The Oerth Project

Abstract

This paper explores the dynamics of planetary projectile motion on Oerth, a fictional planet. Using Newton's laws and equations, projectile motion will be calculated. Including how trajectories change as velocities approach the escape velocity. It will discuss the deviation of expected and the actual distances over theta. Lastly, launching a projectile into orbit from surface is discussed.

Project 2
Mathematics 2130
Submitted by: John Hollett
Submitted to: Ivan Booth
November 11, 2015

1 Introduction

This paper observes a fictional planet called Oerth. Oerth is a planet with similar gravity, mass, and radius. Scientists want to build a weapon that can be used to fire at particular locations on the planet on any given day. This report will discuss these mechanics. This will be completed with a numerical analysis approach.

Calculations are performed in the programming language C#. Graphing analysis is done using Maple plots. With C#, a utility was created for graphing the planetary projectile. It was used to find position of the projectile during each calculation. How this data was gathered and used is explained along with the physics used.

2 Equations

This section explains the main details used to analyze trajectories of the projectile. The gravitational constant (G) and Oerth's mass (M) are consistently used in the calculations. Formulae for distance traveled (R) and force exerted by Oerth (F). Radius of Oerth (r) and the mass (m) of the object being fired. But, the mass of the projectile is neglected. This is due to the mass of the projectile being much smaller than Oerth.

$$R = \frac{v_o^2 \sin 2\theta}{g}, \text{Where } g = 10.0\ m \cdot s^{-2}, v_o = \text{ Initial Velocity} \tag{1}$$

$$F = \frac{GMm}{r^2}, \text{Where } G = 6.67 \times 10^{-11} N \cdot m^2 \cdot kg^{-2}, \quad M = 5.40 \times 10^{24} kg \tag{2}$$

$$V_{Escape} = \sqrt{\frac{2GM}{r}}, \text{ and } V_{Orbital} = \sqrt{\frac{GM}{r}} \tag{3}$$

With projectile motion problems, there are two vector components in these equations. An X and Y component which make up a vector $\hat{U}$ which describes the positional vector. There is also a velocity vector $\hat{V}$ which is affected by the acceleration caused by gravity. Neglecting wind resistance, velocity parallel to the surface is not affected by gravity and is not reflected in these equations. Velocity also has an X and Y component. The escape velocity dictates the maximum magnitude possible at any angle from the center of the mass to the object's initial position. Any velocity greater than the escape velocity will be outside of Oerth's influence. There is also orbital velocity, which is the minimum required of any orbit at a particular radius from a mass.

$$\hat{U} = \hat{x} + \hat{y} \tag{4}$$

$$\hat{V} = \hat{v_x} + \hat{v_y} \tag{5}$$

Where t is time in relation to the current location of $\hat{x}$ and $\hat{y}$. For $\hat{v_x}$ and $\hat{v_y}$, it is velocity in relation to time. Lastly, Δt is the step change in time. Depending value used for Δt, the accuracy of the projectile goes up or down in the numerical analysis of the projectile. The

equations for finding each component of Equation (4) and (5) are as follows.

$$\begin{aligned}\hat{x} &= \hat{x}(t) + \hat{v_x}(t)\Delta t \\ \hat{y} &= \hat{y}(t) + \hat{v_y}(t)\Delta t \\ \hat{v_x} &= \hat{v_x}(t) + \ddot{x}\Delta t \\ \hat{v_y} &= \hat{v_y}(t) + \ddot{y}\Delta t\end{aligned}$$

The acceleration that affects these equations is part of Kepler's Law[4], Although, these listed here are simplified, they are based on a radial unit vector[3] which is more complicated than what was used. A simplified version is used instead, which works for the purposes of this paper.

$$\ddot{x} = -\frac{GM\hat{x}(t)}{(\hat{x}(t)^2 + \hat{y}(t)^2)^{3/2}} \text{ and } \ddot{y} = -\frac{GM\hat{y}(t)}{(\hat{x}(t)^2 + \hat{y}(t)^2)^{3/2}} \tag{6}$$

In the above, $\hat{x}(t)$ and $\hat{y}(t)$ are components of the positional vector $\hat{U}$ in Equation (4) and where t describes the current position at that point in time. These two acceleration functions are used to find the change in velocity vector $\hat{V}$ components in equation (5). These values are then used to find the positional vector components in equation (4).

