingSolutionListX[[All, 2, {1, 2}]];
```

Now we can plot our path on the great circle. Once again the two red points are Pearl Harbor and our target, the black curve (dense concentration of points) is our projectile's path, and the great circle of the earth is appropriately depicted as a blue disk.

In[34]:=
```
Show[{
  ListPlot[{PHGreatCircle, TGGreatCircle}, PlotMarkers → {Automatic, 15},
   AxesOrigin → {0, 0}, AxesLabel → {x, y}, PlotStyle → Red],
  ListPlot[PositionSolutionListX, PlotMarkers → {Automatic, 3},
   AxesOrigin → {0, 0}, PlotStyle → Black],
  Graphics[{Blue, Opacity[0.3], Disk[{0, 0}, EarthRadius]}]},
 PlotRange → {{0, EarthRadius + 3 * 10^5}, {0, EarthRadius + 10^5}},
 AspectRatio → Automatic, ImageSize → Large,
 PlotLabel → Style["Great Circle of Earth", FontSize → 18, FontFamily → "Helvetica"],
 Frame → True]
```

Out[34]=

A very large output was generated. Here is a sample of it:



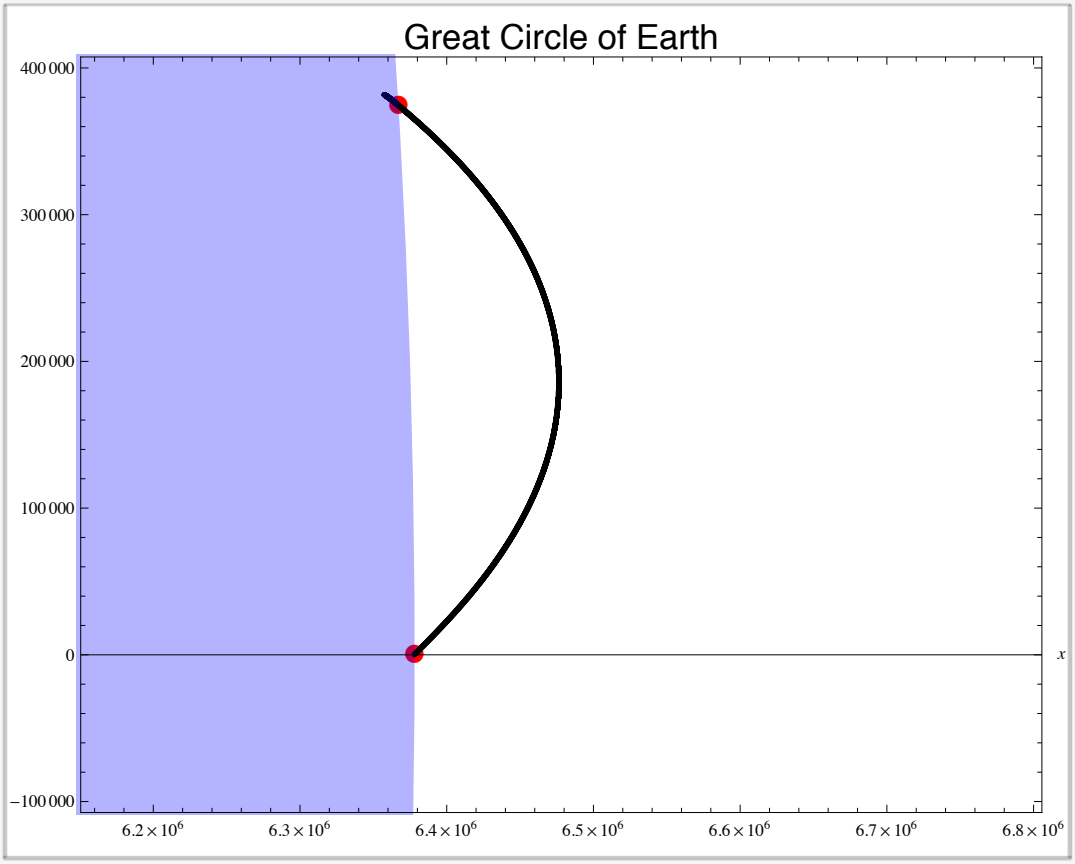
Show Less | Show More | Show Full Output | Set Size Limit...

Just as before, let's zoom in.

In[58]:=
```
Show[{
  ListPlot[{PHGreatCircle, TGGreatCircle}, PlotMarkers → {Automatic, 15},
   AxesOrigin → {0, 0}, AxesLabel → {x, y}, PlotStyle → Red],
  ListPlot[PositionSolutionListX, PlotMarkers → {Automatic, 4},
   AxesOrigin → {0, 0}, PlotStyle → Black],
  Graphics[{Blue, Opacity[0.3], Disk[{0, 0}, EarthRadius]}]},
 PlotRange → {{EarthRadius - 10^5, EarthRadius + 3 * 10^5}, {-10^5, 4 * 10^5}},
 AspectRatio → Automatic, ImageSize → 550,
 PlotLabel → Style["Great Circle of Earth", FontSize → 18, FontFamily → "Helvetica"],
 Frame → True]
```

Out[58]=

A very large output was generated. Here is a sample of it:



Show Less | Show More | Show Full Output | Set Size Limit...

