

Comparing Numerical Integration Methods

Alex Clement

Amy Clement

Greg Fenchel

Jeff Fenchel

Jayson Lynch

April 02, 2008

Team 82

NM Supercomputing Challenge

Table of Contents

EXECUTIVE SUMMARY	4
PART I: DESCRIPTION OF PROGRAM	6
INTRODUCTION TO NUMERICAL INTEGRATION	6
CELESTIAL MECHANICS AND THE TWO-BODY PROBLEM.....	8
ANALYTIC SOLUTION.....	10
EULER'S METHOD.....	12
<i>Euler Cromer Method</i>	13
<i>Leap Frog Method</i>	14
THE RUNGE-KUTTA METHODS	15
<i>Gill's Method</i>	16
<i>Modified RK2</i>	17
ADAMS-BASHFORD METHOD.....	17
PART II: RESULTS AND CONCLUSIONS.....	19
STABILITY IN TIME STEP.....	19
<i>Analytic Solution</i>	19
<i>Euler's Method</i>	20
<i>Runge-Kutta 2</i>	21
<i>Runge-Kutta 4</i>	22
<i>Adams-Bashford</i>	22
<i>Gill's Method</i>	23
<i>Euler Cromer Method</i>	24
<i>Leapfrog Method</i>	24
<i>Conclusions</i>	26
IN-DEPTH ANALYSIS AND COMPARISON TO ANALYTIC SOLUTION	27
<i>Analytic v. Euler</i>	28
<i>Analytic v. Runge-Kutta 2</i>	31
<i>Analytic v. Runge-Kutta 4</i>	34
<i>Analytic v. Gill's</i>	37
<i>Analytic v. Adams-Bashford</i>	39
<i>Analytic v. Euler-Cromer</i>	41
<i>Analytic v. Leapfrog</i>	43
<i>Analytic v. Runge-Kutta 2 (Our Modification)</i>	45
COMPARING POSITION OF ORBITING BODY IN TWO METHODS	50
<i>Euler v. Runge-Kutta 2</i>	50
<i>Euler v. Adams- Bashford</i>	51
<i>Euler v. Gill's</i>	51
<i>Euler v. Modified RK2</i>	51
<i>Runge-Kutta 2 v. Runge-Kutta 4</i>	52
<i>Runge-Kutta 2 v. Euler-Cromer</i>	52
<i>Runge-Kutta 2 v. Leapfrog</i>	53
<i>Runge-Kutta 4 v. Adams-Bashford</i>	53
<i>Runge-Kutta 4 v. Gill's</i>	54
<i>Runge-Kutta 4 v. Modified Runge-Kutta 2</i>	54
<i>Euler-Cromer v. Gill's</i>	55
<i>Euler-Cromer v. Modified RK2</i>	55
<i>Gill's v. Leapfrog</i>	55
FINAL CONCLUSIONS	57
FUTURE WORK	60
ACKNOWLEDGMENTS.....	62
BIBLIOGRAPHY	63

PART III: APPENDIXES.....	64
APPENDIX I: THE CODE.....	64
APPENDIX II: ADDITIONAL GRAPHS.....	105
<i>Euler's Method</i>	105
<i>Runge-Kutta 2</i>	115
<i>Runge-Kutta 4</i>	123
<i>Adams-Bashford</i>	132
<i>Gill's Method</i>	141
<i>Euler-Cromer</i>	150
<i>Leapfrog Method</i>	159
<i>Modified Runge-Kutta 2</i>	168

Executive Summary

The goal of this project is to compare the orbital estimations made by a series of common numerical integration schemes. Numerical integration is a mathematical process that estimates the solution to a differential equation that cannot be solved analytically.

To test the numerical integration schemes, we set up an ideal two-body celestial mechanics problem in C#. We chose a problem that could be solved analytically so that we could compare the estimations produced by the numerical integration schemes to the true orbit of the celestial body.

Using C#, we implemented Euler's Method, Runge-Kutta Order 2, Runge-Kutta Order 4, Adams Bashford, Gill's Method, The Leapfrog Method, and Euler-Cromer Method. In addition, we designed and implemented our own method, a modified version of Runge-Kutta Order 2.

We compared methods based on stability with varying time steps, accuracy in position, accuracy in velocity, conservation of energy, conservation of angular momentum, and conservation of the eccentricity vector. After comparing our results, we determined that our modified Runge-Kutta Order 2 is the most stable solution. Gill's Method is slightly less stable, but more accurate with smaller time steps. Euler's Method and Adams Bashford are the least stable and, consequently, least accurate. Based on our results, we concluded that Gill's Method is the best numerical integration scheme to implement when solving an ODE numerically, especially when estimating orbits in a celestial mechanics problem.

Problem Statement

For years, scientists studying disease spread and orbital mechanics struggled to solve complex differential equations analytically. However, with the advent of computational modeling, scientists and mathematicians began creating numerical integration methods that could easily approximate the solution of a differential equation that could not be easily or efficiently solved analytically. Today the internet is littered with dozens of numerical integration schemes, but which one gives the most accurate approximation of the real solution?

There is no definitive answer to this question. Each ordinary differential equation behaves differently. However, using an ideal two-body orbital mechanics problem we can observe general patterns of behavior in each method. Because the two-body problem is centered on a simple ordinary differential equation that can be solved analytically as well as numerically, an exact solution, a point of reference and comparison, exists.

Part I: Description of Program

Introduction to Numerical Integration

When Leibniz and Newton simultaneously stumbled onto the vast field of graphical and functional analysis that we know as calculus, they introduced the world to a new array of mathematical possibilities. Along with these new problem-solving tools came a new set of mathematical conundrums. In the field of integration, mathematicians quickly discovered that there were still some ordinary differential equations that could not be solved analytically. In order to solve these equations, mathematicians designed a new method of problem solving, numerical integration. Just as the fathers of calculus had used small estimations to determine the integral of a function, Euler, Runge, Kutta, and Adams realized that small estimations could also be used to estimate the path that a function followed, that is, the solution to the problem. Numerical integration was immediately recognized as an invaluable tool, and many mathematicians created their own formulas to express estimations, numerical solutions, to ordinary differential equations that cannot be solved analytically. Although numerical integration has been a central part of mathematics for more than a century, there is still much discussion over the different formulas that have been used to estimate analytical solutions. Because each ordinary differential equation behaves differently, no single method can estimate every analytical solution perfectly. Numerical integration and the search for the ideal method, remain at the forefront of modern mathematics.

Numerical integration was created to solve ordinary differential equations (ODEs) that could not be solved analytically. An ordinary differential equation is a function that expresses the rate of change of a function with respect to the changing variables. In other

words, an ODE is expressed in the form $\frac{dy}{dx} = f(x, y)$. When solving an ordinary differential equation, the goal is to find the function that created a specific rate of change, given the initial values of the variables. Although some ordinary differential equations can be solved with simple algebra and integration, many cannot be solved analytically. However, in physics, engineering, astronomy, and chemistry the solutions to ordinary differential equations are critical. It is relatively simple to measure the rate of change of a function (i.e. velocity, temperature change, decay rate, rate of population growth) and determine an ordinary differential equation that can be solved to express the true function. Solving these ordinary differential equations can provide a link between the theoretical and the experimental, and are critical in every field of science. Although numerical integration schemes do not provide us with an exact solution to some of these more difficult equations, the numerical estimations are invaluable tool in natural science as well as mathematics.

Today, the internet is littered with hundreds of differential numerical integration schemes. All of these methods can be divided into two general categories: implicit and explicit. Explicit methods are methods that find the numerical approximation to a solution given a set of initial values. Implicit methods solve for the position at a given time based on a function of the current and next value and can often be more computationally intensive. This project focuses solely on explicit methods.

The first explicit methods appeared shortly after the introduction of Calculus. Rightly, the most famous of these methods, was the first. In the late 18th century, Leonhard Euler invented the first numerical integration scheme, aptly named Euler's

method. Although Euler is best known for his famous identity $e^{\pi i} + 1 = 0$ and his work with number theory, he also recognized the value of numerical approximation, and greatly expanded the emerging field of calculus. A century later, German mathematicians Carl David Tolmé Runge and Martin Wilhelm Kutta created a new family of explicit numerical integration schemes known as the Runge-Kutta methods. The most popular Runge-Kutta method is the fourth order method, more commonly known as RK4 and is often coupled with predictor correctors or other methods of optimizing step sizes.

Numerical integration has a wide variety of applications in Physics, Astronomy, Chemistry, and Biology. In Biology, scientists use numerical integration to estimate rates of growth for bacterial populations. In Chemistry, numerical integration is often used to analyze changes in energy during chemical reactions. It is also used in fluid dynamics (See Adams-Bashford). In Engineering, numerical integration is often used to analyze current flow in a circuit. But perhaps the most common application of numerical integration is in Astronomy. Astronomers frequently use numerical integration to determine the orbit of a celestial body given an initial observed position and velocity. Our project focuses on this particular application of numerical integration.

Celestial Mechanics and the Two-Body Problem

Celestial mechanics is a subset of astronomy that describes the motion of bodies in an orbital system. As opposed to orbital mechanics which deals specifically with orbiting spacecraft, celestial mechanics is a broader field of astronomy which deals with both natural and man-made bodies. Celestial mechanics uses Newton's laws of motion

and gravity to determine the path of a given body in a system. In 1605, Johannes Kepler derived three mathematical laws from Newton's laws of motion and gravity that dealt specifically with celestial bodies in our solar system. The first of these laws states that the paths of all celestial bodies around the sun are ellipses with respect to the sun at one focus. The second law says that a celestial body sweeps out equal areas over equal amounts of time. Therefore, the celestial body is moving faster when is closer to the sun, resulting in a changing velocity in non-circular orbits. The third law relates the period of the orbit to the length of the semi-major axis (the long radius of the ellipse). This relationship shows us that a planet in a small orbit moves much faster than one in a large orbit. These laws allow astronomers to model the orbit of a celestial body.

For our program, we set up an ideal two-body problem. Because the goal of this project is to compare numerical integration methods, we did not want to focus on creating a complex celestial mechanics problem. In addition, we chose to program a few methods that dealt specifically with the two-body problem. In an ideal two-body problem, there are two celestial bodies that orbit around each other. Often one body is substantially larger than the other. An example of a common two-body problem is the moon orbiting around the Earth or a planet rotating around the sun (we use the latter for our program). We say that this problem is ideal because the orbit is a perfect ellipse, and all of the orbital parameters remain constant. These parameters include the energy, angular momentum, and eccentricity of the system. For our problem, we used a series of general orbital parameters in order to determine our analytic solution. These parameters included Velocity, Position and Mass. The velocity is the speed and direction at which

the planets travel in the model. If velocity increases dramatically the orbit becomes hyperbolic. If the velocity decreases significantly, the planet crashes into the sun. The position is the location of a planet given by its X, Y, and Z coordinates. There is a limited range of positions in which the orbits are stable and do not become hyperbolic or collide with the sun. The mass is defined as the mass of all the celestial bodies involved in the orbital system. If the mass of the planet is reduced, the orbit remains similar because of the large difference in mass between the sun and the planet. If the masses of the sun and the planet are equal, then they both orbit around one another. The default parameters in the program were chosen arbitrarily to define a stable orbit. Modeling planetary orbits is particularly interesting because in addition to utilizing numerical integration, it gave our team an opportunity to explore the realms of celestial mechanics.

Basic Orbital Parameters

Velocity	7
X-Y Direction	180
Z Direction	90
Mass	1
X- Coordinate	0
Y- Coordinate	100
Z- Coordinate	0
Time Step	1
Center Object Mass	4700
Run Length	1000

Analytic Solution

Although numerical integration is normally used to express solutions to problems without analytic solutions, in order to compare the accuracy of each numerical integration method, it is essential that the problem can be solved analytically as well as numerically. Using Kepler's mathematical laws, formulas found in Orbital Motion by A.E. Roy, and help from our mentor, Mark Smith, we created a program in C# that gave an accurate

solution to the ideal two-body problem. Below is a brief description of the analytic solution. The code for this section of the program may be found in Appendix I.

The first step in our analytic program begins with the calculations of the angular momentum, the mechanical energy, and the eccentricity vector, which are all conserved quantities within the orbital system. The eccentricity vector is defined as the distance from the center of the ellipse to the center of mass of the focus. The eccentricity vector's magnitude is proportional to the eccentricity. A planet's location vector is a vector drawn from the center of a planet's mass along the eccentricity vector. After calculating the location vector, we determine the true anomaly by calculating the angle between the eccentricity vector and the location vector as measured from the focus. Using the true anomaly, eccentricity vector, and the current location, we derive the semi-major axis "a" and the semi-minor axis "b". Using Kepler's third law we determine the period. According to the relationship described by Kepler's third law, the square root of the period is proportional to the cube root of the semi-major axis.

Next, we use the true anomaly to determine the eccentric anomaly. Using the eccentric anomaly and the true anomaly, we can derive the mean anomaly which we use to determine the time since perihelion passage. The perihelion passage is the closest point along the orbit to the center of mass (the Sun in our model). Using this information, we can determine the position and velocities of the planets at any point since perihelion passage. The point of perihelion passage is inline with the eccentricity vector. We can calculate the mean anomaly at a new time step because it is directly proportional to time. The eccentric anomaly can be determined by solving Kepler's equation.

Now we have the tools we need to begin calculating the exact position of the orbiting body. We calculate the radius of the ellipse using the true anomaly, eccentricity and semi major axis. Based on the true anomaly and radius, the position along the ellipse in the (x, y) coordinate system of the plane of the ellipse can be determined. This coordinate system has the eccentricity vector as its “x” axis. The angular momentum vector is represented on the positive “z” axis. The cross product of the eccentricity and the angular momentum is shown on the “y” axis. We determine the location of the planet in the standard (x, y, z) coordinate system by finding the rotation between the elliptic and absolute coordinate system. The magnitude of the velocity can be calculated using the mechanical energy. The angle between the velocity and position vector is a function of the eccentricity and the true anomaly. The angle between the velocity vector and the position vector is a function of the true anomaly. By subtracting the true anomaly from this angle we can find the x and y components of the velocity vector. Similarly to the location vector, the velocity vector can be converted into the standard “x” and “y” coordinate system. The solutions to the location and velocity vectors tell us exactly where the orbiting body is and how the body is moving at any given time.

Euler’s Method

Although Leonhard Euler’s explicit numerical integration scheme might seem a little dated today, when it was invented in the late 18th century, it revolutionized calculus. Euler’s estimations might seem poor compared to today’s standards, but we felt it was important to understand where and how numerical integration started. We implemented Euler’s Method first, as a sort of introduction to numerical integration. Next, we

modified the Euler Method slightly to create the Euler Cromer Method a modification that deal with specific ODEs. Below is a brief description of the logic used in Euler's Method.

Euler's method calculates the slope at a given point, and steps forward a given interval to determine a new point. It then repeats this process at the new point. The average slope between the initial point and the next point is changing with time unless the average slope is not equal to the slope of the initial point; thus, Euler's method will underestimate the change in the slope. In the case of the two body problem, we have a known ODE defining the acceleration, the second derivative of the position. This can be broken into two differential equations relating the position to its derivative, velocity, and the velocity to its derivative, acceleration. These coupled differential equations are solved simultaneously using Euler's method. Because the velocity effects the position, the new position is calculated before the new velocity. The code for this section of the program may be found in Appendix I.

Euler Cromer Method

After researching Euler and numerical integration, we discovered that there were many methods build on the original Euler Method. We decided to investigate the Euler Cromer Method, a symplectic integrator. Tom Cromer was an American mathematician who developed this method in the 1980s. A symplectic integrator is a numerical integration scheme that is designed to solve a specific set of ODEs. Euler Cromer can only be used with coupled differential equations and is more accurate than the Euler method for the

ideal two-body problem. The Euler-Cromer calculates the new points based on the slopes at the current location exactly like the Euler method; however it updates the velocity before it updates the position. This means the position is calculated based off of a later velocity increasing the method's accuracy. The code for this section of the program may be found in Appendix I. The equations for this method are below:

$$\begin{aligned}v_{n+1} &= v_n + g(t_n, x_n) \Delta t \\x_{n+1} &= x_n + f(t_n, v_{n+1}) \Delta t\end{aligned}$$

http://en.wikipedia.org/wiki/Euler-Cromer_algorithm

Leap Frog Method

Yet another method that builds on Euler is the Leap Frog Method. This method is commonly used as a replacement to Euler. This method calculates the new position based off the velocity and the acceleration at a given time and updates the velocity based on the next time step. This offsets the calculation of the position and velocity by a time step giving it its name as the positions and velocities “leap frog” past each other. The Leap Frog method only works with coupled differential equations. The code for this section of the program may be found in Appendix I. The general equations for the Leap Frog Method are listed below. The change in time is represented by h . y_{n+1} is the new point.

$$y_{n+1} = y_n + hf(x_n, y_n)$$

$$y_{n+1} = y_n + hf(x_n, y_n) + \frac{h^2}{2!}y''(x_n)$$

<http://www.cs.princeton.edu/introcs/94diffeq/>

The Runge-Kutta Methods

The Runge-Kutta methods were developed by German mathematicians Carl David Tolmé Runge and Martin Wilhelm Kutta in the late 19th century. This family of methods was more complex than Euler's Method and could be applied to a wide variety of problems, just like Euler's Method. The two most popular Runge-Kutta methods are Runge-Kutta Order 2 (RK2) and Runge-Kutta Order 4 (RK4). A second order method is one in which the error per time step is on the order of t^3 . A fourth order method is one in which the error per time step is on the order of t^5 . When the time step is less than one, the error decreases dramatically for the higher order method. However, when the time step is greater than one, the error is greater for a higher order method. The RK4 method is favored by scientists for this reason. Even though it runs slower than some methods, it is able to take longer steps, it is generally considered to be a stable, accurate method. Below is a brief description of both the RK2 and RK4 methods.

The Runge-Kutta 2 method (RK2), also known as the midpoint method, takes a tentative step half way between the current point and the point we are aiming for and uses this slope to calculate the next point. The slope from the midpoint will almost always be closer to the average slope between the current point and the next point. The code for this section of the program may be found in Appendix I. The formulas for Runge-Kutta 2 are listed below. The change in time is represented by h . K_1 is the slope at the initial point. K_2 is the slope at a half time step. Y_{n+1} is the new coordinate.

$$\begin{aligned} k_1 &= h f(x_n, y_n) \\ k_2 &= h f\left(x_n + \frac{1}{2} h, y_n + \frac{1}{2} k_1\right) \\ y_{n+1} &= y_n + k_2 + O(h^3), \end{aligned}$$

Note: This formula table was generated by Wolfram Math-World:

<http://mathworld.wolfram.com/Runge-KuttaMethod.html>

The Runge-Kutta 4 (RK4) uses a total of four tentative steps, K_1 - K_4 , to calculate a slope based on a weighted average. K_1 , K_2 , and Y_{n+1} all represent the same quantities as above. K_3 is also a half time step, but determined using the K_2 slope. K_4 is the slope a full time step using the K_3 . The code for this section of the program may be found in Appendix I. The equations for calculating K_1 - K_4 are as follows:

$$\begin{aligned} k_1 &= h f(x_n, y_n) \\ k_2 &= h f\left(x_n + \frac{1}{2} h, y_n + \frac{1}{2} k_1\right) \\ k_3 &= h f\left(x_n + \frac{1}{2} h, y_n + \frac{1}{2} k_2\right) \\ k_4 &= h f(x_n + h, y_n + k_3) \\ y_{n+1} &= y_n + \frac{1}{6} k_1 + \frac{1}{3} k_2 + \frac{1}{3} k_3 + \frac{1}{6} k_4 + O(h^5) \end{aligned}$$

Note: This formula table was generated by Wolfram Math-World:

<http://mathworld.wolfram.com/Runge-KuttaMethod.html>

Gill's Method

Gill's Method is an explicit numerical integration scheme very similar to Runge-Kutta. It was developed by British mathematician Richard D. Gill in the 1970s. Gill's Method takes tentative steps in a manner nearly identical to the Runge-Kutta 4. The major difference is the weighting system for the slopes and the calculation of the K_3 and K_4 steps based on all the previous slopes. The code for this section of the program may be found in Appendix I. The equations are as follows:

$$\begin{aligned} k_1 &= h f(x_n, y_n) \\ k_2 &= h f\left(x_n + \frac{1}{2} h, y_n + \frac{1}{2} k_1\right) \\ k_3 &= h f\left[x_n + \frac{1}{2} h, y_n + \frac{1}{2} (-1 + \sqrt{2}) k_1 + \left(1 - \frac{1}{2} \sqrt{2}\right) k_2\right] \end{aligned}$$

$$k_4 = h f \left[x_n + h, y_n - \frac{1}{2} \sqrt{2} k_2 + \left(1 + \frac{1}{2} \sqrt{2} \right) k_3 \right].$$

<http://mathworld.wolfram.com/GillsMethod.html>

Modified RK2

In addition to implementing numerical integration methods that we researched online and in print sources, we implemented our own version of the RK2 method. Our modified RK2 works in almost the exact same manner as the RK2 method; however, it calculates the k_1 change in velocity based on the position from $\frac{1}{2}$ time step ahead. This means the final positions and velocities are determined based on the slope at an estimated point one time step ahead. The formulas we designed are listed below. The code for this section of the program may be found in Appendix I.

$$\begin{aligned} K_1 x &= f(t, v) dt \\ K_1 v &= g \left(t + \frac{1}{2} dt, x + \frac{1}{2} K_1 x \right) dt \\ K_2 x &= f \left(t + \frac{1}{2} dt, v + \frac{1}{2} K_1 v \right) dt \\ K_2 v &= g \left(t + \frac{1}{2} dt, x + \frac{1}{2} K_1 x \right) dt \\ V_{n+1} &= V_n + K_2 v \\ X_{n+1} &= X_n + K_2 x \end{aligned}$$

Adams-Bashford Method

Unlike Euler's Method or Runge-Kutta, the Adams -Bashford Method was designed to solve a very specific problem. In 1883, British chemist Francis Bashford approached John Couch Adams with a set of differential equations that modeled capillary action, or the ability of a substance to draw another substance into it. Adams, a prominent British mathematician and the discoverer of Neptune, designed a numerical integration scheme that estimated a solution to Bashford's differential equations, and helped model the flow of liquid in a capillary tube.