3 Analysis

3.1 Universal Constants and Expected Values

Equation (1) describes the expected distance based on angle and velocity. This can be tested numerically, and is within the scope of this report. The equation (2) is used to find the force exerted on the projectile by Oerth. This force is derived by Equation (6), which manipulates the acceleration of the particle launched given the current positional vector quantity from the center of Oerth.

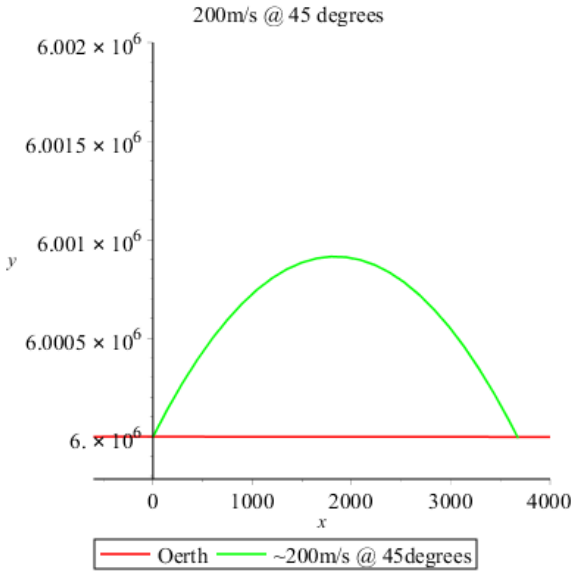
The calculations dictate that for any velocity at which the projectile is launched, it must never exceed the escape velocity described by equation (3). Should this occur, it would be lost into space. This is unlikely to happen due to the atmospheric penetration factor that would have to be at the correct angle, velocity and resistance to make it into space.[2]

There are numerous equations that can be used to numerically solve projectile motion, but this report utilizes simplified versions. The equations used are sufficient to find data that can be utilized to prove the concepts in this paper.

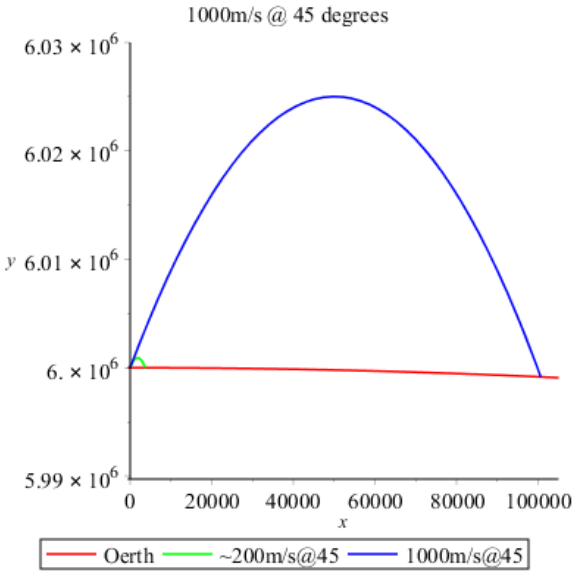
3.2 Launching Speeds

In the Figure (1a) the projectile is traveling a short distance around Oerth. This is the first element of data collected. The initial velocity was approximately $200m \cdot s^{-1}$. Using Equation (1), the result was $3,686.4m$. The actual numerical value is $3,678.472m$ which is less than the expected result. Figure (1b) had an initial velocity of $1000m \cdot s^{-1}$ and the result of equation (1) was $100,000m$. The actual was $100,692.908m$. As the velocity of launch is increased, the values of R become less accurate.