Discussion Questions.

1. *Assuming the enemy watercraft has the dimensions of the U.S.S. Nimitz, what is the permissible window of angles and speeds which will result in a hit?*

According to Wikipedia (*http://en.wikipedia.org/wiki/USS_Nimitz_(CVN-68)*), the U.S.S. Nimitz has a length of 1092 feet and a beam (nautical width, or the width at the widest point) of 252 feet.
We want to place the boat's center at the target's position as calculated earlier, and then tweak our initial projectile parameter's to ensure a hit.
Since our given dimensions are in units of meters, let's convert these dimensions. Assume the boat is a

rectangular shape, so it's surface area is a rectangle and can be easily computed if needed.

In[36]:=
```
TGLength = N[QuantityMagnitude[UnitConvert[Quantity[1092, "Feet"], "Meters"]]]
TGWidth = N[QuantityMagnitude[UnitConvert[Quantity[252, "Feet"], "Meters"]]]
```

Out[36]= 332.842

Out[37]= 76.8096

Now, as previously stated, let's place the center of the boat at

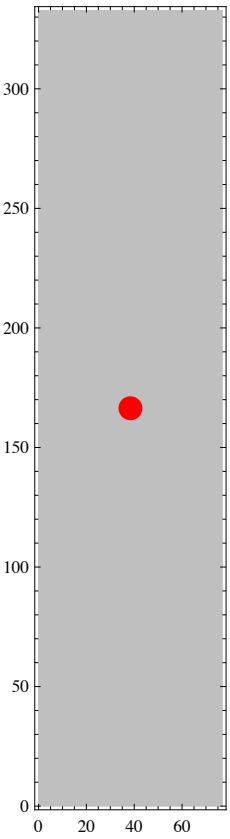
In[38]:=
```
TGGreatCircle
```

Out[38]= $\{6.3671 \times 10^6, 374\,457.\}$

So if we let the boat be a gray rectangle and our target point be a red dot, we have

In[39]:=
```
Show[Graphics[{Gray, Opacity[0.5], Rectangle[{0, 0}, {TGWidth, TGLength}]}],
 Graphics[{Red, Disk[{TGWidth / 2, TGLength / 2}, 5]}],
 Frame → True
]
```

Out[39]=



where the boat is running "parallel" to the earth's surface (sitting on the water as it physically would). On our great circle, notice that we only need to take into account the length of boat (if we assume the boat is pointed at us), since in the great circle plane the boat is represented as a "tangent" line segment that lies on the surface of the earth at the target's position. So to find the "top" and "bottom" of the boat, let's just take the target position and add half the length of the boat to the y component to find our upper bound, and subtract half the both length to find our lower bound.

Before we perform our calculations, we need to be careful about what point to use when determining when our projectile hits the boat (or surface of the earth). We cannot simply take the last point in our WorkingSolutionListX, since depending on our final time, the projectile could be inside the Earth's radius (since we did not instruct our code to stop running once the projectile reached a certain distance from the center of the Earth). To find the point where our projectile hits the Earth, we can let *Mathematica* check each element in our WorkingSolutionListX, find the position vector of each element, calculate the vector magnitude, and compare it to the Earth's radius. We want to the code to stop running once an element's magnitude is less than the Earth's radius. We can call this function PickOutSurfacePoint, and it's output is the element from RK4 where our projectile is closest to the surface (and hopefully boat).

In[40]:=
```
PickOutSurfacePoint[RK4SolutionList_] :=
 Module[{TempList, TempList2, PositionInList},
  TempList = RK4SolutionList[[All, 2, {1, 2}]];
  TempList2 = Select[Map[Norm, TempList], # > EarthRadius &];
  PositionInList = Length[TempList2];
  RK4SolutionList[[PositionInList]]
 ]
```

Now we can calculate our desired quantities.

In[41]:=
```
TGGreatCircleTopBound = TGGreatCircle[[2]] + TGLength / 2
TGGreatCircleBottomBound = TGGreatCircle[[2]] - TGLength / 2
PickOutSurfacePoint[WorkingSolutionListX][[2, {1, 2}]]
```

Out[41]= 374 623.

Out[42]= 374 290.

Out[43]= $\{6.36714 \times 10^6, 374\,345.\}$

To determine if we hit our target we need to check that the y-component of the projectile's impact coordinate falls in-between the upper and lower bounds set by the boat. The function below will perform this comparison.