The Adams-Bashford Method is a type of linear multi-step method for solving Ordinary Differential Equations. Linear multi-step methods use linear combinations of slopes at previous points to calculate the next point. These methods are not self-starting unless a sufficient number of points are already known. We chose to initialize our Adams-Bashford Method with the Runge-Kutta 4 because of its demonstrated accuracy in our tests and its wide usage. The code for this section of the program may be found in Appendix I. The formula for the Adams-Bashford follows:

$$y_{n+4} = y_{n+3} + h \left(\frac{55}{24} f(t_{n+3}, y_{n+3}) - \frac{59}{24} f(t_{n+2}, y_{n+2}) + \frac{37}{24} f(t_{n+1}, y_{n+1}) - \frac{3}{8} f(t_n, y_n) \right) .$$

http://en.wikipedia.org/wiki/Linear_multistep_method

Part II: Results and Conclusions

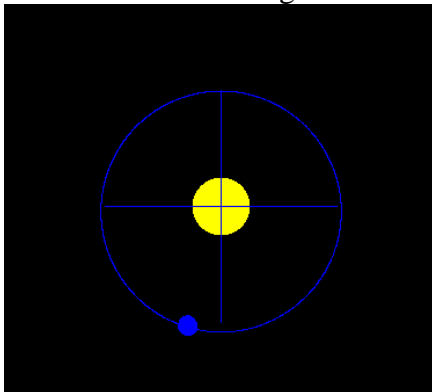
Key

	Analytic Solution
	Euler's Method
	Euler-Cromer Method
	Runge-Kutta 2 Method
	Runge-Kutta 4 Method
	Adams-Bashford Method
	Gill's Method
	Leapfrog Method
	Modified RK2

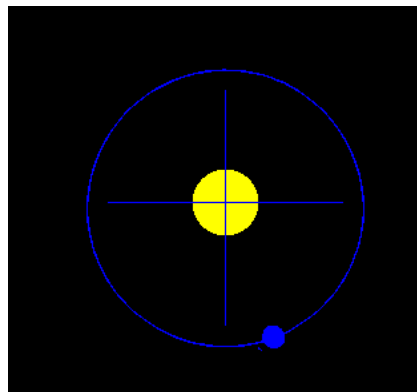
Stability in Time Step

Analytic Solution

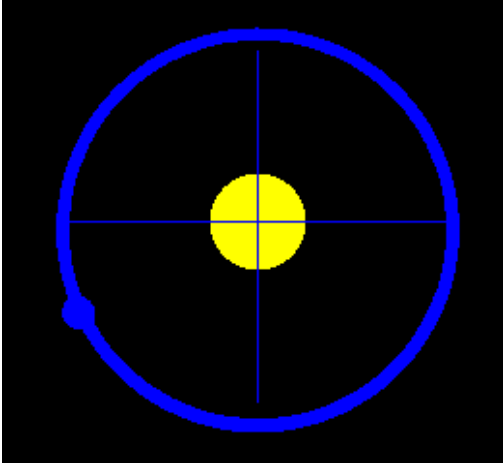
Screenshots from Program:



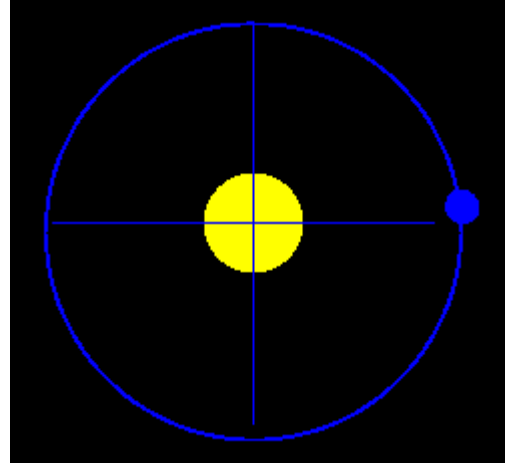
Time Step: 0.1



Time Step: 1



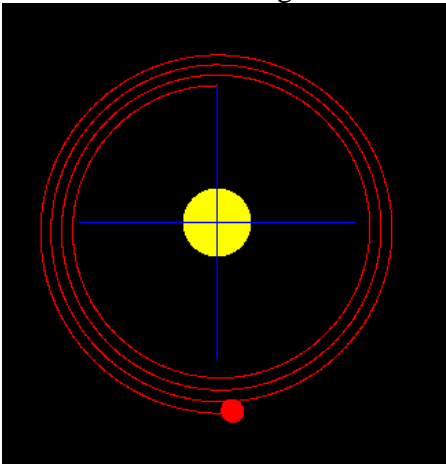
Time Step: 10



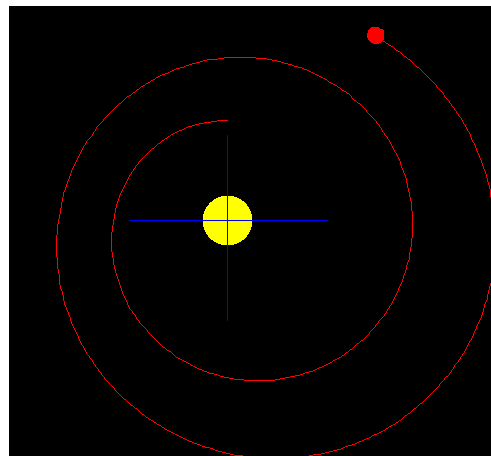
Time Step: 100

Euler's Method

Screenshots from Program:

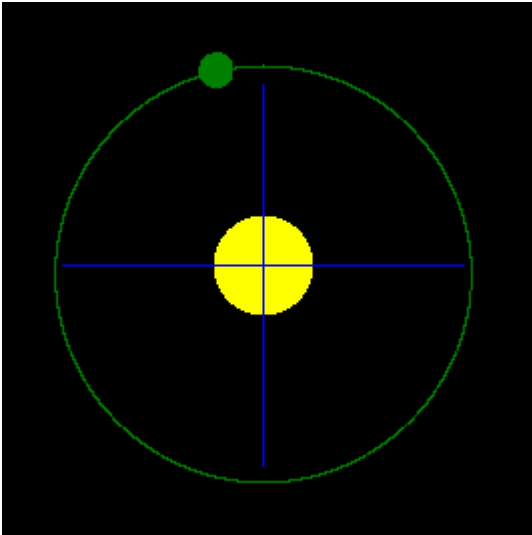


Time Step: .1

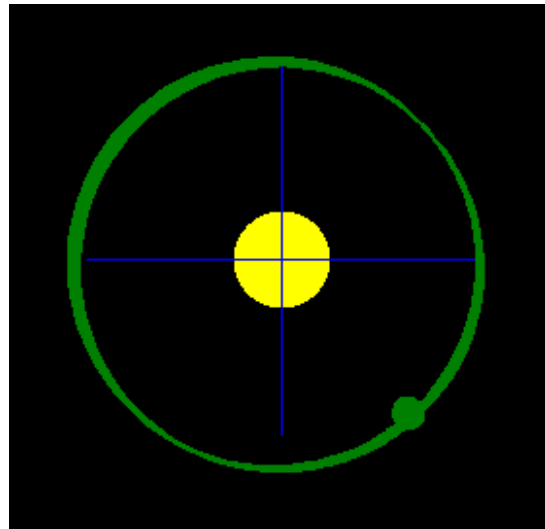


Time Step: 1

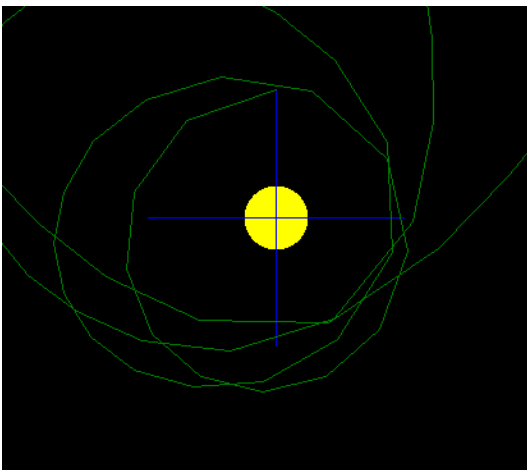
Runge-Kutta 2



Time Step: .1

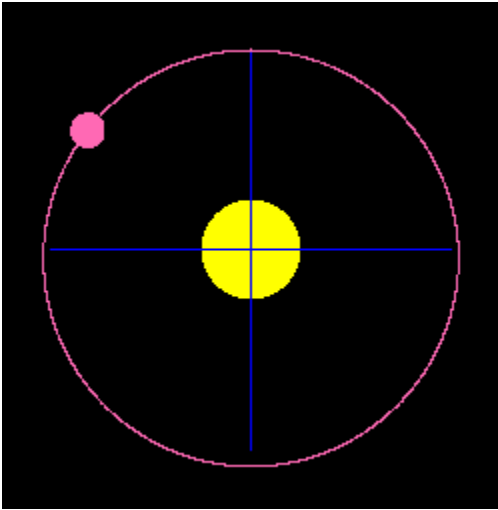


Time Step: 1

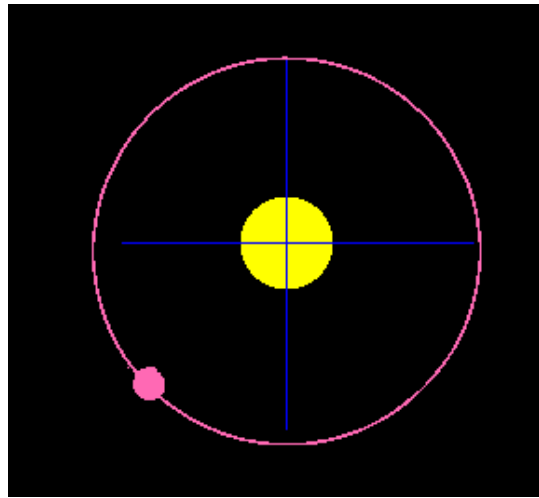


Time Step: 10

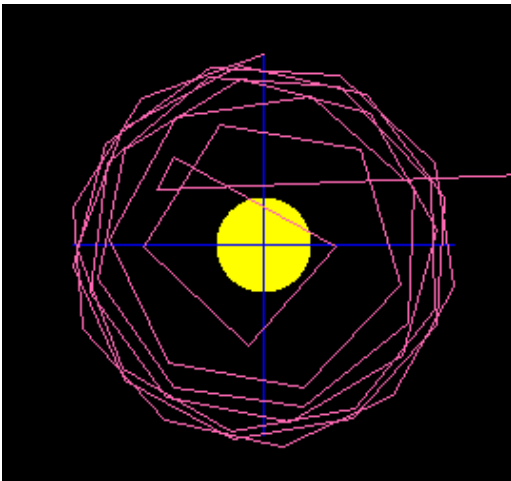
Runge-Kutta 4



Time Step: .1

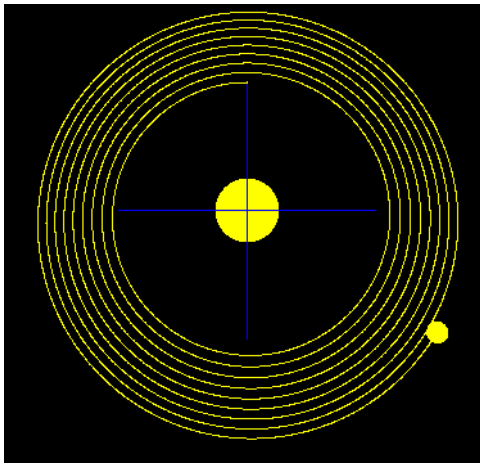


Time Step: 1

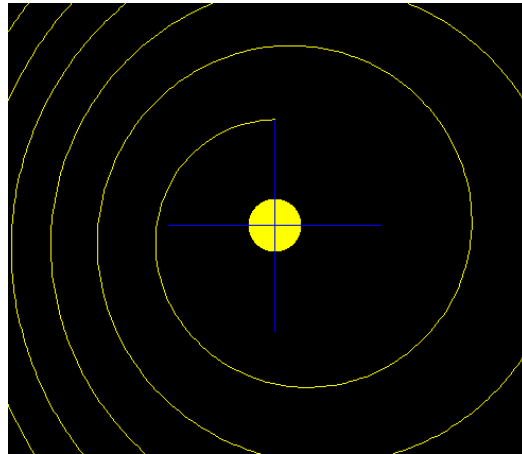


Time Step: 10

Adams-Bashford

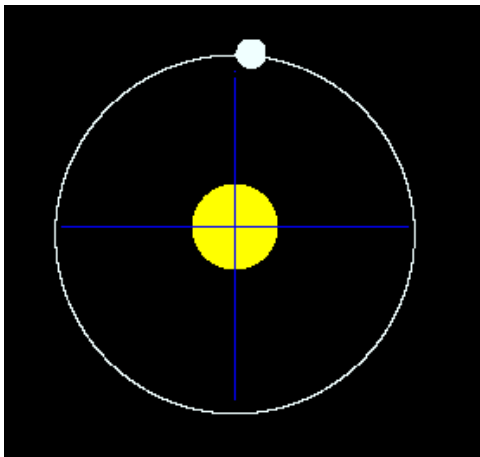


Time Step: .1

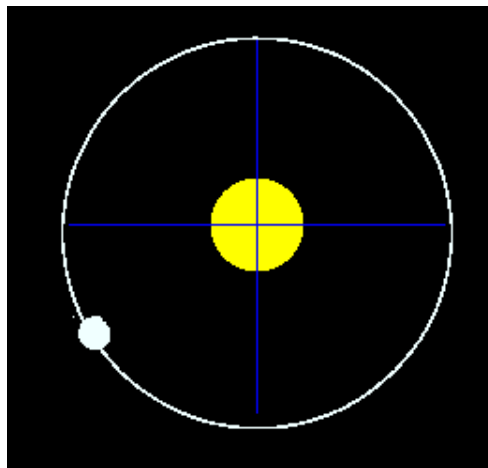


Time Step: 1

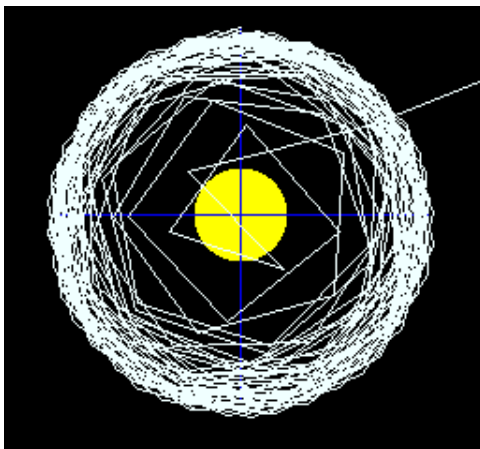
Gill's Method



Time Step: .1

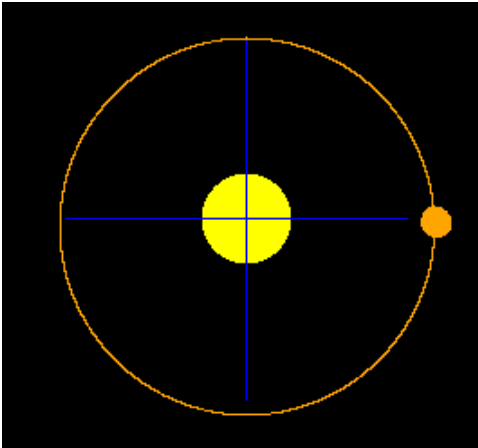


Time Step: 1

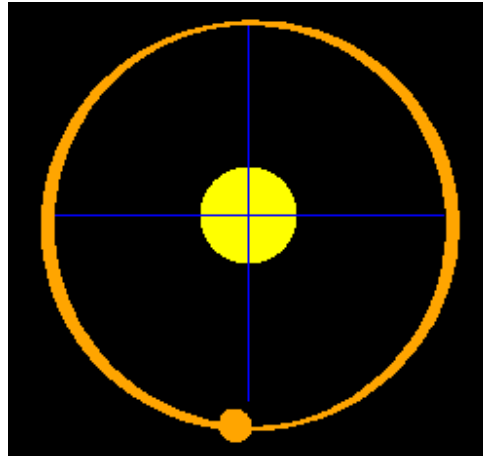


Time Step: 10

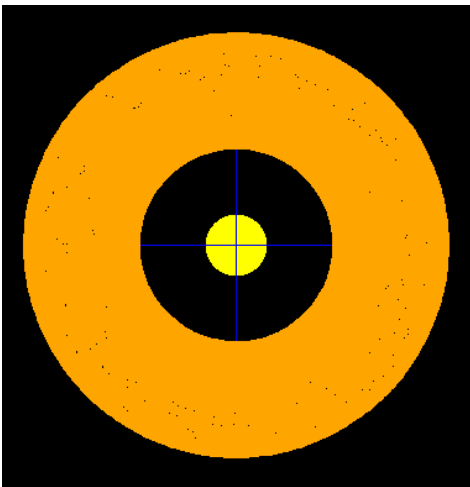
Euler Cromer Method



Time Step: .1

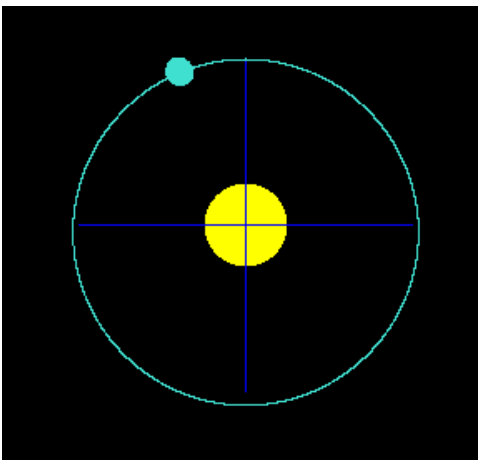


Time Step: 1

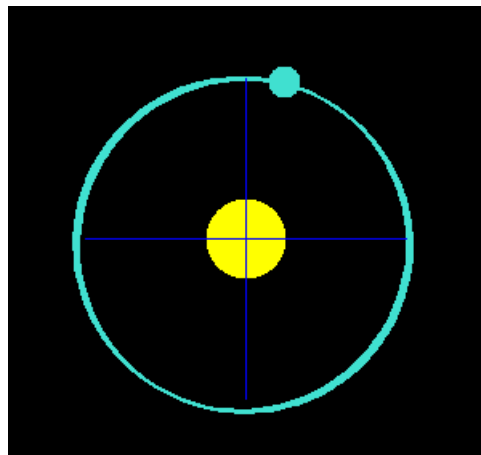


Time Step: 10

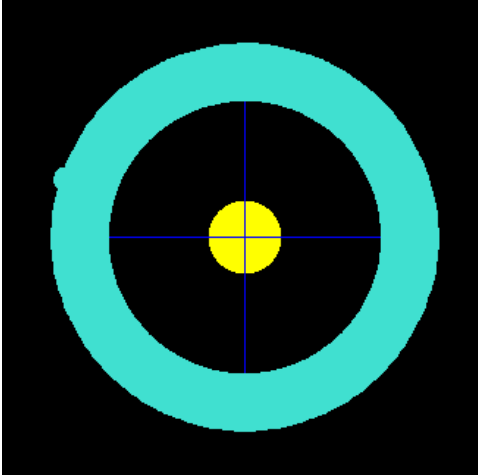
Leapfrog Method



Time Step: .1

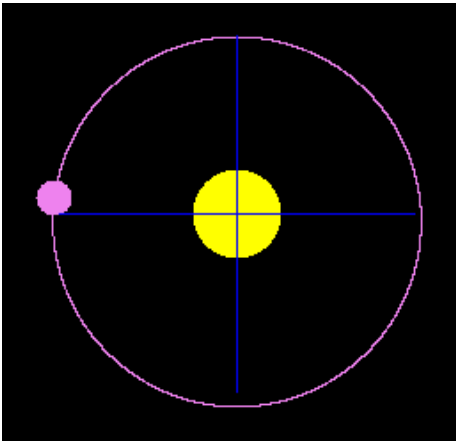


Time Step: 1

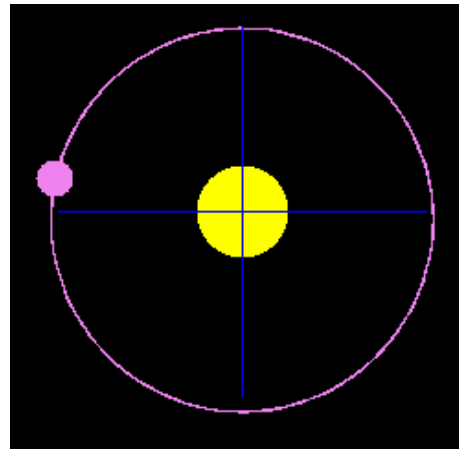


Time Step: 10

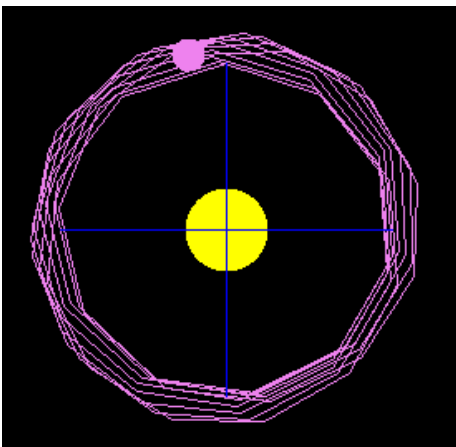
Modified Runge-Kutta 2



Time Step: .1



Time Step: 1



Time Step: 10

Conclusions

The differing time steps do not affect the analytic solution. In addition, the orbit does not change with increasing number of revolutions. The analytic solution decomposes the position and velocity vectors into separate elements which define the ellipse. Based on the current time, the positions and velocities are calculated from these orbital elements. This stability in time step is essential to an analytic solution and is critical to the evaluation of other numerical methods.