Figure 1: Launching at Smaller Velocities

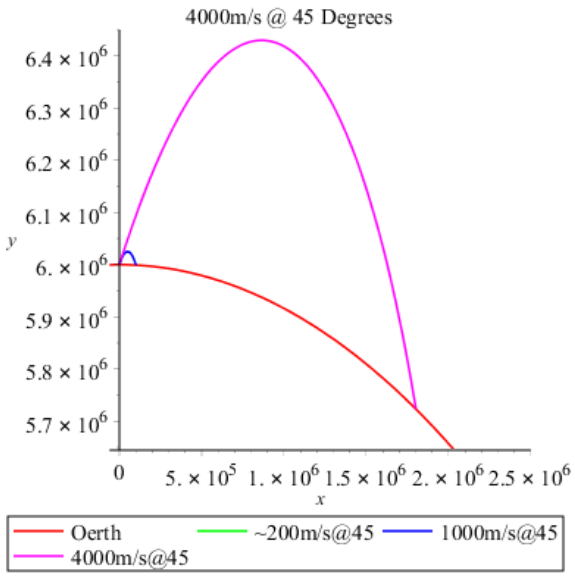


(a) $192m \cdot s^{-1}$ @ 45

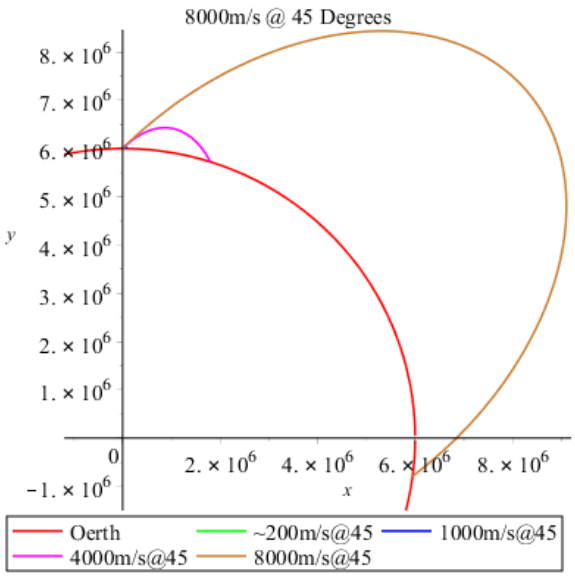


(b) $1000m \cdot s^{-1}$ @ 45

Figure 2: Launching at Larger Velocities



(a) $4000m \cdot s^{-1}$ @ 45



(b) $8000m \cdot s^{-1}$ @ 45

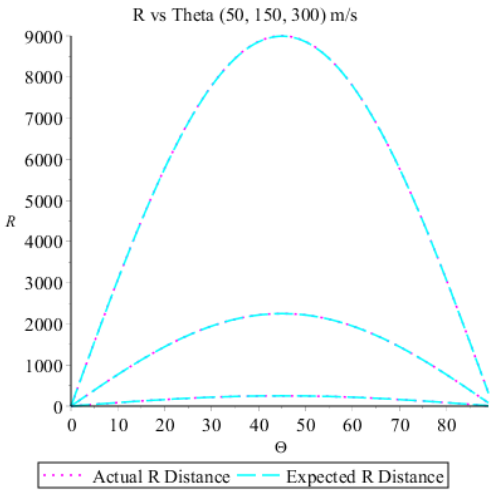
With the larger velocities in Figure (2a) and (2b), there are some early trends to observe regarding how the projectile is being launched.The first is the shifting of the $\hat{x}$ and $\hat{y}$ component acceleration on the particle. It does not become fully visible until larger V_o values. It is important to note that these projectiles are different from those described in 2D-Kinematics because in this case gravity shifts from the Y-axis to the X-axis.

As the orbits get larger, more distance is traveled. Larger orbits take more time to complete a revolution if it does not escape Oerth's influence. When approaching the escape velocity, it takes more time to return as it becomes infinitesimally closer to this value. It is more difficult to calculate as it gets approaches escape velocity. This can be observed in figure (4), where as the velocities get closer to V_{Escape}, they become more skewed elliptical in shape.