In[44]:=
```
DidWeHitOurTarget[RK4SolutionList_] := Module[{ImpactPoint},
  ImpactPoint = PickOutSurfacePoint[RK4SolutionList][[2, {1, 2}]][[2]];

  If[TGGreatCircleBottomBound < ImpactPoint < TGGreatCircleTopBound,
   Print["We win!"], Print["We missed!"]]
 ]
```

All that's let to do is run our function on our WorkingSolutionListX, recalling that this solution was generated by choosing an angle of 45° and initial velocity of ~4405 m/s.

In[45]:=
```
DidWeHitOurTarget[WorkingSolutionListX]
```

```
We win!
```

As another check, we can find the distance from our final point and subtract EarthRadius to see how far off the ground our projectile is. The smaller the difference, the closer we are to hitting the boat.

In[46]:=
```
Norm[{PickOutSurfacePoint[WorkingSolutionListX][[2, {1, 2}]]}] - EarthRadius
```

Out[46]= 38.5568

So we are ~38 meters above the ground; however, recall that our solution is a set of points—not a continuous path. So if we were to increase the number of intervals RK4 takes, we could obtain a final point above the surface that would be closer in radial magnitude than our current solution (even better

we could interpolate over our solution path and find where that interpolating function intersects the Earth). Knowing that the draft (height of the boat as measured from the water to the bottom of the boats hull) is only ~11m, are final solution is a bit high. However, remember that the target will have a height much larger than ~11m since there are multiple levels, towers, and aircraft on top of the hull. So our height is indeed low enough to hit the boat!

Now that we have a certain initial velocity that we know hits the boat, we can find a range of velocities, and thus a range of launch angles, that also result in a hit. Assuming a launch angle of ~45°, let's impose that v_x = v_y. By experimentation, I have found the ranges that our initial velocity can be so that we hit the boat

In[47]:=
```
DidWeHitOurTarget[N[RK4[X, {EarthRadius, 0.0, 3114.9, 3114.9}, 0, 325, 5000]]]
DidWeHitOurTarget[N[RK4[X, {EarthRadius, 0.0, 3116.1, 3116.2}, 0, 325, 5000]]]
```

```
We win!
```

```
We win!
```

So our range of velocities where launch angle θ = 45° is approximately $\sqrt{2}$ * [3114.9, 3116.2] =

In[49]:=
```
NumberForm[Sqrt[2] * {3114.9, 3116.2}, 15]
```

Out[49]//NumberForm=
```
{4405.13382543595, 4406.97230306704}
```

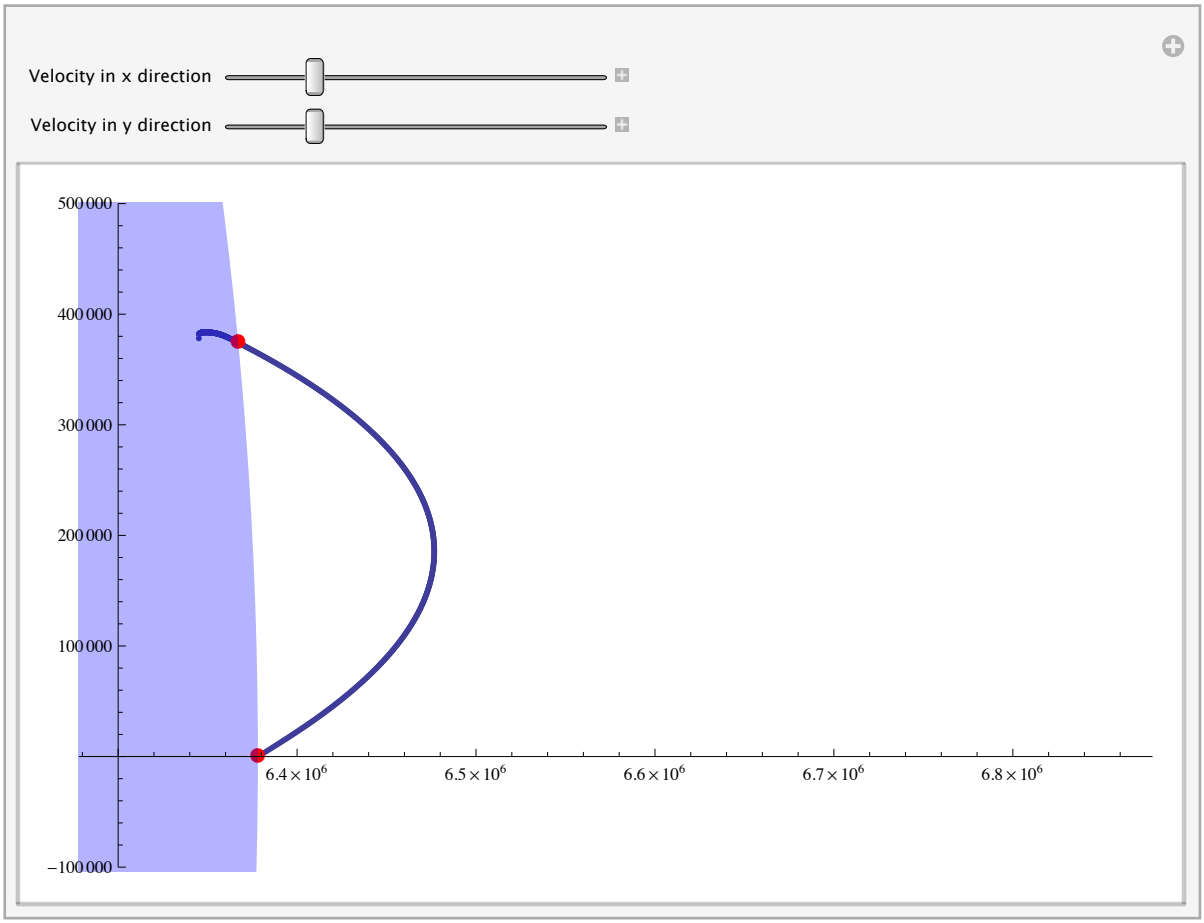
So for one specific angle we have our range of velocities, but what about our other angles? Instead of calculating a range for multiple launch angles, I made a Manipulate plot so that one can vary the initial velocity in the x and y directions, and consequently pick a launch angle. *Theoretically*, for every launch angle θ in the open interval (0°, 90°) there should be a velocity {xPrime, yPrime} = {v_x, v_y} that hits our target. We can visually see this by dragging the sliders to find new solutions to our equation. As a test, set the y-directional velocity to 0, so that we are firing straight up. Our projectile path should go straight up and back down (which it does!).