For small time steps the Euler's method is moderately accurate. This method seems to accumulate error in the conserved quantities along with the position and velocity linearly. As time steps increase the error also increases, eventually evidently showing sinusoidal error in position.

Like Euler's Method, Runge-Kutta 2, is more stable when the time step is shorter. Although it is not as unstable as Euler at a time step of 1, when we increase the time step by a factor of 10, it is apparent that RK2 becomes unstable. Logically, we assumed this would happen because of the nature of RK2. In a system with a rapidly changing slope, we expect large steps will not accurately measure the change.

Although the Runge-Kutta 4 is more stable than most methods, including Runge-Kutta 2, it also becomes unstable when the time step increases.

Like Euler, the Adams-Bashford method becomes unstable very quickly. The Adams-Bashford method is probably unstable in this program because it was specifically designed to solve ODEs associated with capillary action, not celestial mechanics.

Gill's method was very stable as the time step increased. Although it did become unstable at time step 10, it appeared to retain a controlled orbit longer than the RK4.

The Modified Runge-Kutta 2 was very stable even when the time step increased to ten. It was less stable than Euler Cromer and Leapfrog, but it was more stable than Gill's method.

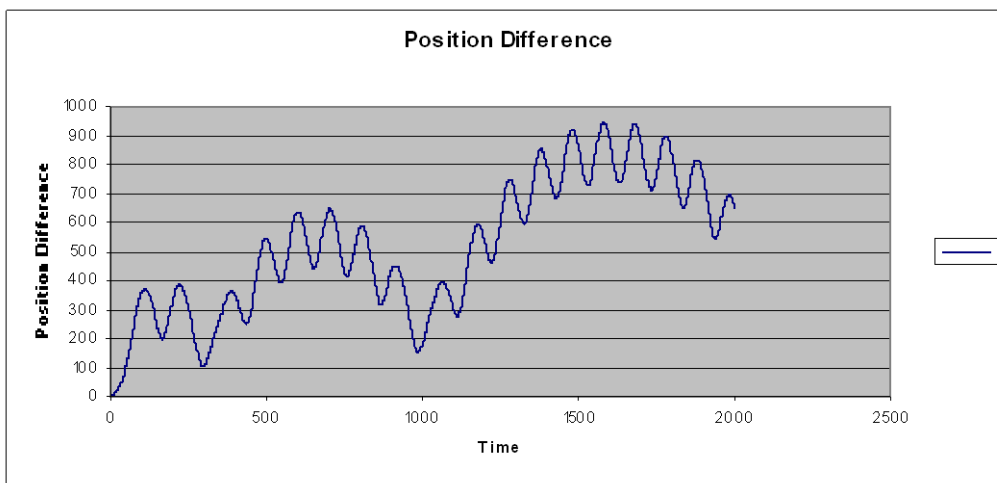
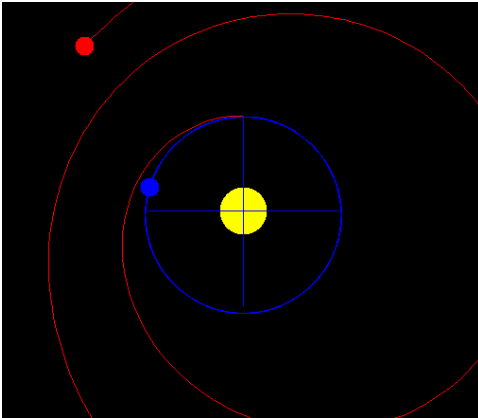
Although both Euler Cromer and Leapfrog gained a fair amount of energy as the time step increased, their orbits remained concentric and smooth. These two methods appeared to be the most stable as the time step increased.

Based on these results, we can conclude that all numerical integration methods are more accurate when the time step is small. All of the methods displayed similar tendencies when the time step increased.

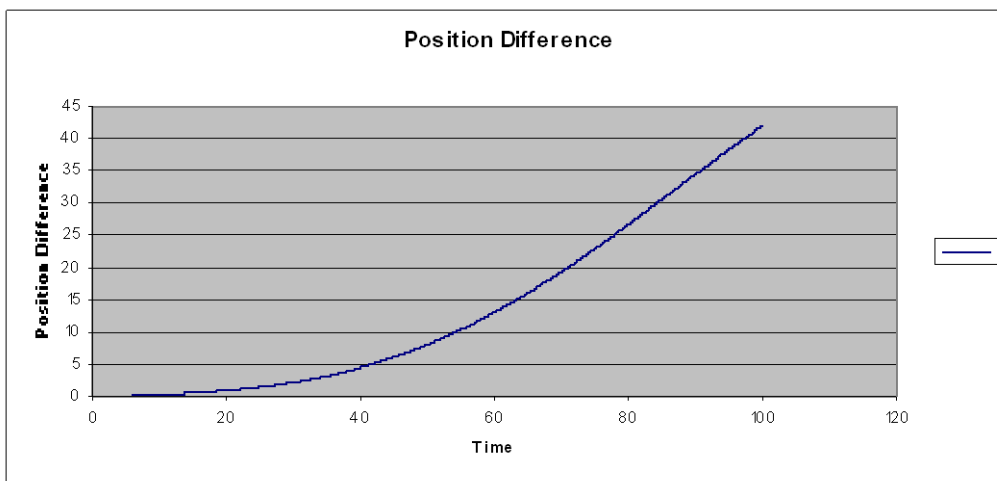
In-Depth Analysis and Comparison to Analytic Solution

The next eight sections compare each method to the analytic solution. The first image is a screenshot of the graphic output of the program in each case. Underneath the screenshot, we have included several graphs that help demonstrate numerically the stability and accuracy of each method in terms of energy, angular momentum, eccentricity, and position. All of the screenshots are from runs with a time step 2. The graphs were created from data with a run length of 2000. The time step for each graph is labeled in the lower left corner.

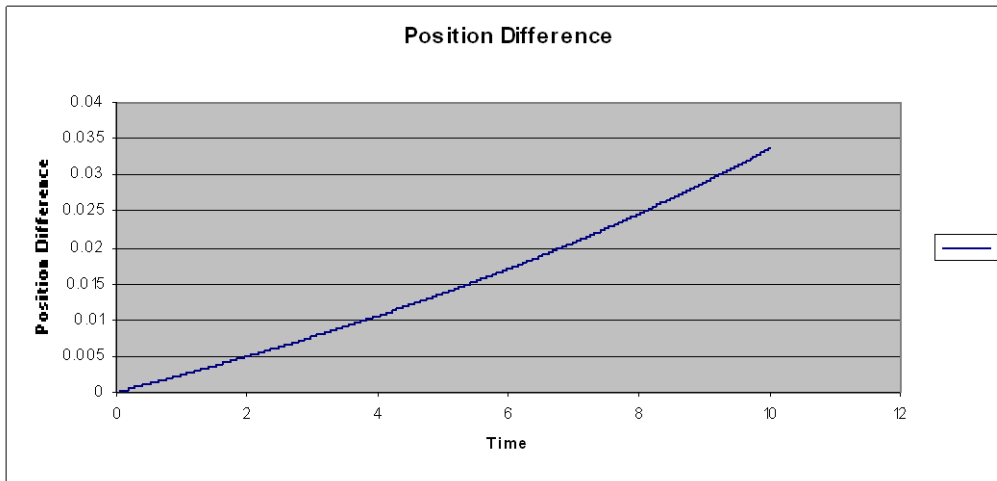
Analytic v. Euler



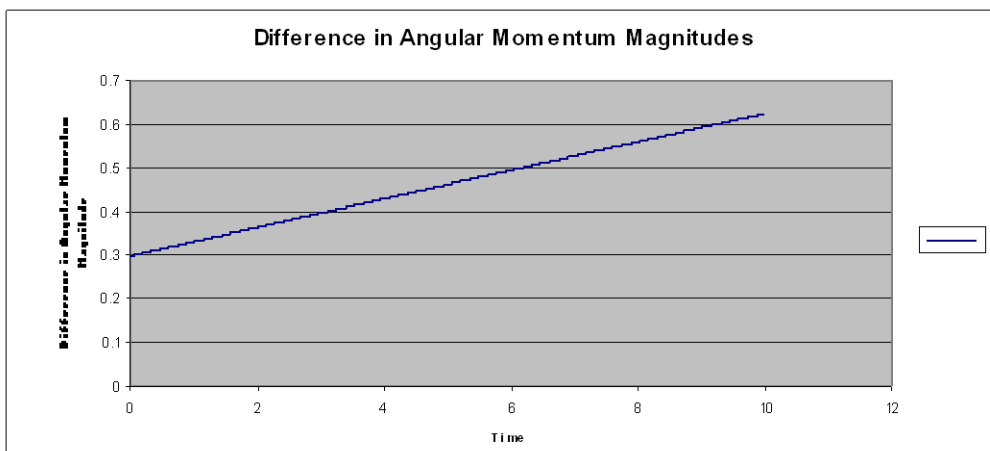
Time step: 2



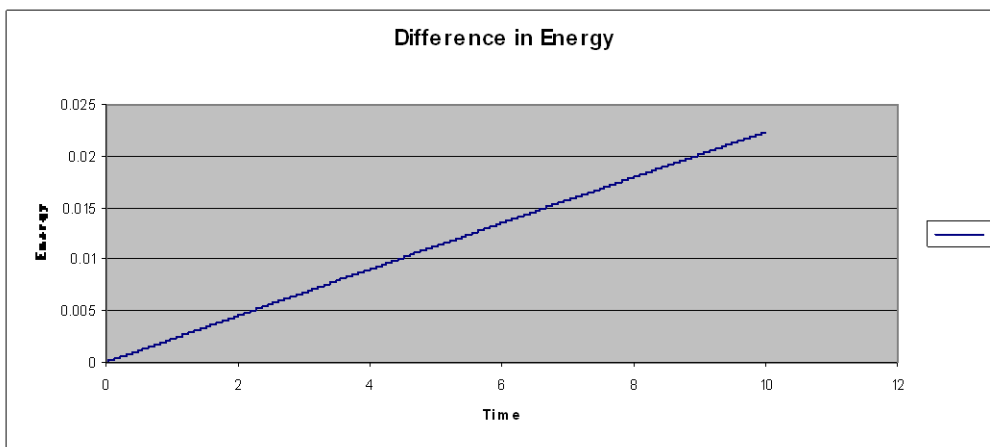
Time step: .1



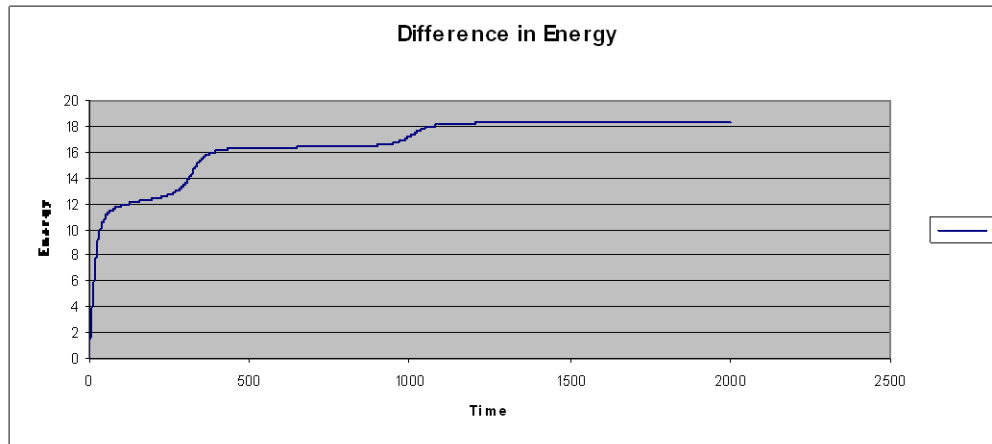
Time step: 0.01



Time step: 0.01



Time step: 0.01

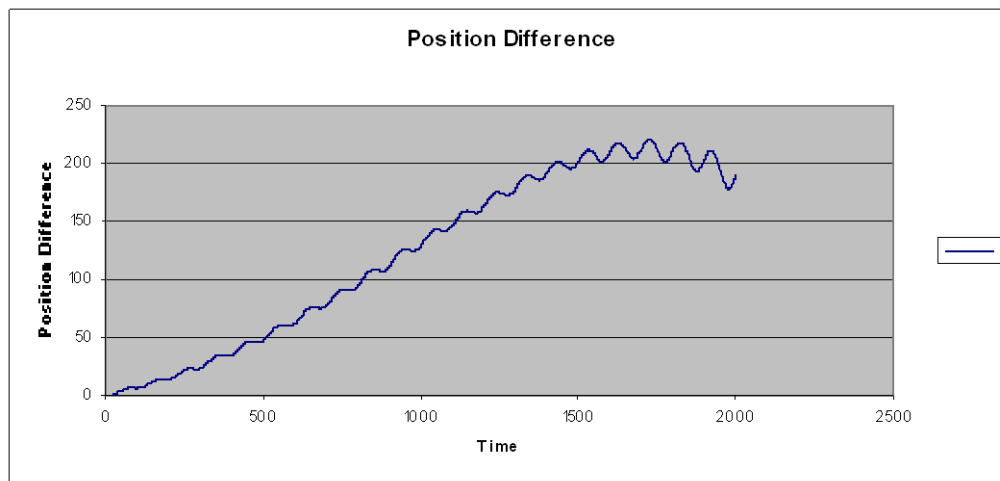
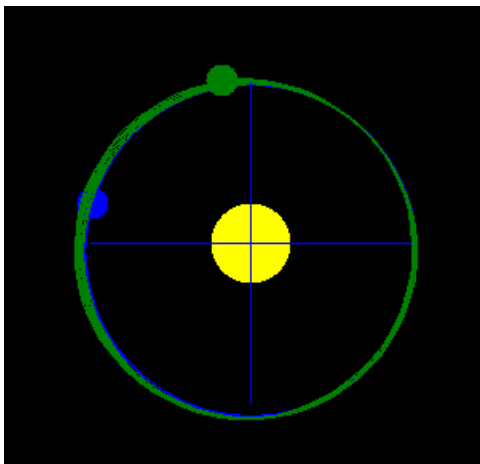


Time step: 2

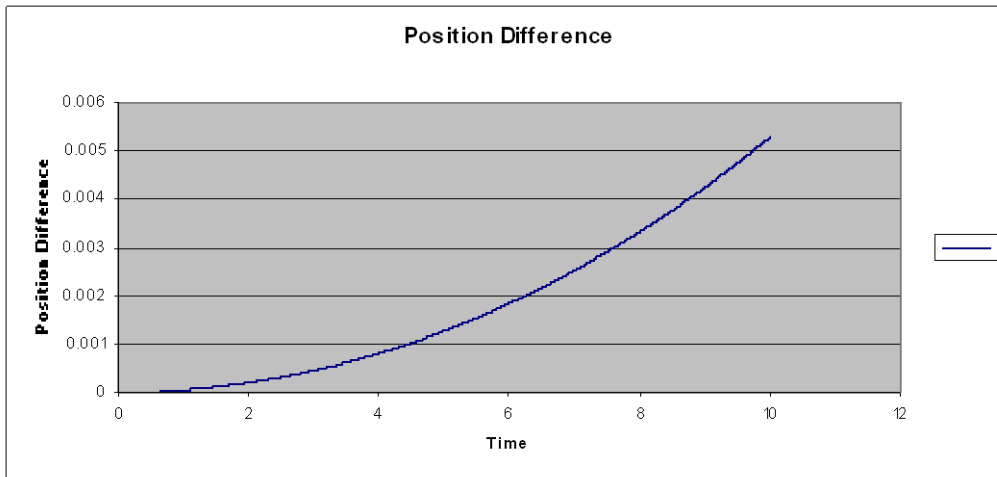
Both of the above graphs illustrate the inaccuracy of Euler's method compared to the analytic solution. To create the position difference graph, we took the difference between the x, y, and z coordinates of Euler and the analytic solution. In these runs the z coordinates are 0 for ease of analysis. We then took the magnitude of this distance, giving the spatial distance between the calculated positions of the two planets. The Euler performs fairly accurately with very small time steps with a gradual accumulation in error. If the time step is increased, the accumulation becomes exponential and very rapid. With increasing time steps, chaotic and oscillatory behavior can be observed in the error in position graphs. As the planets travel about their elliptical orbits, the velocity of the planets and the curvature of the ellipses are constantly changing. Consequently, along certain parts of the ellipse the trailing planet is able to "catch up" before the other planet completes its curve. The large oscillations in the position difference graph occur because the analytic solution gets further along a revolution than the Euler's Method. Then, as the analytic solution becomes a revolution ahead of the Euler's method, the two solutions approach a closer point in the revolution; thus coming closer together. The difference in angular momentum was found by subtracting the angular momentum of Euler's from the

angular momentum in the analytic solution. Angular momentum should be a conserved quantity in this system; however, the graph clearly indicates that the system is gaining angular momentum. Further analysis of other quantities showed that Euler was not conserving energy, angular momentum, or the eccentricity vector. These quantities to behave dynamically as time step increases culminating in huge upward spikes followed by stable regions.