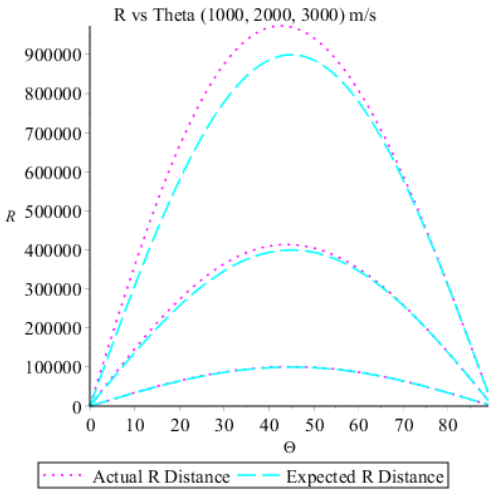
In figure (2a) and (2b), the larger values of initial velocity start to approach the speeds necessary to complete orbits or travel distances much further. To achieve half the distance around the planet, the projectile would need to be launched closer to the surface. To launch a projectile around the earth, the gravity needs to be utilized to manage longer distances. This can also be seen in the figure (2b). The angle is just as important as the velocity. The advantage of launching projectiles at angles closer to the surface makes use of gravity to slingshot it around the planet much easier.

3.3 R vs Theta

Figure 3: R vs Theta



(a) Smaller Velocities



(b) Larger Velocities

In Figure (3a), there is a relationship between R, which is the distance traveled along the surface, and Theta, which is the angle of launch. This figure contains the launch velocities $50m \cdot s^{-1}$, $150m \cdot s^{-1}$ and $300m \cdot s^{-1}$. To make it easier to see the relation between these velocities, two colors and different style lines were used. This is important because at lower velocities, this relationship stays true. Using the equation (1), the data stays within an acceptable error at these velocities. It is not exact, but it is adequately close.

When examining figure (3b) there is a noticeable change. It represents the speeds $1000m \cdot s^{-1}$, $2000m \cdot s^{-1}$ and $3000m \cdot s^{-1}$. After $2000m \cdot s^{-1}$ the actual R distance changes substantially. And as this change is noticeable early in velocities leading up to V_{Escape}, it is expected this deviation will be larger as it approaches V_{Escape}. Differences in these values is more noticeable at the 45° angle. There is also bias towards velocities at angles $0° \leq \Theta < 45°$.

It is important to make this distinction between actual and expected R values. The shifting gravity and decline in the surface affects the projectile's R distance as velocities increase. To quantify this from equation (1), as $v_o \to (V_{Orbital} \vee V_{Escape})$ with $0° \leq \Theta < 45°$, it will deviate increasingly.

3.4 Orbital Plausibility

This section describes whether or not it is possible to put a projectile into an orbit from the surface. The short answer is yes. However, this does not come without complications. The approach to this problem also matters significantly.

To calculate this numerically, Δt needs to be low so that it is accurate enough to eliminate some of the error that occurs doing this in steps. To ensure that the calculations are successful, the initial height must be increased. However, this means the projectile is no longer launched from the surface. The safest value found was at least five meters above the surface. A circular orbit is possible with no ellipse, however it is required to be launched at zero degrees to the surface. The velocity needs to be sufficient to negate the gravity of Oerth. This was found to be approximately $7748m \cdot s^{-1}$.

Using more rigorous testing, the results are proof that it is possible. While this is certainly possible, the test assumes the surface is perfectly spherical. The path of the projectile would need to be free of obstacles in its path. This test neglects wind resistance as well. This would be impossible in real scenarios.

In an elliptical orbit, some of these concerns are alleviated. However, if the orbit is large enough to hit the atmosphere it will burn the projectile.

Achieving an orbit from the surface is impossible. Launching at the surface, the projectile will return to the position it originated from which would be the surface itself. This is shown in figure (4). These trends will not change at orbital velocity. Ignoring these problems entirely and launching the projectile above the surface makes an orbit possible.