Note I: the manipulate below will contain error messages when the notebook is open and has not been evaluated.

Note II: the manipulate is running RK4 for 650 seconds, a final time that is much larger than the one used earlier. This is to account paths that extend a great distance off the Earth (longer flight time) so that RK4 can still calculate where the projectile lands. This longer time consequentially produces underflow errors for some particular velocities—this may be a result of an n value (number of intervals) that is too low to accurately model the path, i.e., *some debugging is still needed*!

In[50]:=
```
Manipulate[
 Show[
  ListPlot[N[RK4[X, {EarthRadius, 0.0, a, b}, 0, 650, 1000]][[All, 2, {1, 2}]],
   PlotRange → {{EarthRadius - 10^5, EarthRadius + 5 * 10^5}, {-10^5, 5 * 10^5}}],
  ListPlot[{PHGreatCircle, TGGreatCircle}, PlotMarkers → {Automatic, 12},
   AxesOrigin → {0, 0}, AxesLabel → {x, y}, PlotStyle → Red],
  Graphics[{Blue, Opacity[0.3], Disk[{0, 0}, EarthRadius]}],
  ImageSize → Large],
 {{a, 3115, "Velocity in x direction"}, 0, 15 000, 1},
 {{b, 3115, "Velocity in y direction"}, 0, 15 000, 1}]
```

Out[50]=



2. *What is the kinetic energy of the projectile on impact?*

From classical mechanics we know that an object with mass moving at non-relativistic speeds has a kinetic energy of
K = 0.5*m*v^2,
so let's plug in our values, using the magnitude of the velocity of the projectile as it hits the boat. To find this velocity, we can use our PickOutSurfacePoint function just as before and select only the velocity components.

In[51]:=
```
PickOutSurfacePoint[WorkingSolutionListX]
ImpactVelocity = Norm[PickOutSurfacePoint[WorkingSolutionListX][[2, {3, 4}]]]
```

Out[51]= $\{297.31, \{6.36714 \times 10^6, 374\,345., -650.277, 545.535\}\}$

Out[52]= 848.804

So we obtain

In[53]:=
```
KineticE = 0.5 * m * ImpactVelocity^2
```

Out[53]= 6.53598×10^8

in Joules.

3. *How does this compare to the chemical energy of an equivalent weight of TNT?*

Wikipedia (*http://en.wikipedia.org/wiki/Trinitrotoluene*) states *"TNT is reported to contain 2.8 mega joules per kilogram explosive energy."*
Using this conversion we can find the amount of equivalent TNT required to be

In[54]:=
```
TNTConversion = 2.8 * 10^6;
TNTWeight = KineticE / TNTConversion
```

Out[55]= 233.428

So to obtain the same amount of energy from our rail-gun shot, one would need 132kg of TNT! That's approximately

In[56]:=
```
N[QuantityMagnitude[UnitConvert[Quantity[TNTWeight, "Kilograms"], "Pounds"]]]
```

Out[56]= 514.621

pounds of explosive!