Analytic v. Runge-Kutta 2



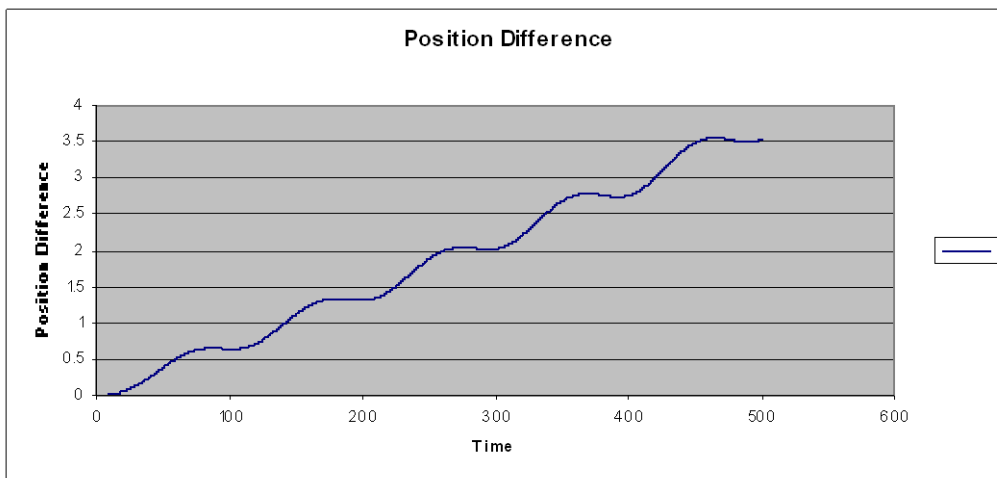
Time step: 2



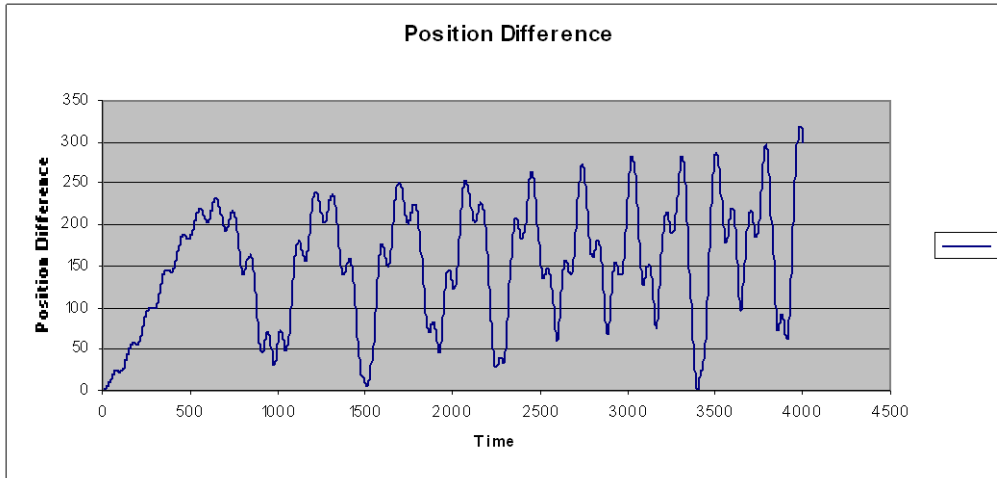
Time step: .01



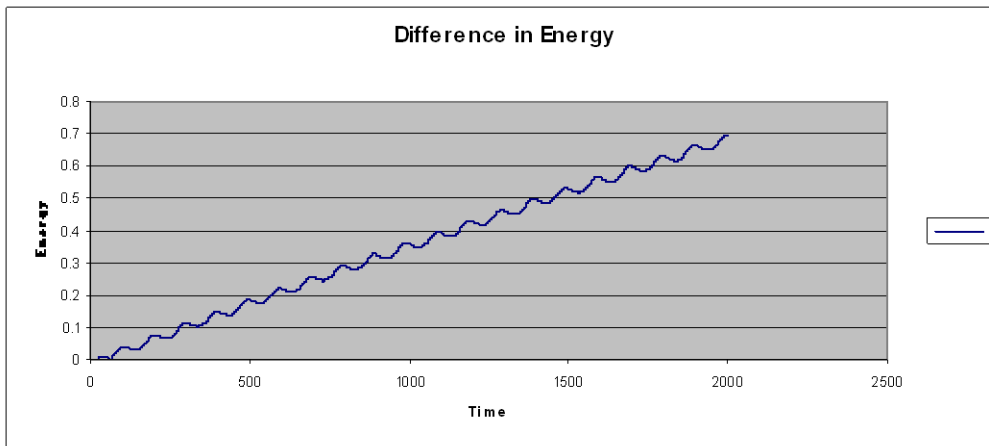
Time step: .1



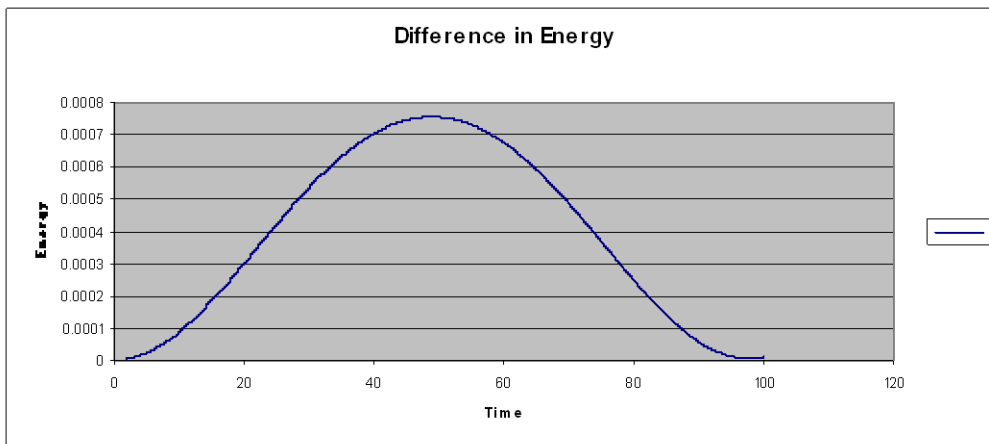
Time step: .5



Time step: 4



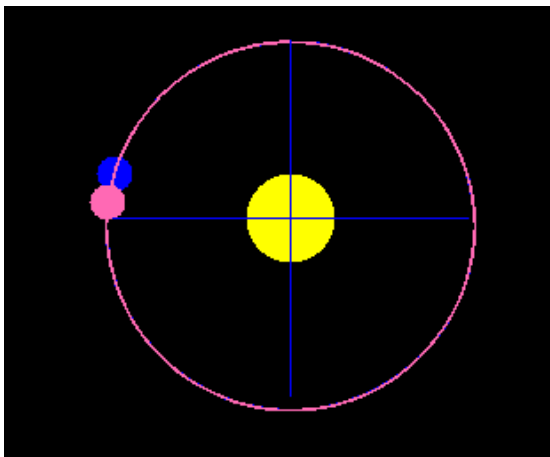
Time step: 2

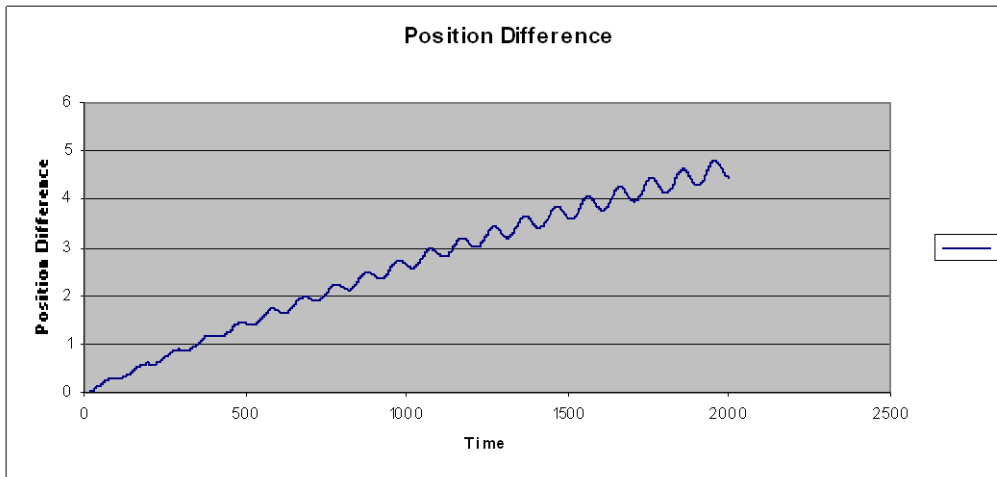


Time step: .1

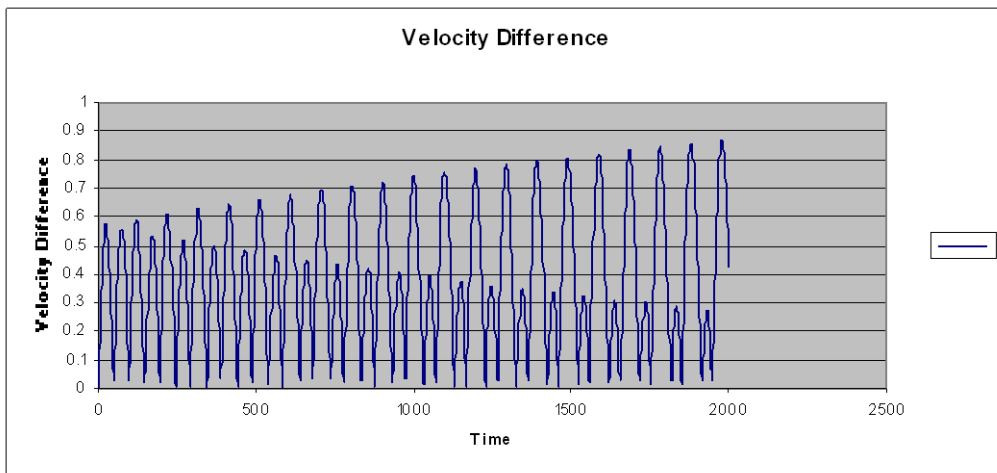
For very small time steps we observed the RK2 performing similar to the Euler; however, when the time step was increased the position error exhibited sigmoidal behavior and with further increase, gathered error approximately linearly with time. Thus, time steps in the range of ten times greater than the Euler's can be used while preserving a reasonable degree of accuracy. However when we increase the time step to 2 and compare RK2 to the analytic solution, it becomes unstable and accumulates error very rapidly. As with Euler, large oscillations between the differences of magnitudes occurred with the RK2. The graph of the energy difference was created in the same way as the graph of the angular momentum difference. The general upward trend of the energy graph reveals that the RK2 gains energy fairly quickly. All of the other conserved quantities increased as well. Like Euler, further analysis reveals this method to be fairly inaccurate.

Analytic v. Runge-Kutta 4

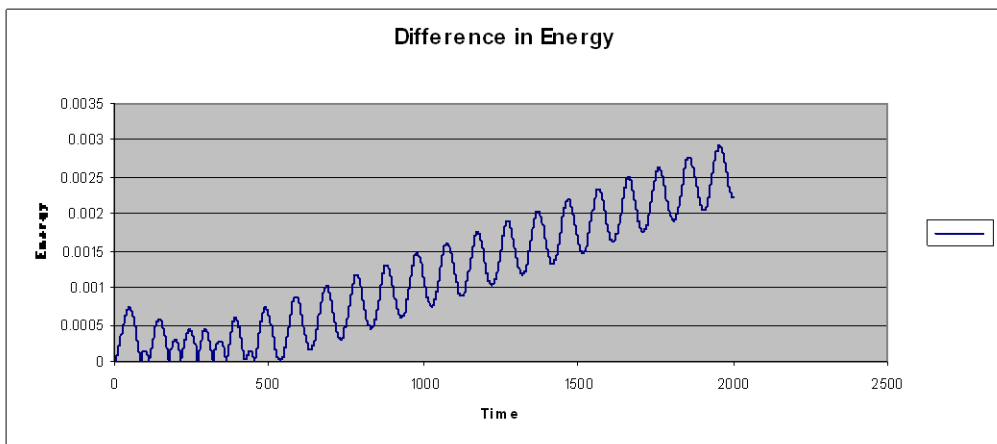




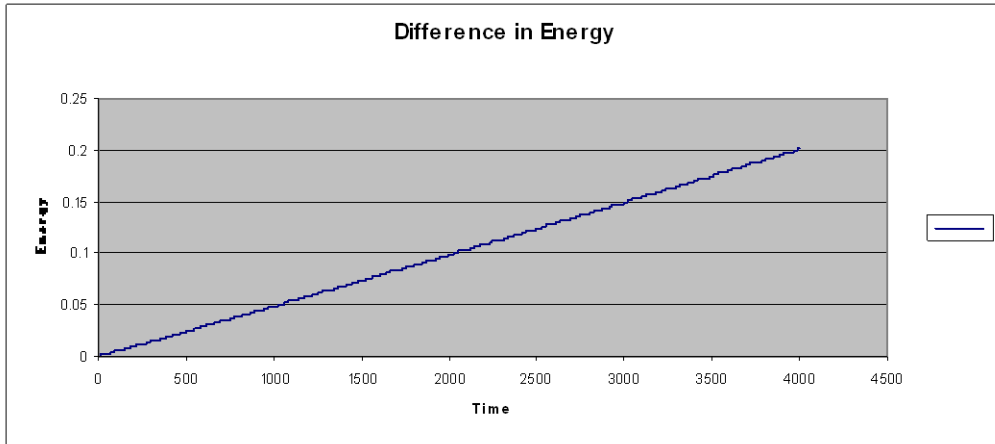
Time step: 2



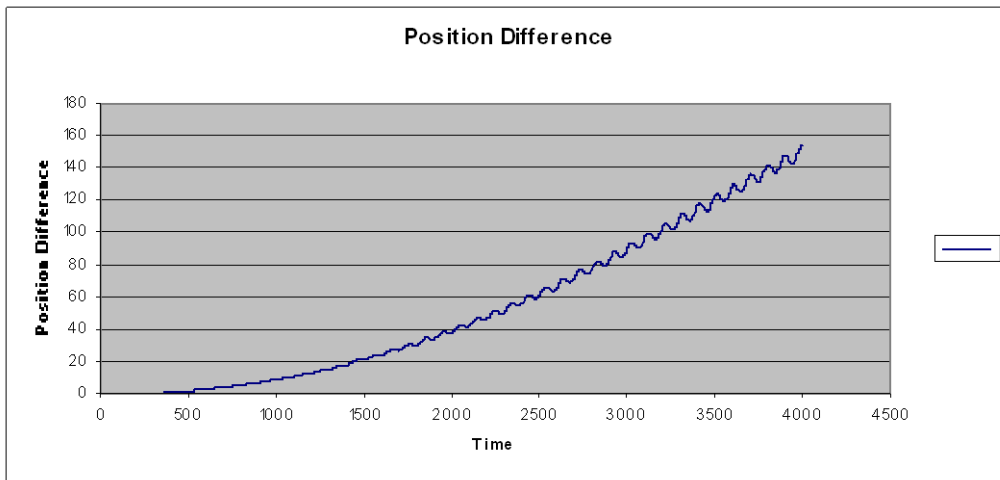
Time step: 2



Time step: 2



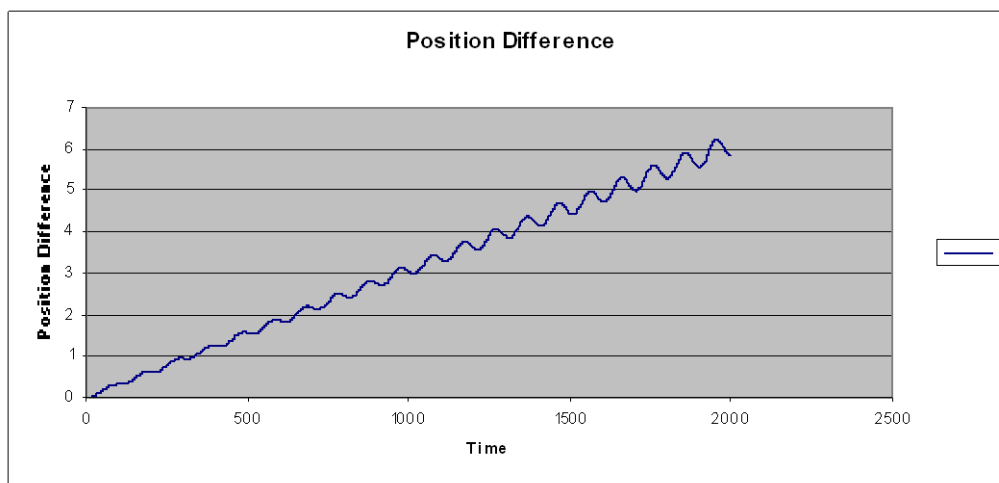
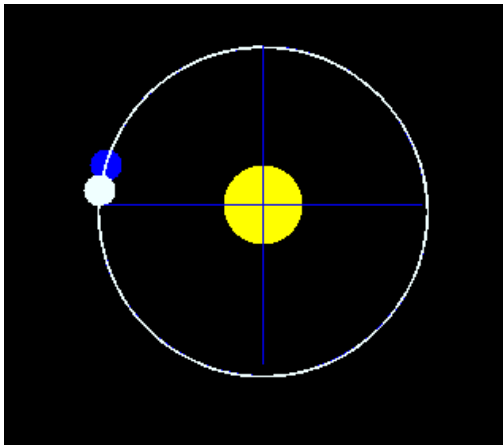
Time step: 4



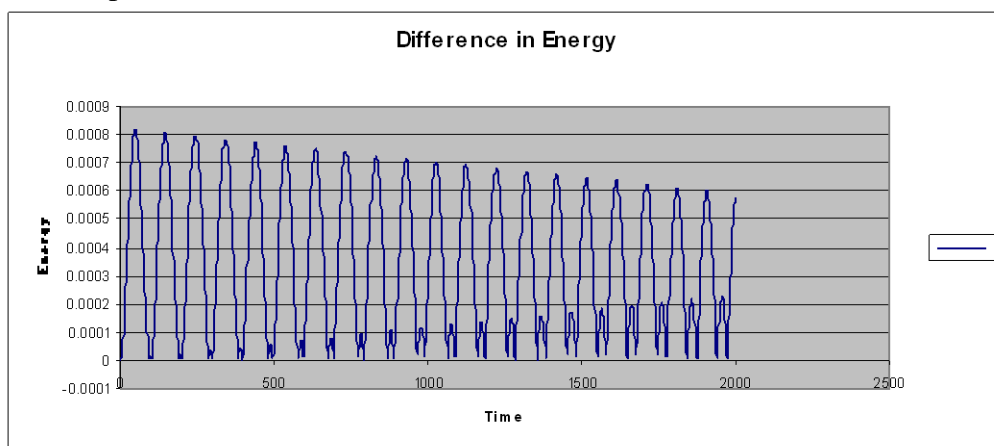
Time step: 4

The behavior of the graphs of the differences between the RK4 and the Analytic behave almost identically to those of the RK2 verses Analytic, although they RK4 is clearly more accurate with much higher time steps. The RK4 did not accumulate substantial error until our run with a time step of 4, where it began spiraling inward and moving ahead of the analytic. RK4 is certainly more accurate and stable than RK2, and although it is not the most accurate numerical integration scheme in this case, it is a very versatile method and can be applied to a wide variety of ODEs.