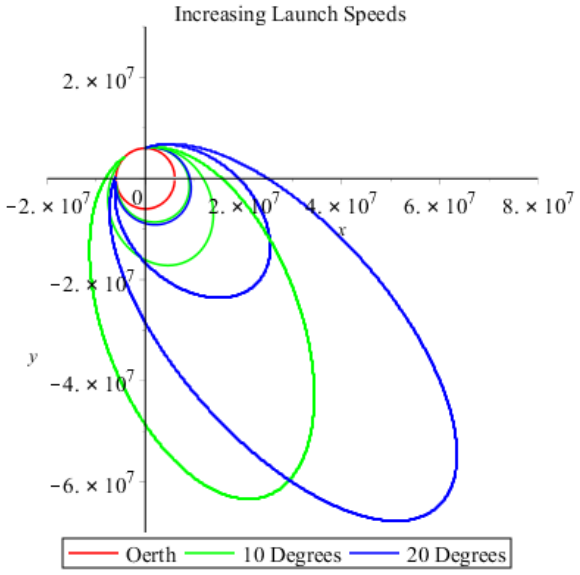


Figure 4: Increasing velocities @ 10 and 20 degrees

Assuming that the cannon is above the ground or raised up and lowered after a launch makes this possible. In a perfect circular orbit initiated at the surface, the orbit itself would the surface. In an elliptic orbit, it still makes contact with the surface after a completed orbit. It is improbable to get a proper orbit from the surface under real conditions. However, it can also be said it is not impossible either under the right conditions.

4 Technical Details

This section describes the calculations that have been used. Using the code referenced in the index created from the sources listed and project details, the data was calculated. This allowed data to be plotted as accurately as possible given a Δt.

The calculations that take place are carried out within a defined object. This object has its own values for the relevant launch. It has properties defined like magnitude, V_{Orbit} and V_{Escape} to help accuracy. It takes $\hat{x}$ and $\hat{y}$ from equation (4) as part of the initial position. A velocity and angle is given in order to calculate $\hat{v}_x$ and $\hat{v}_y$ for equation (5). This is done using the commonly used Pythagoras theorem.

These calculations come with serious drawbacks. One of the most profound observations is the lack of accuracy in higher valued changes in time. This affects calculations done with change in time stepping greater than half a second. This produces less accurate results. Even lower values than this can still be enough to throw off the accuracy depending on velocity. The reason for this is at the position where each point is recorded, the next point acts like a line with length determined by the value of Δt.

This can cause the calculations at each step to change the X and Y components of the $\hat{U}$ vector. During the period between calculations, changes that should be happening with the gravity do not happen until the next step in calculation. To visualize this, it is similar to using a series of lines to create a circle. It will not be rounded. Using increasingly shorter lines makes for a rounder circle.

During each calculation, a step function is used which is defined to calculate velocity and acceleration before applying it to the position. This happens at each change in time defined by Δt. Using some rough scaling, the number of points is reduced but, the accuracy of the calculations is unchanged. To calculate the radians around Oerth, the well known equation was used:

$$cos\ \Theta = \frac{\hat{F} \cdot \hat{G}}{\|\hat{F}\|\ \|\hat{G}\|}$$

With this equation, using conditional statements, the distance around the surface of Oerth is found. This works for the numerical analysis of R in equation (1) because, R does not take into account the circular properties of Oerth or other scenarios.

5 Conclusion

The dynamics of projectile motion on a planetary scale, while very similar, has differences. Some of these are the limits of the velocity and angles these projectiles can have. The escape velocity plays a part in determining a speed limit, relative to the distance from planets. Simple physics formulae used in basic kinematics are sufficient for the low-scale, but on the larger-scale, more advanced methods are necessary. Without these it would not be possible to accomplish much space exploration and much more.

The projectile motion that is used in this report does not fully convey how important these calculations are. The laws these formulae are derived from are used in calculations involving satellite motion and whether or not orbits are clear of any intersecting objects, like space stations and so on.

If these methods for calculating motion did not exist, devastating accidents could occur more frequently. Objects moving at the speed necessary to maintain orbit, makes for deadly collisions that can easily destroy objects such as space stations. Debris that this creates is mutually assured to be destructive to anything else in its path.

Furthermore, when these laws and formulae are used to their full potential, they can calculate and plan long distance space travel. The probes sent to space use orbital dynamics and escape velocities to project paths to other masses in space like a slingshot.

Understanding of these concepts is important for the future of space travel and exploration. They will always play a role in models of space exploration. Many of these mechanics are quantified to explain behaviors of the visible universe. These equations as a result, must be appreciated.