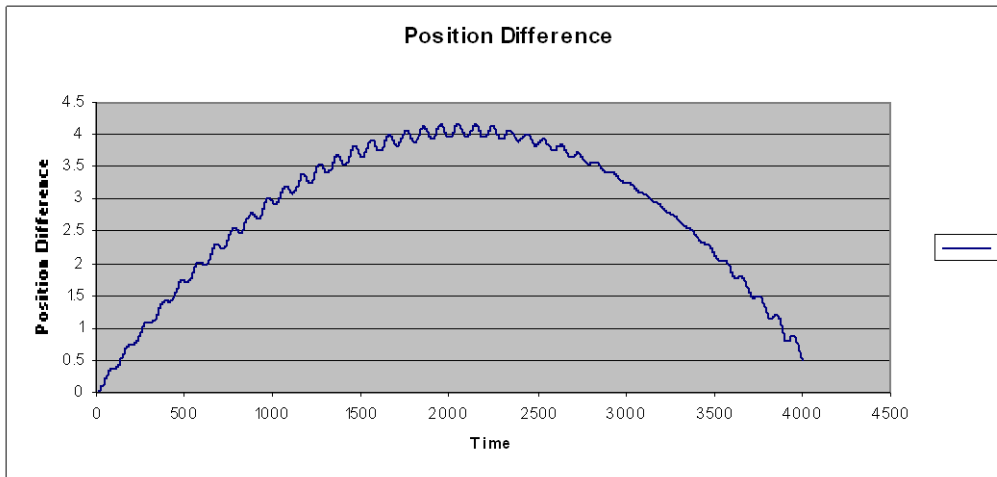
Analytic v. Gill's



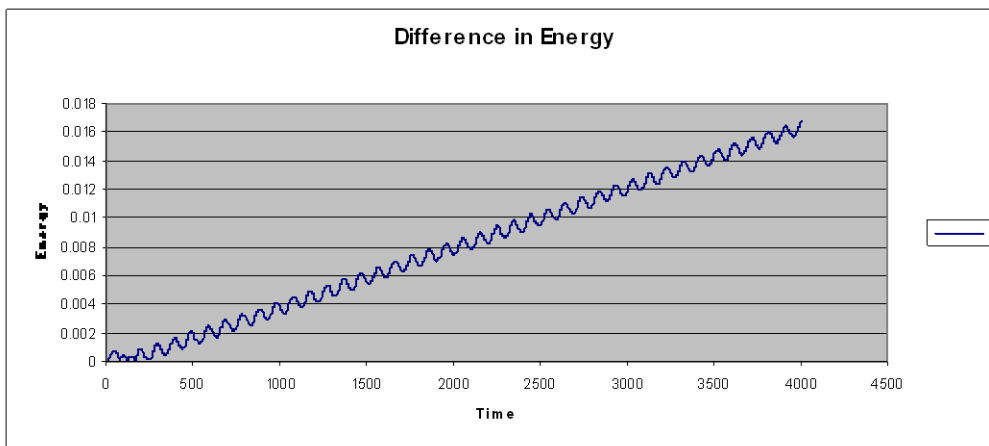
Time step: 2



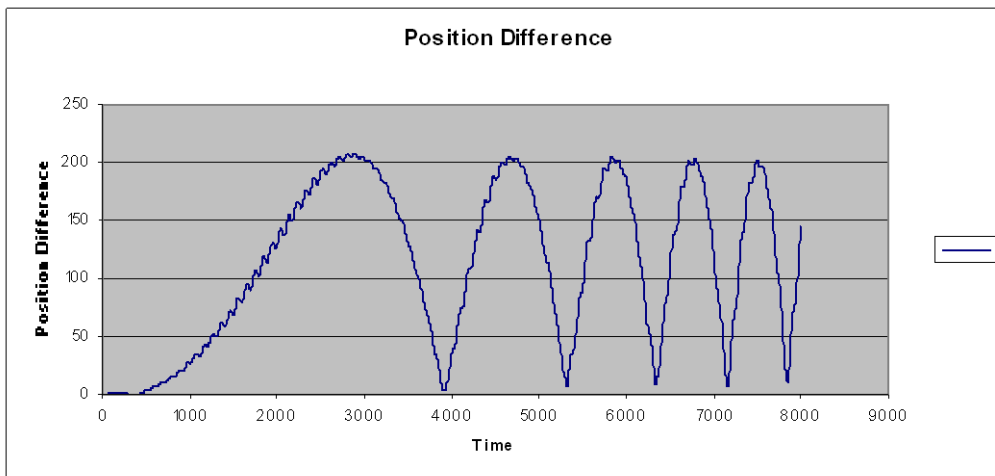
Time step: 2



Time step: 4



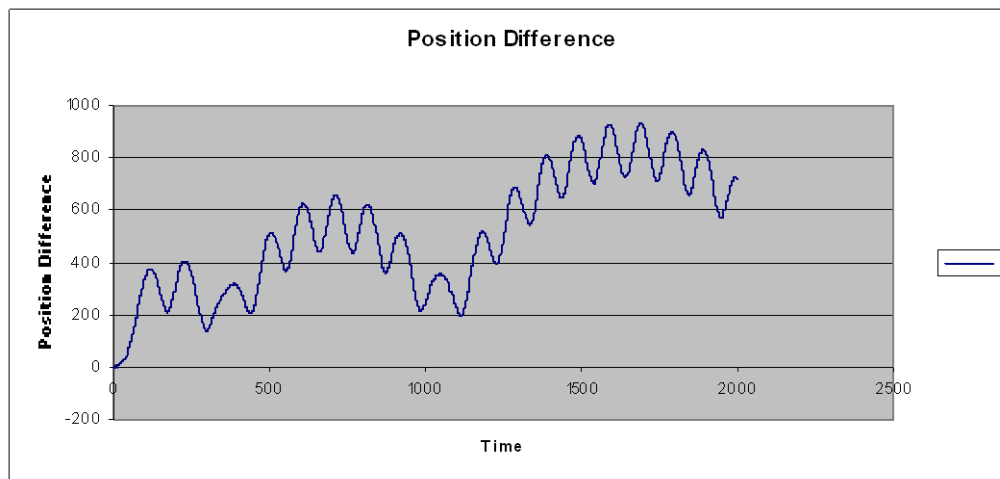
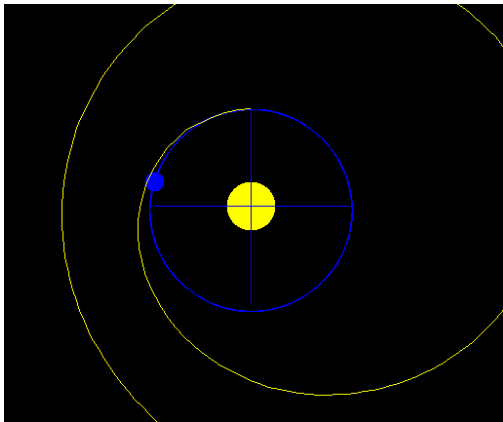
Time step: 4



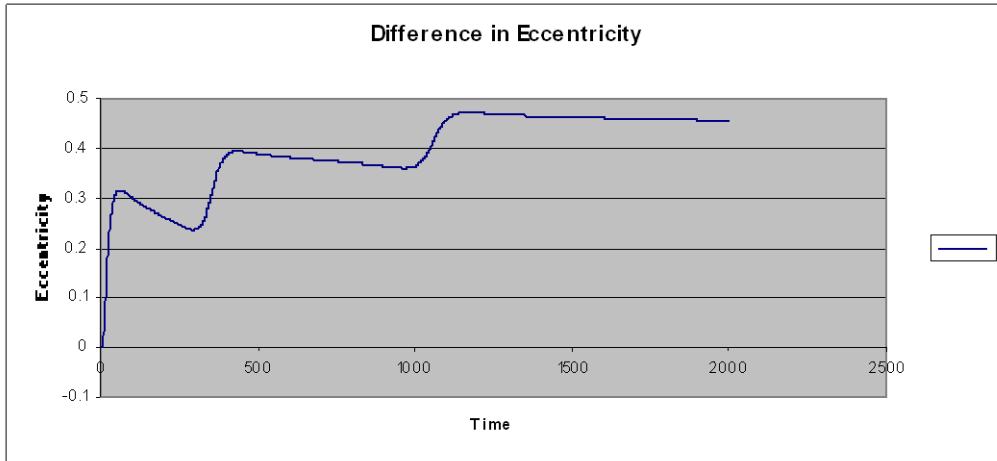
Time step: 8

Gill's Method behaves much in the same manner as RK4. Like RK4, it gradually spirals inward. Although energy loss is not visible on this graph, the system probably lost a minuet amount of energy as the orbit began spiraling inward. With small time steps, Gill's Method performs comparably or slightly worse than the RK4; however, it is significantly more stable at higher time steps and appears to have minimal error even up to a time step of four.

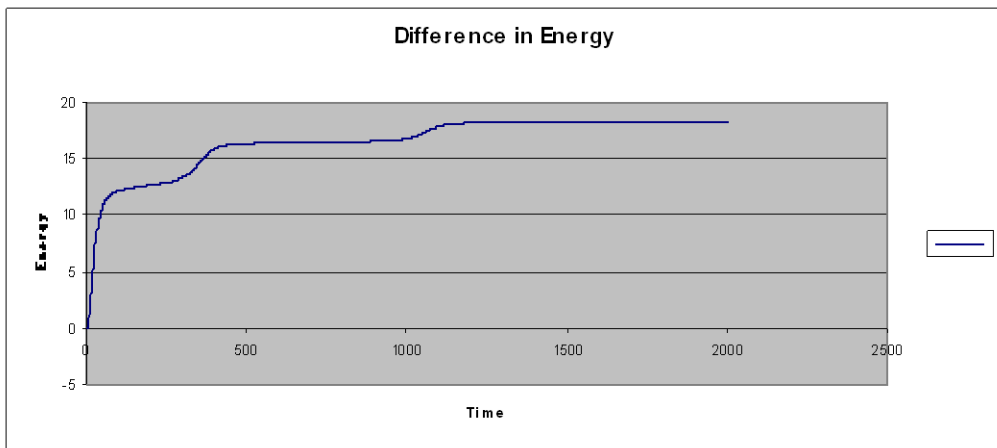
Analytic v. Adams-Bashford



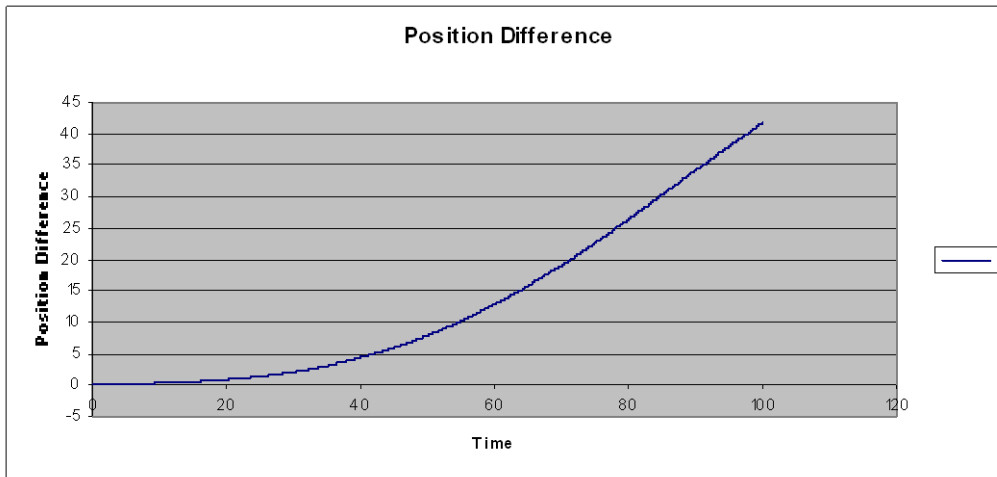
Time step: 2



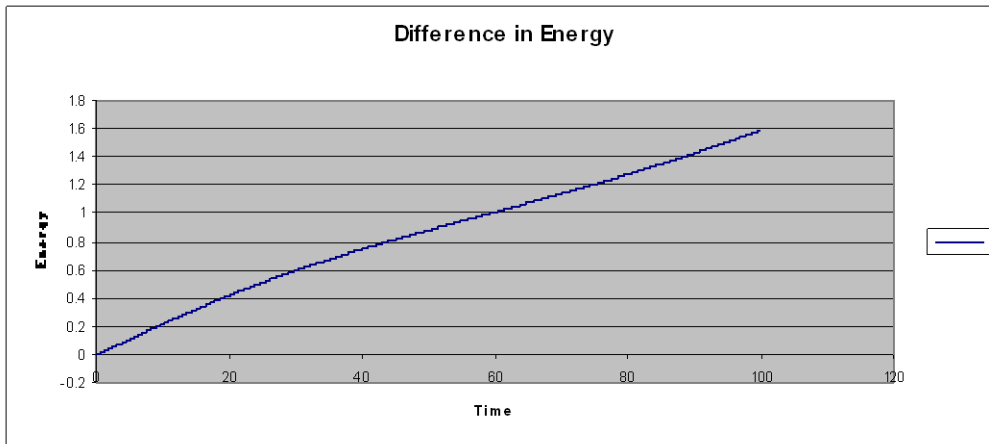
Time step: 2



Time step: 2



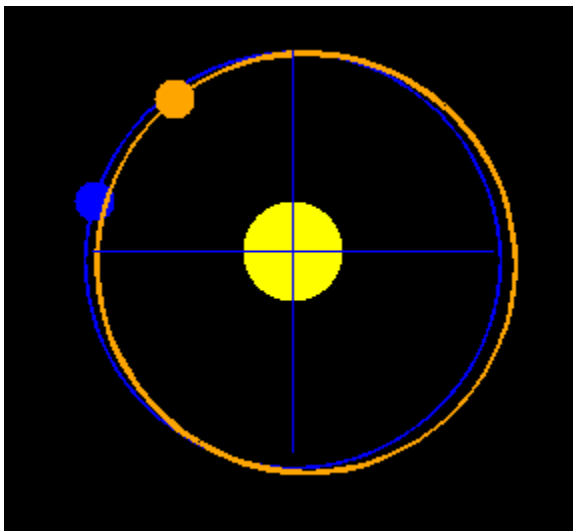
Time step: .1

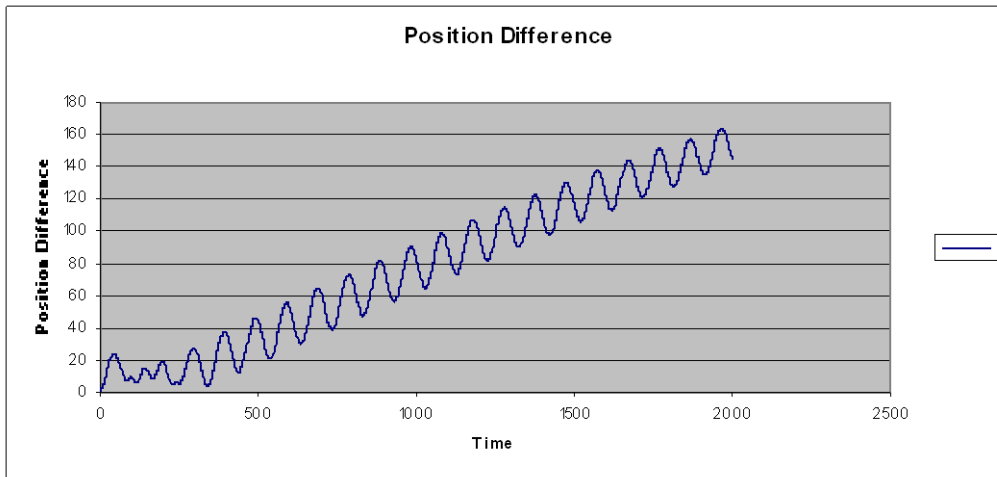


Time step: .1

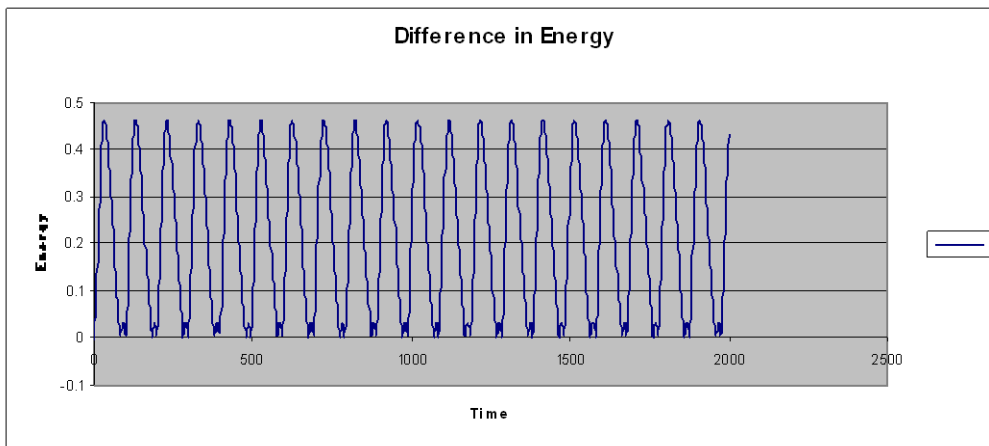
The accumulation of error in the Adams-Bashford is almost identical to that of Euler's method. Upon careful review, it is apparent that the Adams-Bashford is slightly more accurate and stable. This increase in accuracy for the two body problem does not seem very significant, especially when considering the extra computing time and implementation effort required.

Analytic v. Euler-Cromer

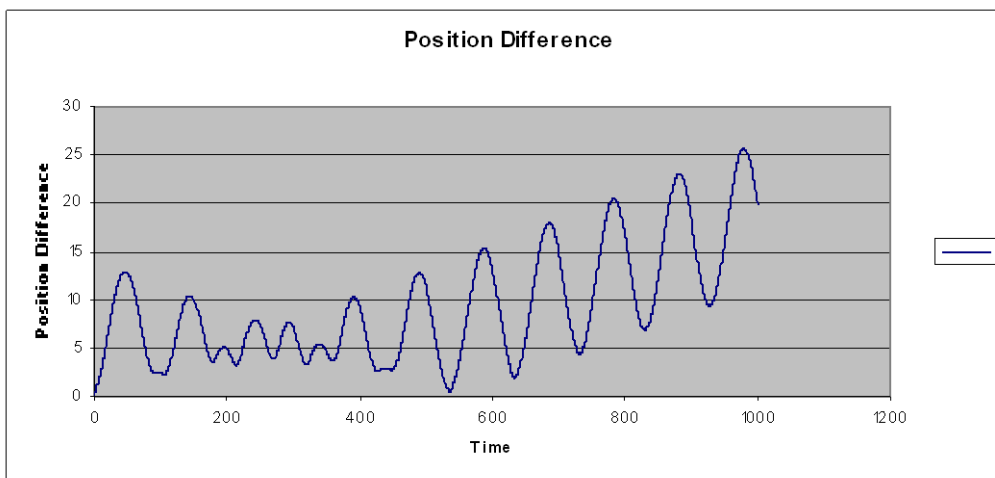




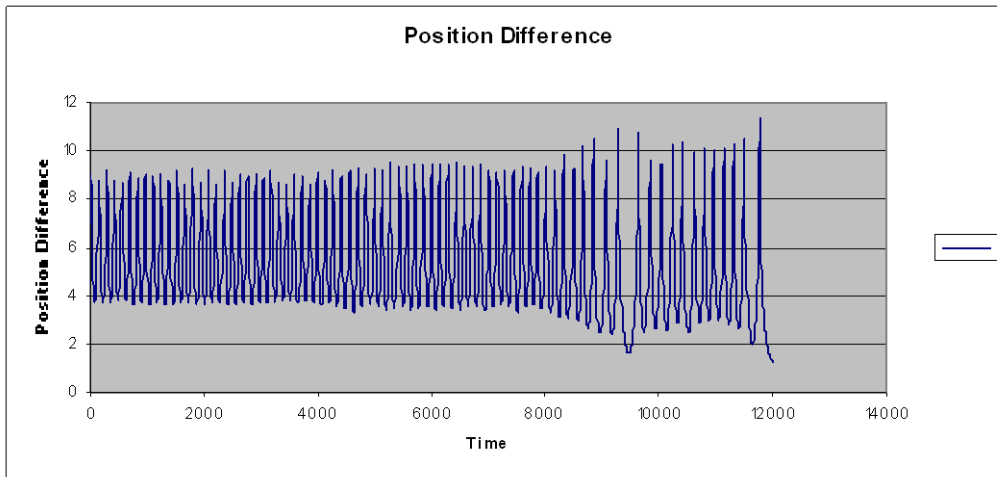
Time step: 2



Time step: 2



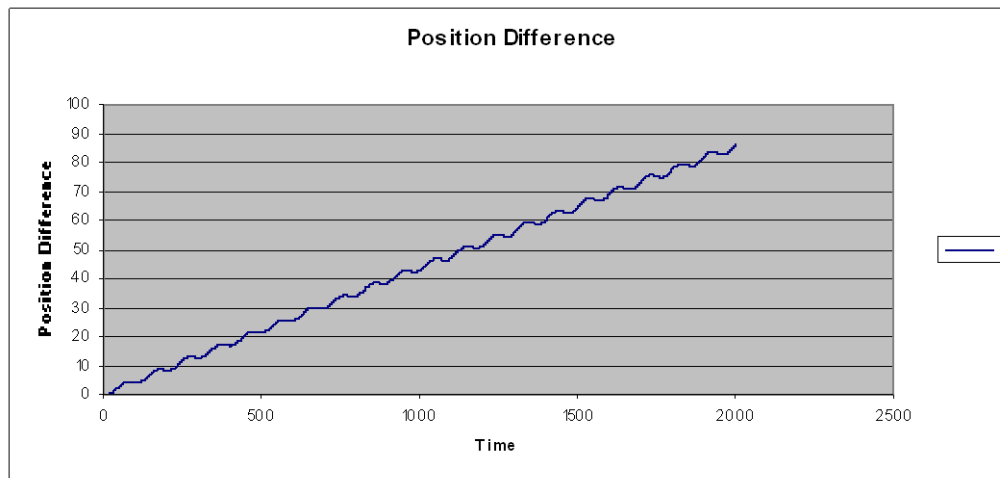
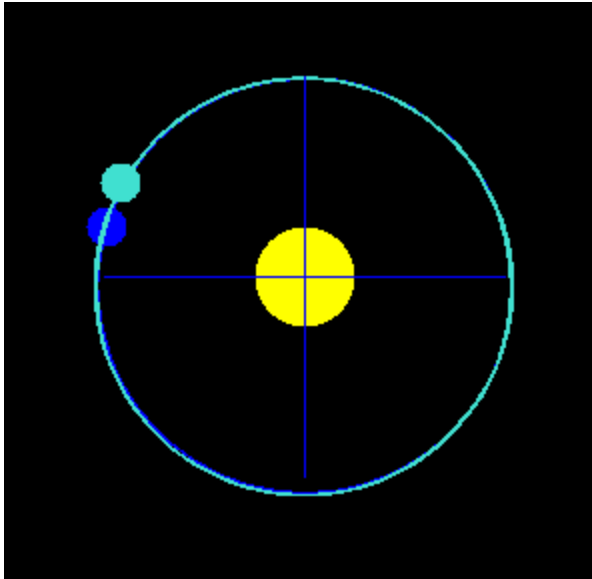
Time step: 1



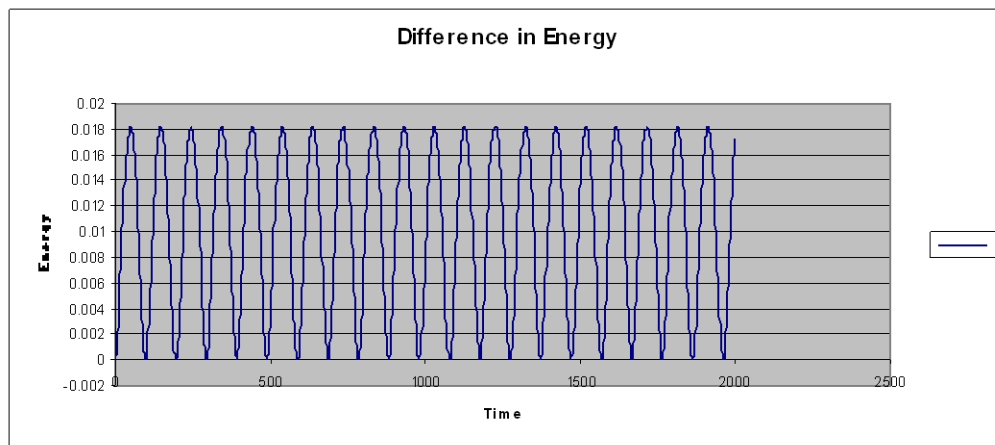
Time step: .1

The Euler-Cromer method is very stable and at low time steps the Euler-Cromer method conserves Angular Momentum and Eccentricity perfectly, and energy extremely well. For low time steps, its error in position is erratic and often significantly larger than the RK methods. The method is also quite accurate up until the time step reaches one. Further, the amount of computation required in this method is equal to that of the standard Euler's; however, it is only usable with coupled differential equations.

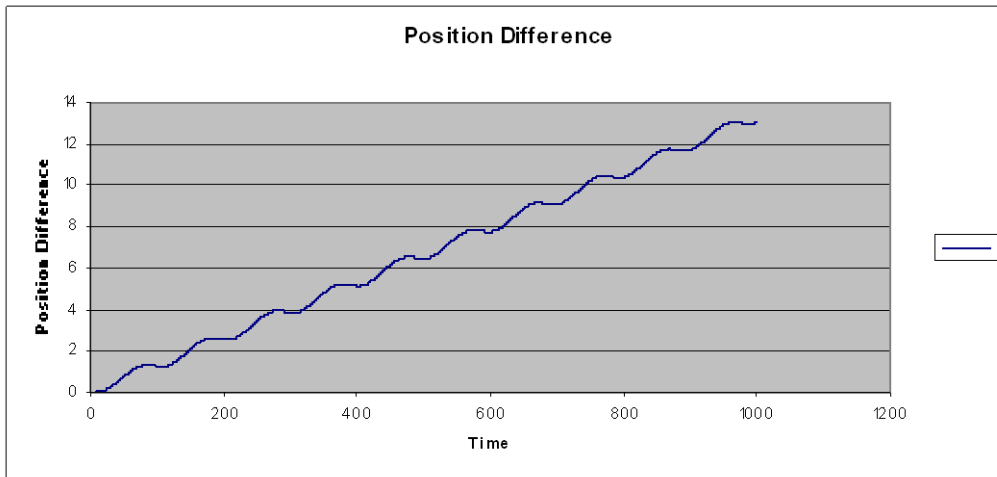
Analytic v. Leapfrog



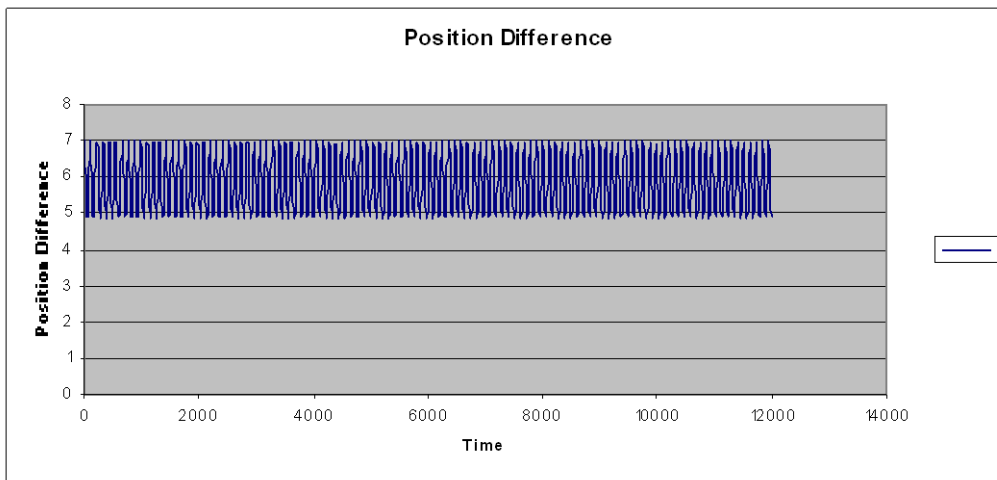
Time step: 2



Time step: 2



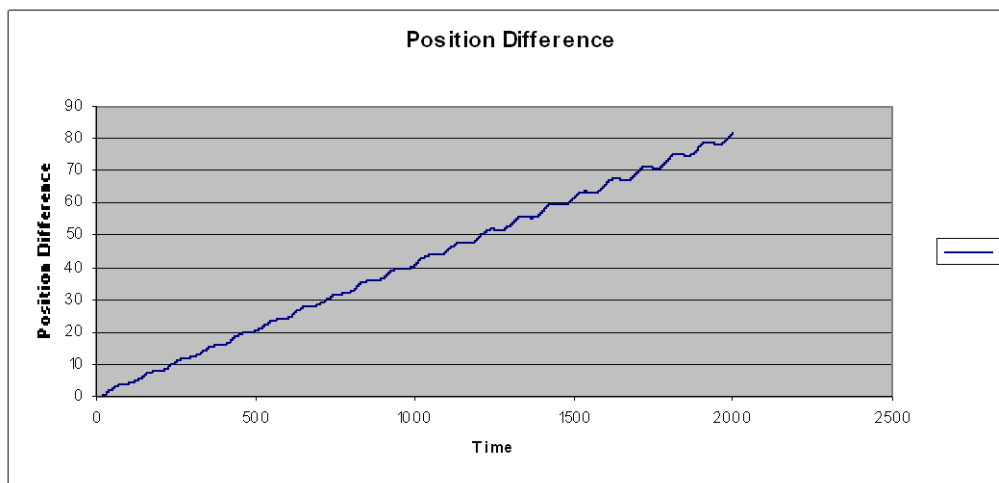
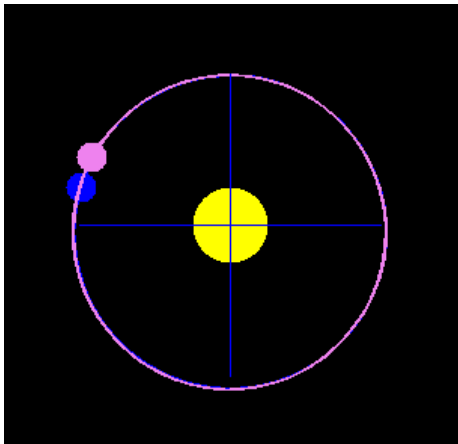
Time step: 1



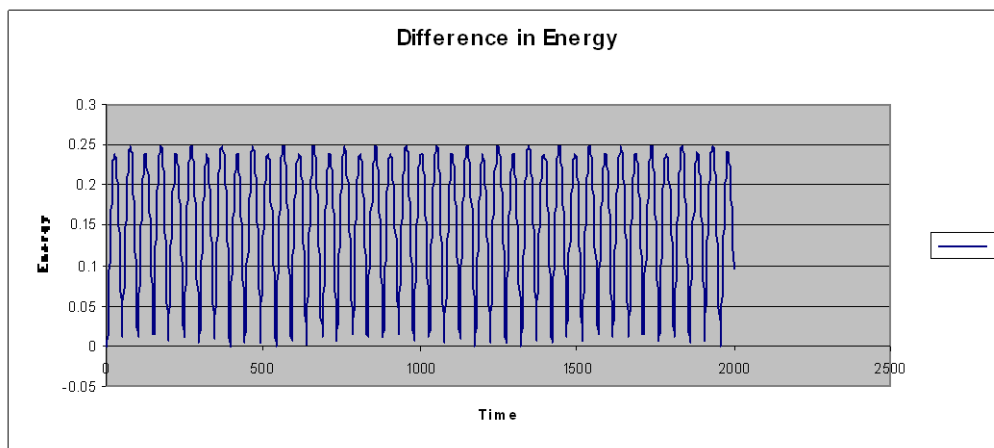
Time step: .1

The behavior of the Leapfrog method is similar to that of the Euler Cromer. It conserves angular momentum, eccentricity, and energy very well at low time steps, and is a very stable method. It also seems to have a consistent, but substantial error in position no matter how low the time step is reduced. It appears to be more accurate and stable than the Euler Cromer, but requires an additional calculation of the acceleration of the bodies. As with the Euler Cromer, it is only applicable to coupled differential equations.

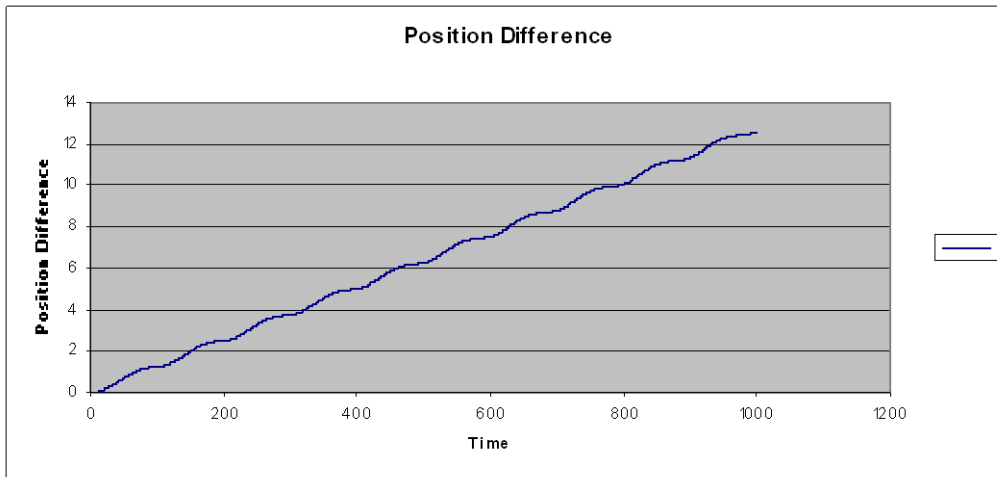
Analytic v. Runge-Kutta 2 (Our Modification)



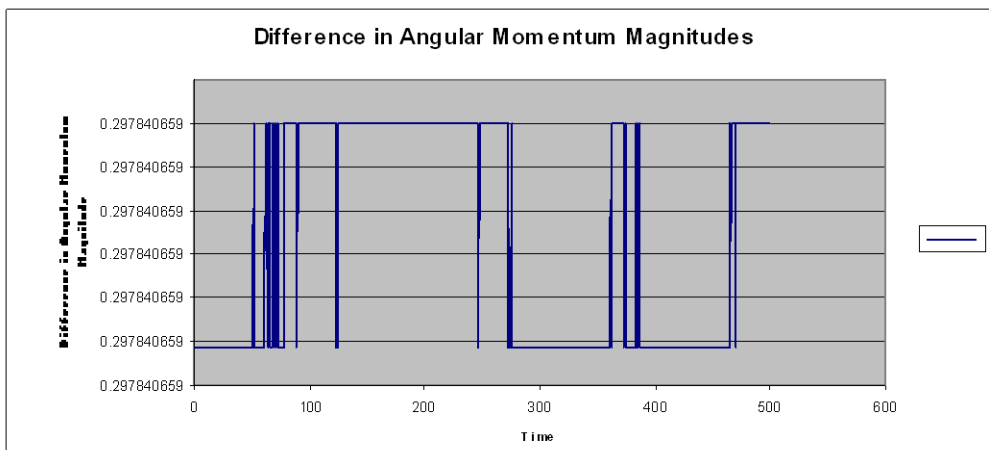
Time step: 2



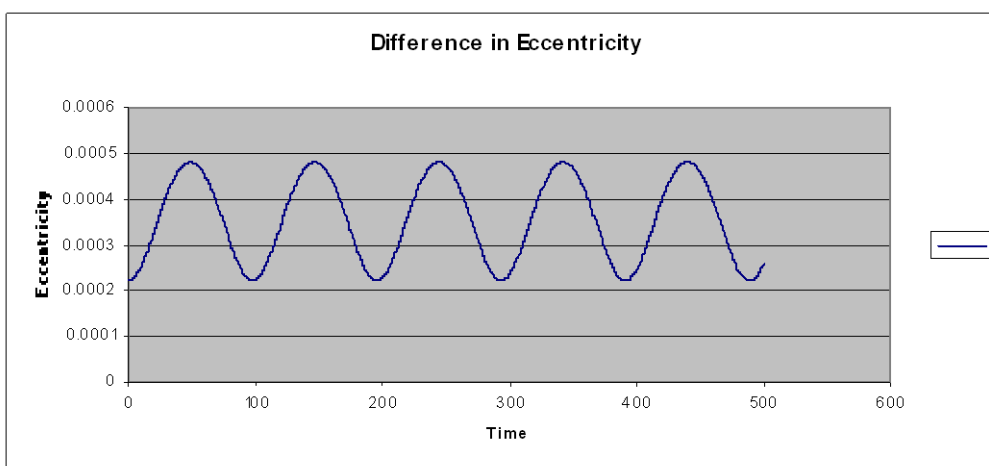
Time step: 2



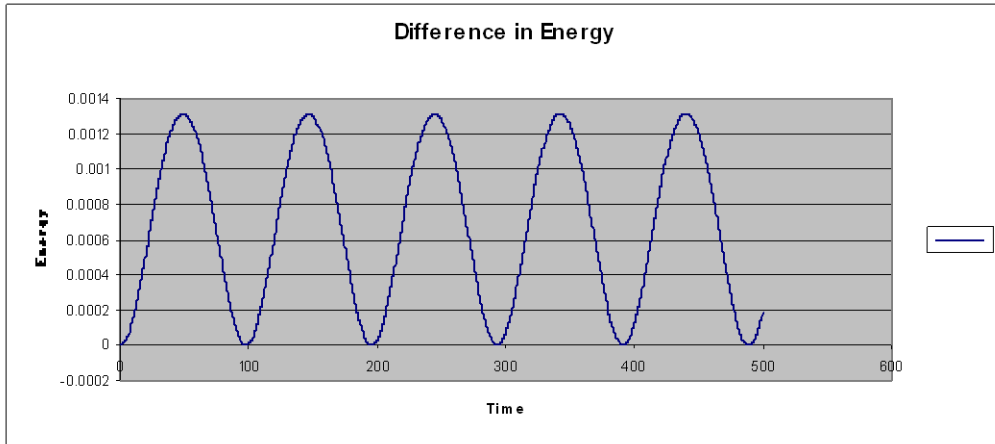
Time step: 1



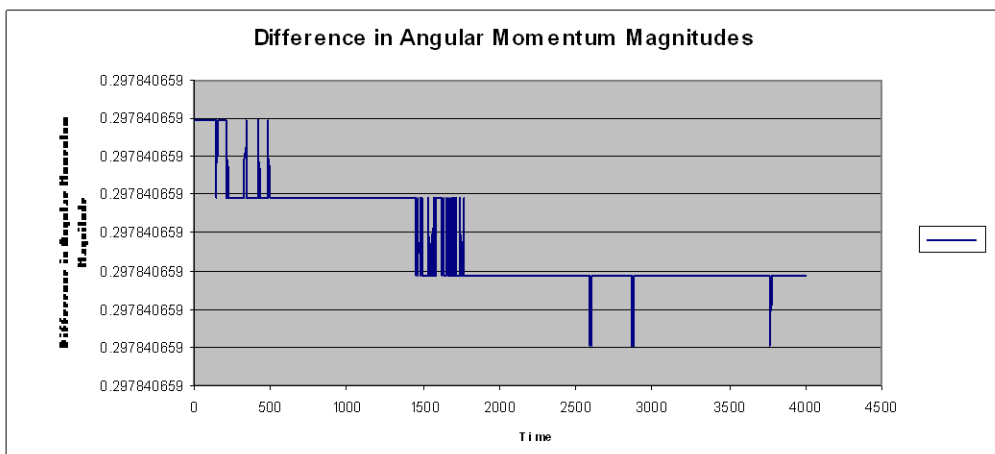
Time step: .5



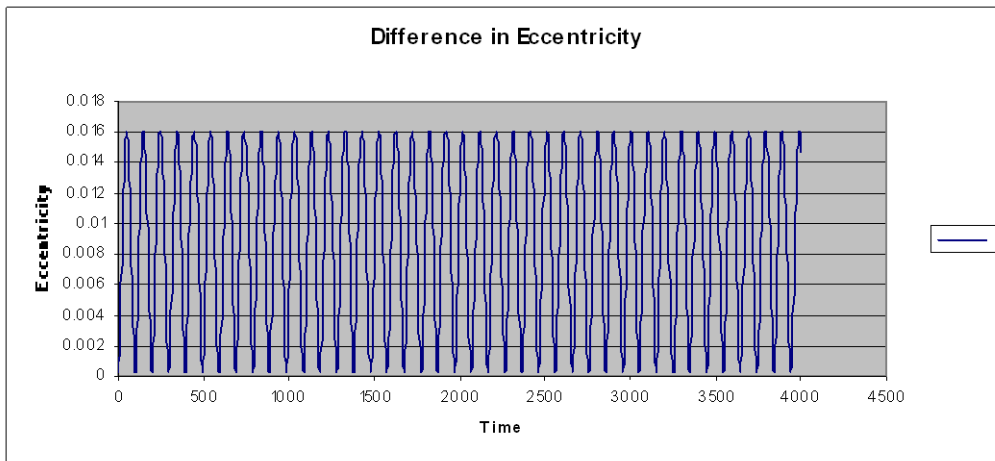
Time step: .5



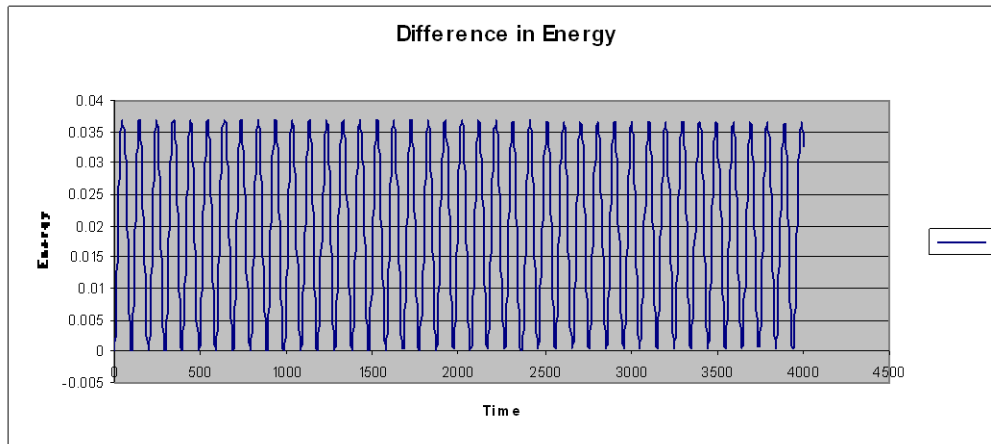
Time step: .5



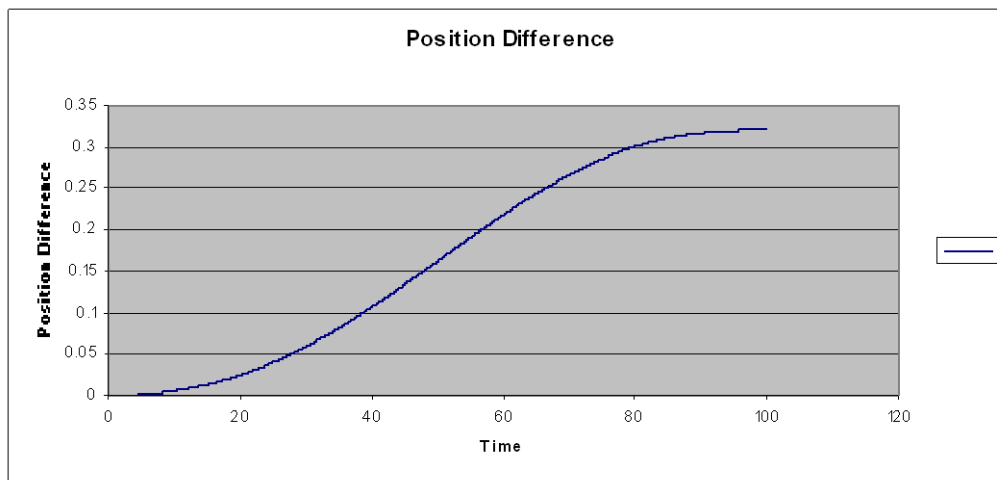
Time step: 4



Time step: 4



Time step: 4



Time step: .1

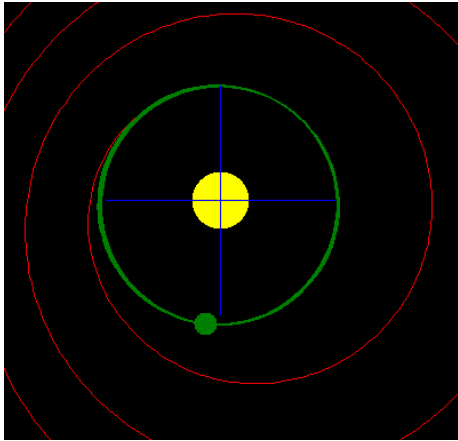
Although Gill's and RK4 are more accurate than our personal rendition of RK2, our modification is definitely an improvement on the original RK2. This method behaves in a manner very similar to that of the Leapfrog method and of the RK methods. For low time steps it conserves angular momentum and eccentricity very well, if erratically, although it is not perfect like the Leapfrog or Euler Cromer. Unlike those methods, it sigmoidally accumulates error in position on par with the RK4, which is significantly less. It is also better at conserving energy. At higher time steps the RK4 and Gill's have far less error in position. The Modified RK2 required the largest time steps before it no

longer maintained a stable orbit. This method is only applicable to coupled differential equations, but the computing time is only marginally larger than the RK2.