6 C# Code

Listing 1: KeplerProjectile.cs

```
  using System;
using System.IO;

namespace Math2130Project2 {

   public class Kepler {
      private static uint N = 0;

      public static void Run( KeplerProjectile p , bool Orbital ) {
         Console.WriteLine( "-=-=-=-=-=-" + "Points(V=" + Math.Round( p.
             Velocity , 2 ) + ").txt" + "-=-=-=-=-=-" );
         string path = @"Points(V="+Math.Round(p.Velocity,2)+").txt";

         StreamWriter file;
         bool writeable = true;
         if ( File.Exists( path ) && !Orbital ) {
            try {
               File.Delete( path );
            } catch ( IOException ) {
               Console.WriteLine( "Could not write file: " + path );
               writeable = false;
            }
         }
         if ( writeable ) {
            Console.WriteLine( "V_orbit = " + p.OrbitVelocity );
            Console.WriteLine( "V_escape = " + p.EscapeVelocity );
            Console.WriteLine( "Start = Pt[{0:0.000}, {1:0.000}], Velocity
                [{2:0.000},{3:0.000}]" , p.X , p.Y , p.VelocityX , p.VelocityY
                );
            Console.WriteLine( "Time(s): {0:0.000}, Range(m): {1:0.000},
                Distance(m):{2:0.000}\nVelocity(m/s):{3:0.000}, Gravity(m/s^2)
                :{4:0.000}\n" , p.Time , p.R , p.Distance , p.Velocity , p.
                Gravity );
            if ( p.Velocity > p.EscapeVelocity ) {
               Console.WriteLine( "Projectile too fast\nV:{0:0.000} EV
                   :{1:0.00}" , p.Velocity , p.EscapeVelocity );
               Console.ReadKey();
               return;
            }
            double limit = 2*(1-p.DeltaStep)*(Math.Pow(p.Velocity, 0.333 ));

            if ( !Orbital ) {
               file = File.CreateText( path );
               file.WriteLine( "{0:0.000} {1:0.000}" , p.X , p.Y );
               while ( !p.Landed ) {
                  for ( int i = 0; i < limit && !p.Landed; i++ ) {
                     p.Step();
                  }
                  file.WriteLine( "{0:0.000} {1:0.000}" , p.X , p.Y );
               }
```

```
            file.Close();
        } else {
            while ( !p.Landed ) {
                for ( int i = 0; i < 16 * limit && !p.Landed; i++ ) {
                    p.Step();
                }
                Console.WriteLine( "{0:0.000} {1:0.000} RADS: {2:0.000}" , p
                    .X , p.Y , p.R );
            }
        }
        Console.WriteLine( "Finish = Pt[{0:0.000}, {1:0.000}], Velocity
            [{2:0.000},{3:0.000}]" , p.X , p.Y , p.VelocityX , p.VelocityY
            );
        Console.WriteLine( "Time(s): {0:0.000}, Range(m): {1:0.000},
            Distance(m):{2:0.000}\nVelocity(m/s):{3:0.000}, Gravity(m/s^2)
            :{4:0.000}\n" , p.Time , p.R , p.Distance , p.Velocity , p.
            Gravity );
    }
    Console.WriteLine( "-=-=-=-=-=-" + "End" + "-=-=-=-=-=-\n" );
}

public static void Orbital() {
    KeplerProjectile p1 = new KeplerProjectile(0,6.0E+6+5,1,10,0.5);
    KeplerProjectile p2 = new KeplerProjectile(0,6.0E+6+5, p1.
        OrbitVelocity,0,0.0001);
    Run( p2 , true );
}

public static void Main( string[] args ) {
    //Orbital();
    RvsTheta( 50 );
    RvsTheta( 150 );
    RvsTheta( 300 );
    RvsTheta( 1000 );
    RvsTheta( 2000 );
    RvsTheta( 3000 );
    Console.ReadKey();
}

public static void BasicCalculations() {
    KeplerProjectile p1 = new KeplerProjectile(0, 6.0E+6, 192, 45, 0.1);
        //First Projectile Fig 1
    KeplerProjectile p2 = new KeplerProjectile(0, 6.0E+6, 1000, 45, 0.2);
        //Second Projectile Fig 2
    KeplerProjectile p3 = new KeplerProjectile(0, 6.0E+6, 4000, 45, 0.3);