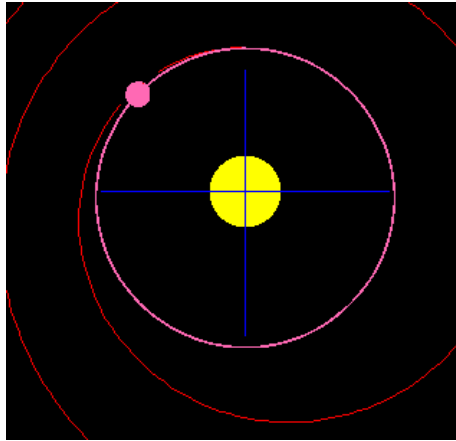
Comparing Position of Orbiting Body in Two Methods

The sections below give a visual comparison of each method to every other method. The time step for each method was set at 1. The run length varied, depending on the stability of the method. This data is meant to give the reader a clearer idea of the accuracy of each method in comparison to the others. We used numerical results primarily in comparing the true solution (analytic solution) to each method. However, we did include a numerical comparison of Gill's and RK4, because they tended to be the most stable and accurate.

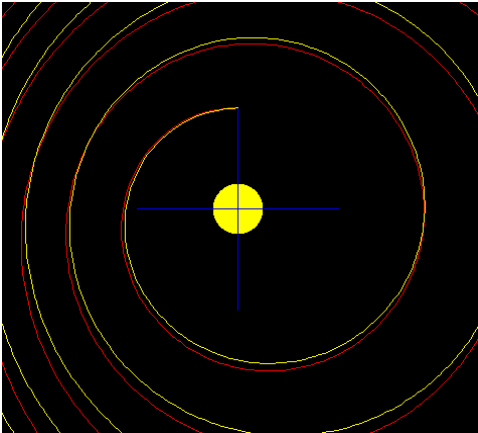
Euler v. Runge-Kutta 2



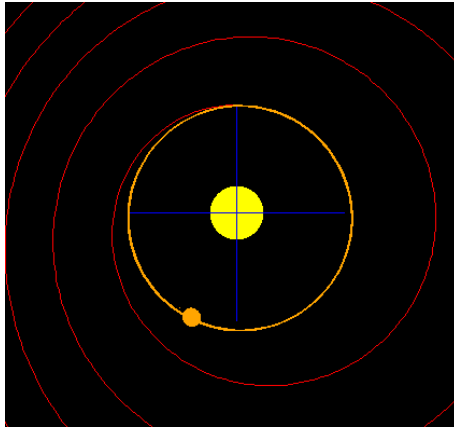
Euler v. Runge-Kutta 4



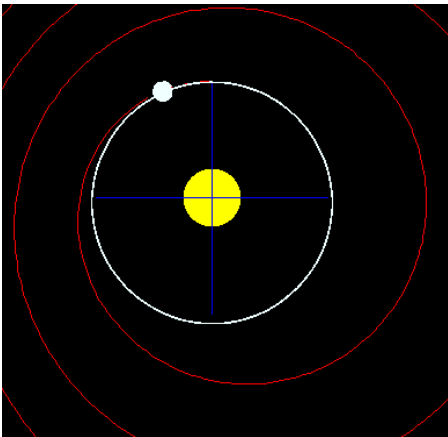
Euler v. Adams- Bashford



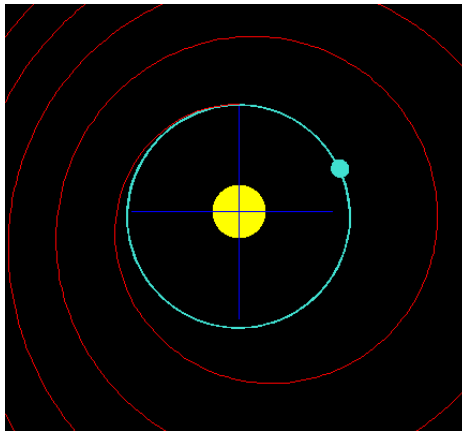
Euler v. Euler Cromer



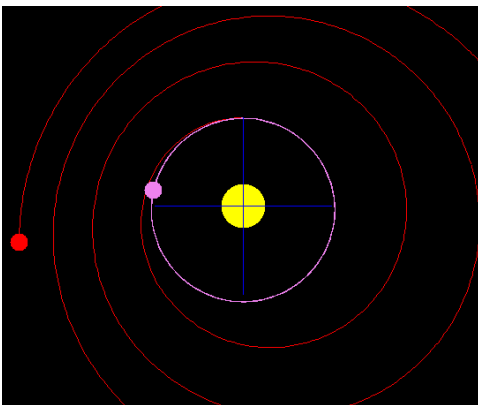
Euler v. Gill's



Euler v. Leapfrog

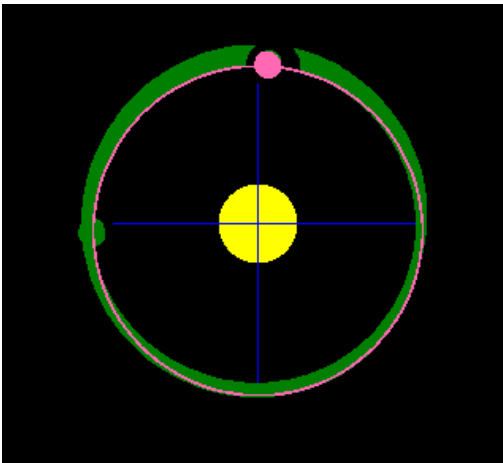


Euler v. Modified RK2

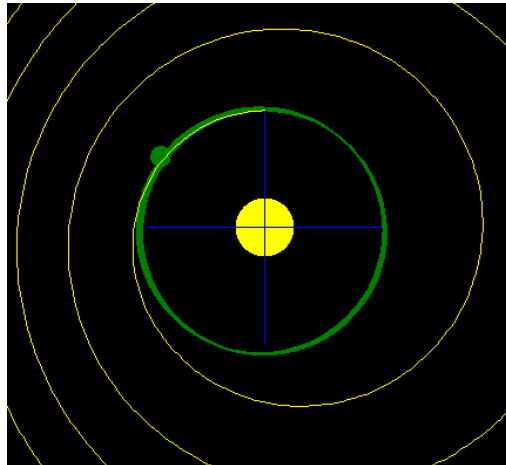


These results clearly show that Euler's Method is one of the least accurate methods. The only method that is as unstable and inaccurate as Euler is the Adams-Bashford Method. Again, because the Adams-Bashford Method was built to solve very specific problems, this is not a surprise. Even though Euler-Cromer and the Leapfrog Method are based off of the Euler Method, they are significantly more stable than Euler.

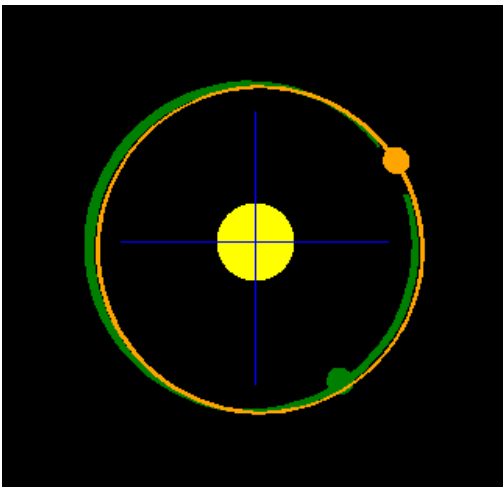
Runge-Kutta 2 v. Runge-Kutta 4



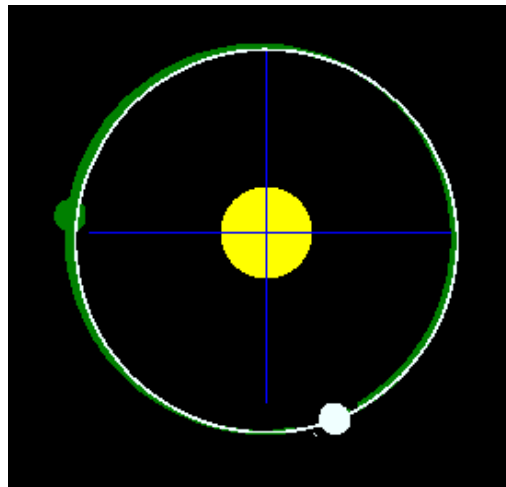
Runge-Kutta 2 v. Adams-Bashford



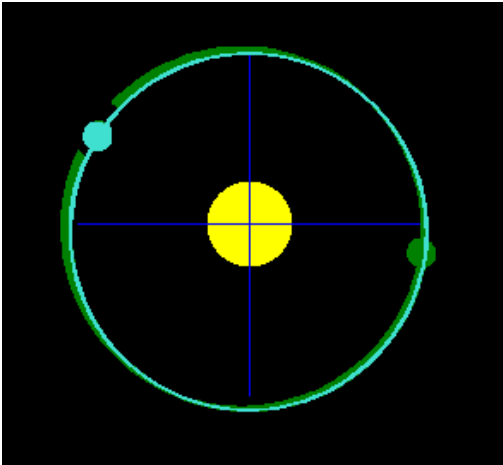
Runge-Kutta 2 v. Euler-Cromer



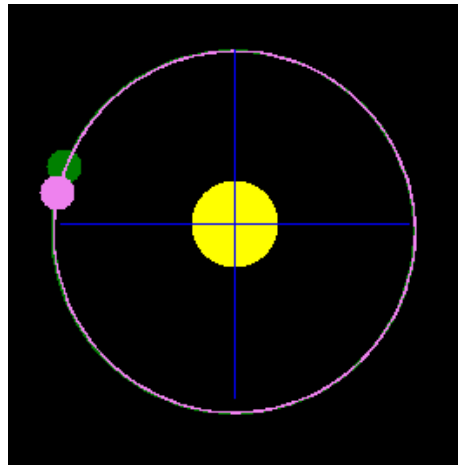
Runge-Kutta 2 v. Gill's



Runge-Kutta 2 v. Leapfrog

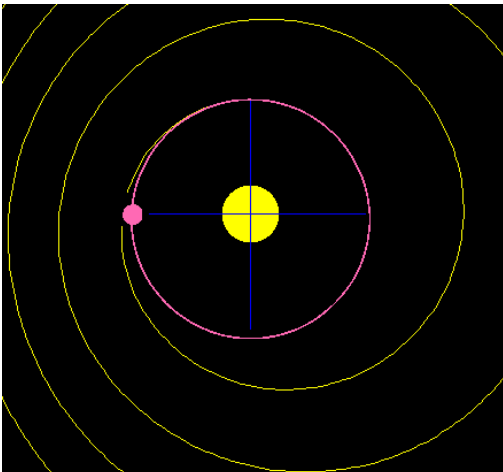


Runge-Kutta 2 v. Modified Runge-Kutta 2

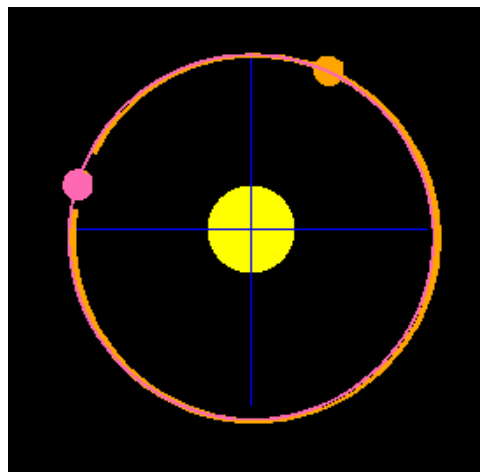


Although the Runge-Kutta 2 is not as inaccurate as the Adams-Bashford and Euler methods, it appears to gain energy every time. The thicker green line indicates that the RK2 has a slight increase in energy every time. The Euler-Cromer, Leapfrog, Gill's, and RK4 methods all appear to remain more stable and accurate than the RK2. Visually, our modification to the RK2 appears to be about as accurate as the original; however, our comparisons of energy, angular momentum, and eccentricity indicate that the modified RK2 is more accurate overall.

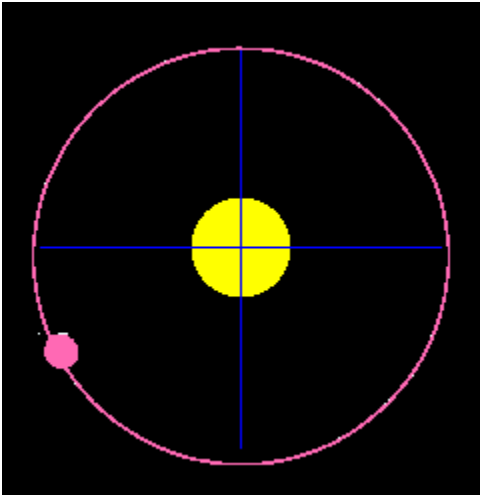
Runge-Kutta 4 v. Adams-Bashford



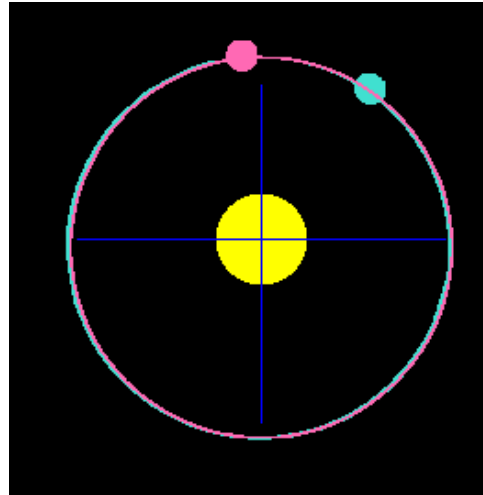
Runge-Kutta 4 v. Euler-Cromer



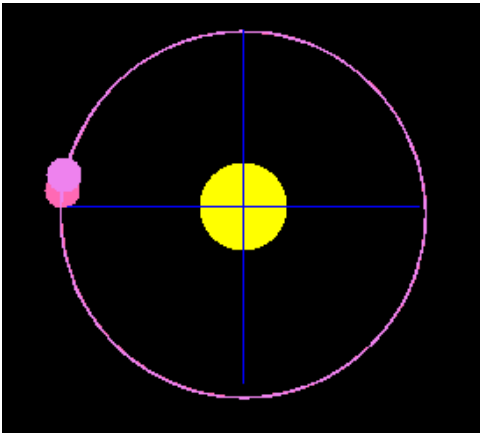
Runge-Kutta 4 v. Gill's



Runge-Kutta 4 v. Leapfrog

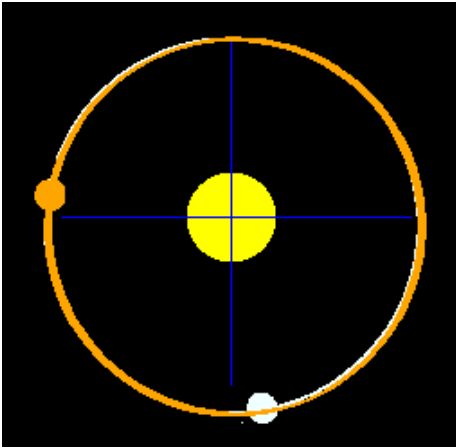


Runge-Kutta 4 v. Modified Runge-Kutta 2

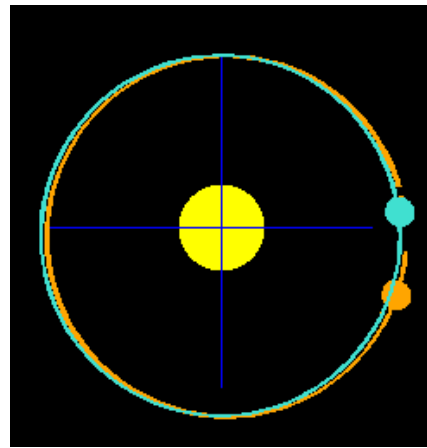


The Runge-Kutta 4 is more accurate than both the Adams-Bashford and the Euler Cromer Methods. The Gill's and Leapfrog methods follow a virtually identical path. Modified Runge-Kutta 2 also follows the RK4 very closely. We assumed that Gill's and RK4 would be very similar because Gill's is simply an adaptation of RK4 with a different weighting system.

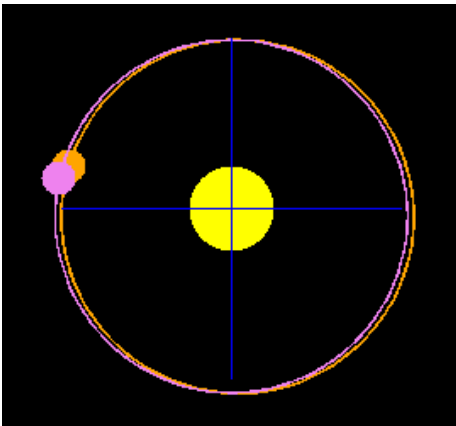
Euler-Cromer v. Gill's



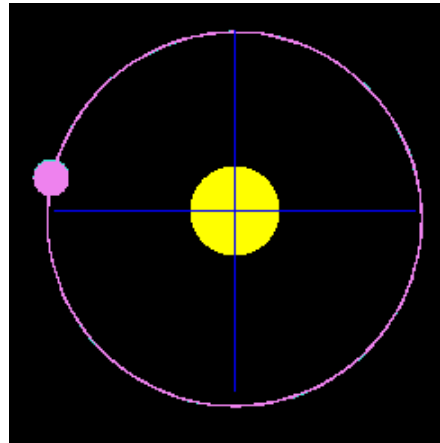
Euler-Cromer v. Leapfrog



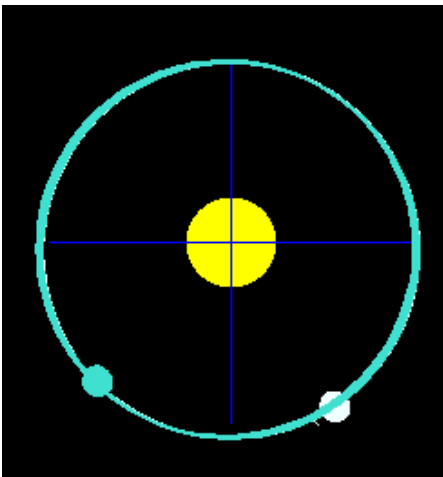
Euler-Cromer v. Modified RK2



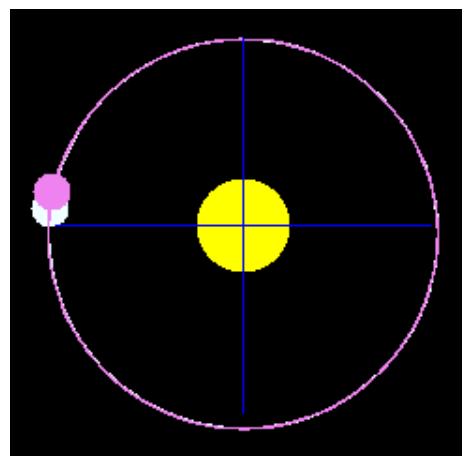
Leapfrog v. Modified RK2



Gill's v. Leapfrog



Gill's v. Modified RK2



Based on these results Gill's is the most accurate method. Although the orbits are still right on top of each other, the lines drawn by the Euler Cromer and Leapfrog orbits are thicker, indicating a slight increase in energy. Based upon all of the above results, it appears that visually, Gill's and RK4 are the most accurate methods.