        //Third Projectile Fig 3
    KeplerProjectile p4 = new KeplerProjectile(0, 6.0E+6, 8000, 45, 0.4);
        //Fourth Projectile Fig 4
    KeplerProjectile p5 = new KeplerProjectile(0, 6.0E+6, 8600, 10, 0.5);
    KeplerProjectile p6 = new KeplerProjectile(0, 6.0E+6, 10000, 20, 0.5)
        ;
    KeplerProjectile p7 = new KeplerProjectile(0, 6.0E+6, 10600, 20, 0.5)
        ;
    KeplerProjectile p8 = new KeplerProjectile(0, 6.0E+6, 10500, 10, 0.5)
```

```
        ;
    KeplerProjectile p9 = new KeplerProjectile(0, 6.0E+6, 10700, 20, 0.5)
        ;
    KeplerProjectile p10 = new KeplerProjectile(0, 6.0E+6, 10800, 10,
        0.5);
    KeplerProjectile p11 = new KeplerProjectile(0, 6.0E+6, 10900, 20,
        0.5);
    KeplerProjectile p12 = new KeplerProjectile(0, 6.0E+6, 10910, 10,
        0.5);
    Run( p1 , false );
    Run( p2 , false );
    Run( p3 , false );
    Run( p4 , false );
    Run( p5 , false );
    Run( p6 , false );
    Run( p7 , false );
    Run( p8 , false );
    Run( p9 , false );
    Run( p10 , false );
    Run( p11 , false );
    Run( p12 , false );
}

public static void RvsTheta( int velocity ) {
    String path = @"RvsTheta@V"+velocity+"A.txt";
    String path2 = @"RvsTheta@V"+velocity+"B.txt";
    StreamWriter file;
    StreamWriter file2;
    if ( File.Exists( path ) && File.Exists( path2 ) ) {
        try {
            File.Delete( path );
            File.Delete( path2 );
        } catch ( IOException ) {
            Console.WriteLine( "Could not write file(s): " + path + ", " +
                path2 );
        }
    }
    file = File.CreateText( path );
    file2 = File.CreateText( path2 );
    for ( int i = 0; i <= 89; i++ ) {
        KeplerProjectile p = new KeplerProjectile(0,6.0E+6, velocity, i
            ,0.001);
        while ( !p.Landed ) {
            p.Step();
        }
        file.WriteLine( "{0:0.000} {1:0.000}", i , p.R );
        file2.WriteLine( "{0:0.000} {1:0.000}", i , p.R_Test );
    }
    file.Close();
    file2.Close();
}

public class KeplerProjectile {
    private const double G = 6.67E-11; //Gravitation Constant
```

```
private const double Mass = 5.40E+24; //Mass of Oerth
private const double Radius = 6.0E+6; //Radius of Oerth
public double Xo { get; private set; } // Original X coordinate of
    the projectile
public double Yo { get; private set; } // Original Y coordinate of
    the projectile
public double X { get; private set; } // Current X coordinate of the
    projectile
public double Y { get; private set; } // Current Y coordinate of the
    projectile
public double Angle { get; private set; } // The angle given for the
    current projectile
public double Magnitude { get { return Math.Sqrt( Math.Pow( X , 2 ) +
     Math.Pow( Y , 2 ) ); } } // The current magnitude (Distance) from
     center of Oerth
public double MagnitudeO { get { return Math.Sqrt( Math.Pow( Xo , 2 )
     + Math.Pow( Yo , 2 ) ); } } // The original magnitude from center
     of Oerth
public double DeltaStep { get; private set; } // The change step
    between Coordinates of X and Y
public double VelocityX { get; private set; } // Current X component
    of the velocity vector
public double VelocityY { get; private set; } // Current Y component
    of the velocity vector
public double Distance { get; private set; } // Current Total
    curvature distance traveled
public double Time { get; private set; } // Current time traveled
public double EscapeVelocity { get { return Math.Sqrt( ( 2 * G * Mass
     ) / Radius ); } } //Absolute maximum velocity or the projectile
    escapes orbit.
public double OrbitVelocity { get { return Math.Sqrt( G * Mass /
    Magnitude ); } } //The velocity required for a successful orbit
public double Velocity { get { return Math.Sqrt( Math.Pow( VelocityX
    , 2 ) + Math.Pow( VelocityY , 2 ) ); } } // Current velocity
    magnitude
public double Gravity { get { return ( G * Mass ) / Math.Pow(
    Magnitude , 2 ); } } // The current gravity based on distance from
     Oerth
public bool Landed { get { return ( Magnitude ) < Radius; } } //
    Returns the whether or not the projectile is below the radius,
    thus being landed.
public double R { get { return Rads * Radius; } } // The actual R
    distance
public double R_Test { get { return ( Math.Pow( Velocity , 2 ) * Math
    .Sin( 2 * ( ( Math.PI / 180 ) * Angle ) ) ) / Gravity; } } // The
    tested R distance using equation (1)

public KeplerProjectile( double x , double y , double vel , double
    angle ) {
    this.Xo = ( this.X = x );
    this.Yo = ( this.Y = y );
    this.Angle = angle;
    this.VelocityX = vel * Math.Cos( ( Math.PI / 180 ) * angle );
    this.VelocityY = vel * Math.Sin( ( Math.PI / 180 ) * angle ); ;
```

```
        this.DeltaStep = 0.5;
    }

    public KeplerProjectile( double x , double y , double vel , double
        angle , double step ) : this( x , y , vel , angle ) {
        this.DeltaStep = step;
    }

    public double Rads {
        get {
            double COS_THETA = Math.Abs(( Xo * X + Yo * Y ) / ( Magnitude *
                MagnitudeO ));
            double RADS=0;
            double Xs = X != 0 ? X/Math.Abs(X) : 1;
            double Ys = Y != 0 ? Y/Math.Abs(Y) : 1;

            if ( Xs > 0 ) {
                if ( Ys > 0 )
                    RADS = Math.Acos( COS_THETA );
                else {
                    RADS = Math.PI - Math.Acos( COS_THETA );
                }
            } else {
                if ( Ys < 0 ) {
                    RADS = ( Math.PI ) + Math.Acos( COS_THETA );
                } else {
                    RADS = 2 * ( Math.PI ) - Math.Acos( COS_THETA );
                }
            }
            return RADS;
        }
    }

    private double AccelX() {
        return -( G * Mass * X ) / Math.Pow( Math.Pow( X , 2 ) + Math.Pow(
            Y , 2 ) , 3.0 / 2.0 );
    }

    private double AccelY() {
        return -( G * Mass * Y ) / Math.Pow( Math.Pow( X , 2 ) + Math.Pow(
            Y , 2 ) , 3.0 / 2.0 );
    }

    private double VelX() {
        return ( VelocityX = VelocityX + ( AccelX() * DeltaStep ) );
    }

    private double VelY() {
        return ( VelocityY = VelocityY + ( AccelY() * DeltaStep ) );
    }

    public bool Step() {
        this.Time += this.DeltaStep;
        this.Distance += Math.Sqrt( Math.Pow( X - StepX() , 2 ) + Math.Pow
```

```
                ( Y - StepY() , 2 ) );
            return Landed;
        }

        private double StepX() {
            return ( X = X + ( VelX() * DeltaStep ) );
        }

        private double StepY() {
            return ( Y = Y + ( VelY() * DeltaStep ) );
        }
    }
  }
}
```

References

[1] "Newton's Laws of Motion" Wikipedia: The Free Encyclopedia. Wikimedia Foundation, Inc. 13 Oct 2015. Web. 18 Oct 2015

[2] "Atmospheric Flight" NASA History. http://history.nasa.gov/conghand/atmosphe.htm National Aeronautics and Space Administration, Web. 20 Oct 2015

[3] "Elementary Mathematical and Computer Tools for Electrical and Computer Engineers Using MATLAB" Jamal T. Manassah. https://books.google.ca/books?id=qGEImarSDDIC&pg=PA260#v=onepage&q&f=false Taylor & Francis Group, 2007, Web. 21 Oct 2015

[4] "The Radial Velocity Equation" Department of Physics & Astronomy http://w.astro.berkeley.edu/~kclubb/pdf/RV_Derivation.pdf San Francisco State University, Aug 2008, Web. 21 Oct 2015