Final Conclusions

After comparing all of the numerical integration methods to the analytic solution and testing each one for stability in time step, we decided Gill's Method is the best numerical integration scheme to use in this particular situation. It produced accurate results with a large time step. Further, it is not significantly more computationally intensive or difficult to implement than the basic RK4 method. We used eleven different factors in our comparison process including: Position Difference, Position Magnitude Difference, Velocity Difference, Velocity Magnitude Difference, Difference in Angular Momentum Magnitude, Difference in the Direction of Angular Momentum (Difference between L Vectors), Difference in Eccentricity, Difference in the Direction of the Eccentricity Vector (Difference between E Vectors), and Energy Difference. We also compared the position of the orbits visually, and analyzed the stability in time step for each method along with considering the difficulty of implementation and number of slope computations required. The complete set of graphs illustrating our analysis may be found in Appendix II.

As expected, Euler's Method, performed very poorly. Although it was stable on extremely small time steps, the Euler Method quickly gained energy as the time step increased. The position and velocity were extremely inaccurate compared to the analytic solution. Neither angular momentum nor eccentricity was successfully conserved in this system.

Our runner up for least accurate method was a surprise. Although we expected that Adams-Bashford would not do as well as some of the methods designed for coupled

differential equations, we did not expect it to do nearly as poorly as Euler's Method.

Like Euler's Method, Adams-Bashford was somewhat stable and accurate for the lower time steps, but it quickly became inaccurate as the time step increased. Also, the Adams-Bashford tended to gain energy and eccentricity very quickly (within the first 500 runs) and then stabilize at the new level.

Runge-Kutta 2 did fairly well. Although it also gained energy and failed to conserve angular momentum and eccentricity, it was slightly more stable than Euler's and Adams-Bashford and tended to spiral outwards at a much slower rate.

Runge-Kutta 4 was a clear runner-up. It was relatively stable, and did have any major deviations from the analytic solution until the time step was increased to 2. Runge-Kutta 4 is generally considered to be an accurate, stable numerical integration method that is easy to implement. Our results are consistent with this general sentiment. We found RK4 to be a simple, fairly accurate method. We found that a slight modification to the weighting system (Gill's Method) substantially improves the accuracy and stability of the two-body celestial mechanics problem for higher time steps. However, the RK4 method proved to be more accurate than Gill's at lower time steps. It is also very easily adapted into predictor-corrector methods, allowing optimal time steps given a necessitated accuracy to be determined automatically.

The Euler-Cromer and Leapfrog Methods were extremely stable and accurate at low time steps. For a time step of 0.1, both methods had perfect velocities, and conserved angular momentum. However, as the time step increased, both methods displayed large deviations in position. As expected, these methods did very well because they were specifically designed to estimate the solutions to coupled differential equations.

Our modified version of the Runge-Kutta 2 was a significant improvement upon the normal RK2 method. It was also the most stable method tested. At low time steps the Modified RK2 was as accurate as the RK4 and Gill's Methods, and it conserved angular momentum, eccentricity, and energy as well as the Leapfrog and Euler-Cromer methods. The computational and implementation considerations are almost identical to that of the standard RK2 method; however, it appears the Modified RK2 is only usable with coupled differential equations.

Our final conclusion is that out of the eight methods, Gill's Method is the most accurate, stable numerical integration method that can be applied to a wide range of differential equations.

Future Work

Although we were able to compare a good number of quantities in each method, we did little to experiment with the orbital parameters. In the future, we will experiment with changing the shape of the ellipse, as well as the Z Coordinate Plane. We performed all of our analysis in two dimensions for ease of documentation. We will also experiment with the masses of the bodies, creating scenarios where the two are almost equal and both have visible orbits. This would help demonstrate the effect of Newton's Law of Gravitation.

In addition, we will implement more solutions of our own design. Now that we understand how numerical integration methods are created, we can experiment with the weighting systems within the methods to develop new methods. This will allow us to tailor existing methods to specific coupled differential equations and ODEs. We attempted to implement a modified version of the RK4 earlier in the year, but our efforts have so far been unsuccessful. We will continue working with this modification to see if we can make it more stable and accurate.

We will also experiment with the animated graphic that displays the paths of the orbit, designing a more realistic scenario with graphics of the sun, earth, and other planets.

However, the most important thing we will be doing with our new knowledge base is applying it to schoolwork and real world applications. We can now estimate the solution to equations in our calculus and physics class. Not only did this project increase our understanding of C# and Microsoft Excel, but also we learned about and

experimented with a whole new field of mathematics that we can apply to much of our advanced math and science work.

Acknowledgments

Thanks to Mark Smith, our mentor, for helping us every step of the way. We appreciate everything you do for us.

Thanks to Mr. McBeth, our teacher sponsor, for helping us with all the paperwork and transportation issues.

Thanks to Realty Executives and Anita and Eric Gallagher for providing us with a great space to work and meet.

Thanks to all the family and friends who supported, encouraged, and at the very least put up with us (even at 1 am).

Bibliography

- Bowling, Sue Ann, and Geophysical Institute, University of Alaska Fairbanks. "The Earth's Changing Orbit." Alaska Science Forum 825 (15 June 1987). 30 Mar. 2008 <<http://www.gi.alaska.edu/ScienceForum/ASF8/825.html>>.
- Hornbeck, Robert W. Numerical Methods. New York: Quantum Publishers, 1975.
- Roy, A.E. Orbital Motion. 1978. 4th ed. Philadelphia: Institute of Physics Publishing, 2005.
- Sedgewick, Robert, and Kevin Wayne. Introduction to Programming in Java. 2007. Princeton. 30 Mar. 2008 <<http://www.cs.princeton.edu/introcs/94diffeq/>>.
- United States Navy. "Earth's Seasons." US Naval Observatory. 31 Jan. 2008. Astronomical Applications Department of the U.S. Naval Observatory. 30 Mar. 2008 <<http://aa.usno.navy.mil/data/docs/EarthSeasons.php>>.
- Weisstein, Eric W. "Adams' Method." Math World. Wolfram. 30 Mar. 2008 <<http://mathworld.wolfram.com/AdamsMethod.html>>.
- Weisstein, Eric W. "Gill's Method." Math World. Wolfram. 30 Mar. 2008 <<http://mathworld.wolfram.com/GillsMethod.html>>.
- Weisstein, Eric W. "Runge-Kutta Method." Math World. Wolfram. 30 Mar. 2008 <<http://mathworld.wolfram.com/Runge-KuttaMethod.html>>.
- Wikipedia. 2 Aug. 2007 <http://en.wikipedia.org/wiki/Talk:Euler%E2%80%93Cromer_algorithm>.
- Wikipedia. "Linear multistep method." Wikipedia. 16 Mar. 2008. 29 Mar. 2008 <http://en.wikipedia.org/wiki/Linear_multistep_method>.

Part III: Appendixes

Appendix I: The Code

Program.cs

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace IntegrationMethodApplication
{
    static class Program
    {
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            // runs the parent form of the application
            Application.Run(new BaseForum());
        }
    }
}
```

ProgramData.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace IntegrationMethodApplication
{
    public class ProgramData
    {
        // array of planets that contains one for each selected method
        public Body[] planets;
        //time step used by the program
        public double timeStep;
        //center body mass
        public double m1;
        //selected option to run forever
        public bool runForever;
        // amount of time until program stops
        public double runtime;
        //default constructor
        public ProgramData()
        {
        }
    }
}
```


NumericIntegration.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Threading;
//
//      This is the main form for the application
//
namespace IntegrationMethodApplication
{
    public partial class BaseForum : Form
    {
        ProgramData parameters; // class that stores all initial starting values
        SolutionClasses.OrbitFinder orbitFinder; // runs the simulation
        SolutionClasses.OrbitModel orbitModel; // forum that displays the orbits
        Analysis.DataTextOutput textData; // forum that shows all the numeric data for each method

        Thread orbits; // thread used to execute all the methods

        public BaseForum()
        {
            //creates the interface for the forum
            InitializeComponent();

            parameters = new ProgramData();

            // displays the parameter forum
            ParametersForm paramForum = new ParametersForm(ref parameters);
            paramForum.MdiParent = this;
            paramForum.Show();

            orbitModel = new SolutionClasses.OrbitModel(ref parameters);
            orbitFinder = new SolutionClasses.OrbitFinder(ref parameters, ref orbitModel);
            orbits = new Thread(new ThreadStart(orbitFinder.movePlanet));
        }

        /// <summary>
        /// this menu item of the toolbar open the parameters forum
        /// </summary>
        private void settingsToolStripMenuItem_Click(object sender, EventArgs e)
        {
            ParametersForm paramForum = new ParametersForm(ref parameters);
            paramForum.MdiParent = this;
            paramForum.Show();
        }

        /// <summary>
        /// This menu item of the tool strip opens a forum that visualises the orbit
        /// </summary>
        private void orbitModelMenuItem_Click(object sender, EventArgs e)
        {

```

```

        if (orbitModelMenuItem.Checked == true)
        {
            orbitModel.MdiParent = this;
            orbitModel.Show();
            orbitModelMenuItem.CheckState = CheckState.Checked;
        }
        else
        {
            orbitModel.Close();
            orbitModelMenuItem.CheckState = CheckState.Unchecked;
        }
    }

    /// <summary>
    /// this menu item opens the forum that shows all numerical data for the different methods
    /// </summary>
    private void locationComparisonToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if (locationComparisonToolStripMenuItem.Checked == false)
        {
            textData = new Analysis.DataTextOutput(parameters);
            textData.MdiParent = this;
            textData.Show();
            locationComparisonToolStripMenuItem.CheckState = CheckState.Checked;
        }
        else
        {
            textData.Close();
            locationComparisonToolStripMenuItem.CheckState = CheckState.Unchecked;
        }
    }

    /// <summary>
    /// This menu item closes all child forums and then quits the application
    /// </summary>
    private void ExitToolsStripMenuItem_Click(object sender, EventArgs e)
    {
        foreach (Form childForm in MdiChildren)
        {
            childForm.Close();
        }
        Application.Exit();
    }

    /// <summary>
    /// this button starts or continues the simulation
    /// </summary>
    private void run_Click(object sender, EventArgs e)
    {
        if (orbits.ThreadState == ThreadState.Unstarted)
        {
            orbits.IsBackground = true;
            orbits.Priority = ThreadPriority.Lowest;
            orbits.Start();
        }
    }

```

```

    }
    else
        if ( orbits.ThreadState != ThreadState.Background )
            orbits.Resume();
    }

    /// <summary>
    /// this button pauses the simulation
    /// </summary>
    private void stop_Click(object sender, EventArgs e)
    {
        if (orbits.ThreadState != ThreadState.Unstarted)
        {
            lock (parameters.planets)
            {
                orbits.Suspend();
            }
        }
    }
}

```

NumericIntegration.Designer.cs

```

namespace IntegrationMethodApplication
{
    partial class BaseForum
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
        false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();

```

```

        System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(BaseForum));
        this.menuStrip = new System.Windows.Forms.MenuStrip();
        this.fileMenu = new System.Windows.Forms.ToolStripMenuItem();
        this.settingsToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
        this.exitToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
        this.viewMenu = new System.Windows.Forms.ToolStripMenuItem();
        this.orbitModelMenuItem = new System.Windows.Forms.ToolStripMenuItem();
        this.statusBarToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
        this.analisisToolsToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
        this.locationComparisonToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
        this.statusStrip = new System.Windows.Forms.StatusStrip();
        this.toolStripStatusLabel = new System.Windows.Forms.ToolStripStatusLabel();
        this.ToolTip = new System.Windows.Forms.ToolTip(this.components);
        this.toolStrip1 = new System.Windows.Forms.ToolStrip();
        this.run = new System.Windows.Forms.ToolStripButton();
        this.stop = new System.Windows.Forms.ToolStripButton();
        this.menuStrip.SuspendLayout();
        this.statusStrip.SuspendLayout();
        this.toolStrip1.SuspendLayout();
        this.SuspendLayout();
        //
        // menuStrip
        //
        this.menuStrip.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
        this.fileMenu,
        this.viewMenu});
        this.menuStrip.Location = new System.Drawing.Point(0, 0);
        this.menuStrip.Name = "menuStrip";
        this.menuStrip.Size = new System.Drawing.Size(771, 24);
        this.menuStrip.TabIndex = 0;
        this.menuStrip.Text = "MenuStrip";
        //
        // fileMenu
        //
        this.fileMenu.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {
        this.settingsToolStripMenuItem,
        this.exitToolStripMenuItem});
        this.fileMenu.ImageTransparentColor = System.Drawing.SystemColors.ActiveBorder;
        this.fileMenu.Name = "fileMenu";
        this.fileMenu.Size = new System.Drawing.Size(35, 20);
        this.fileMenu.Text = "&File";
        //
        // settingsToolStripMenuItem
        //
        this.settingsToolStripMenuItem.Name = "settingsToolStripMenuItem";
        this.settingsToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
        this.settingsToolStripMenuItem.Text = "Settings";
        this.settingsToolStripMenuItem.Click += new
System.EventHandler(this.settingsToolStripMenuItem_Click);
        //
        // exitToolStripMenuItem
        //
        this.exitToolStripMenuItem.Name = "exitToolStripMenuItem";
        this.exitToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
        this.exitToolStripMenuItem.Text = "E&xit";

```

```

        this.exitToolStripMenuItem.Click += new
System.EventHandler(this.ExitToolsStripMenuItem_Click);
//
// viewMenu
//
this.viewMenu.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {
this.orbitModelMenuItem,
this.statusBarToolStripMenuItem,
this.analisisToolsToolStripMenuItem});
this.viewMenu.Name = "viewMenu";
this.viewMenu.Size = new System.Drawing.Size(41, 20);
this.viewMenu.Text = "&View";
//
// orbitModelMenuItem
//
this.orbitModelMenuItem.CheckOnClick = true;
this.orbitModelMenuItem.Name = "orbitModelMenuItem";
this.orbitModelMenuItem.Size = new System.Drawing.Size(152, 22);
this.orbitModelMenuItem.Text = "Orbit Model";
this.orbitModelMenuItem.Click += new System.EventHandler(this.orbitModelMenuItem_Click);
//
// statusBarToolStripMenuItem
//
this.statusBarToolStripMenuItem.Name = "statusBarToolStripMenuItem";
this.statusBarToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
//
// analisisToolsToolStripMenuItem
//
this.analisisToolsToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
this.locationComparisonToolStripMenuItem});
this.analisisToolsToolStripMenuItem.Name = "analisisToolsToolStripMenuItem";
this.analisisToolsToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
this.analisisToolsToolStripMenuItem.Text = "Analysis Tools";
//
// locationComparisonToolStripMenuItem
//
this.locationComparisonToolStripMenuItem.Name = "locationComparisonToolStripMenuItem";
this.locationComparisonToolStripMenuItem.Size = new System.Drawing.Size(184, 22);
this.locationComparisonToolStripMenuItem.Text = "Location Comparison";
this.locationComparisonToolStripMenuItem.Click += new
System.EventHandler(this.locationComparisonToolStripMenuItem_Click);
//
// statusStrip
//
this.statusStrip.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
this.toolStripStatusLabel});
this.statusStrip.Location = new System.Drawing.Point(0, 584);
this.statusStrip.Name = "statusStrip";
this.statusStrip.Size = new System.Drawing.Size(771, 22);
this.statusStrip.TabIndex = 2;
this.statusStrip.Text = "StatusStrip";
//
// toolStripStatusLabel
//
this.toolStripStatusLabel.Name = "toolStripStatusLabel";

```

```

this.toolStripStatusLabel.Size = new System.Drawing.Size(38, 17);
this.toolStripStatusLabel.Text = "Status";
//
// toolStrip1
//
this.toolStrip1.Dock = System.Windows.Forms.DockStyle.None;
this.toolStrip1.ImageScalingSize = new System.Drawing.Size(32, 32);
this.toolStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
this.run,
this.stop});
this.toolStrip1.Location = new System.Drawing.Point(0, 21);
this.toolStrip1.Name = "toolStrip1";
this.toolStrip1.Size = new System.Drawing.Size(84, 39);
this.toolStrip1.Stretch = true;
this.toolStrip1.TabIndex = 6;
this.toolStrip1.Text = "toolStrip1";
//
// run
//
this.run.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image;
this.run.ForeColor = System.Drawing.Color.White;
this.run.Image = ((System.Drawing.Image)(resources.GetObject("run.Image")));
this.run.ImageTransparentColor = System.Drawing.Color.Ivory;
this.run.Name = "run";
this.run.Size = new System.Drawing.Size(36, 36);
this.run.Text = "Start";
this.run.Click += new System.EventHandler(this.run_Click);
//
// stop
//
this.stop.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image;
this.stop.Image = ((System.Drawing.Image)(resources.GetObject("stop.Image")));
this.stop.ImageTransparentColor = System.Drawing.Color.Magenta;
this.stop.Name = "stop";
this.stop.Size = new System.Drawing.Size(36, 36);
this.stop.Text = "toolStripButton2";
this.stop.Click += new System.EventHandler(this.stop_Click);
//
// BaseForm
//
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(771, 606);
this.Controls.Add(this.toolStrip1);
this.Controls.Add(this.statusStrip);
this.Controls.Add(this.menuStrip);
this.IsMdiContainer = true;
this.MainMenuStrip = this.menuStrip;
this.Name = "BaseForm";
this.Text = "Numeric Integration with Orbits";
this.menuStrip.ResumeLayout(false);
this.menuStrip.PerformLayout();
this.statusStrip.ResumeLayout(false);
this.statusStrip.PerformLayout();
this.toolStrip1.ResumeLayout(false);
this.toolStrip1.PerformLayout();

```

```

        this.ResumeLayout(false);
        this.PerformLayout();

    }
    #endregion

    #region toolBar
    private System.Windows.Forms.MenuStrip menuStrip;
    private System.Windows.Forms.StatusStrip statusStrip;
    private System.Windows.Forms.ToolStripStatusLabel toolStripStatusLabel;
    private System.Windows.Forms.ToolStripMenuItem fileMenu;
    private System.Windows.Forms.ToolStripMenuItem exitToolStripMenuItem;
    private System.Windows.Forms.ToolStripMenuItem viewMenu;
    public System.Windows.Forms.ToolStripMenuItem orbitModelMenuItem;
    private System.Windows.Forms.ToolStripMenuItem statusBarToolStripMenuItem;
    private System.Windows.Forms.ToolTip ToolTip;
    #endregion

    private System.Windows.Forms.ToolStripMenuItem settingsToolStripMenuItem;
    private System.Windows.Forms.ToolStripMenuItem analysisToolsToolStripMenuItem;
    private System.Windows.Forms.ToolStripMenuItem locationComparisonToolStripMenuItem;
    private System.Windows.Forms.ToolStrip toolStrip1;
    private System.Windows.Forms.ToolStripButton run;
    private System.Windows.Forms.ToolStripButton stop;
    }
}

```

PlanetData.cs

```

using System;
using System.Collections.Generic;
using System.Text;
using System.Data;

namespace IntegrationMethodApplication
{
    public enum SolutionMethods { ANALYTIC = 0, EULERS, RK2, RK4, ADAMS, GILLS,
    EULERCROMERS, LEAPFROG, SYMPRK2};

    public class Body
    {
        public Vector r_; // position vector
        public Vector v_; // velocity vector
        public double mass; // mass of orbiting planet
        public SolutionMethods method; // solution method being used
        public DataTable storedPlanetData; // data table with all the data for the planet

        /// <summary>
        /// initializes the Body class adding the proper columns to the data table
        /// </summary>
        public Body()
        {
            storedPlanetData = new DataTable();
            storedPlanetData.Columns.Add(new DataColumn("Time"));
            storedPlanetData.Columns.Add(new DataColumn("Position Magnitude"));
            storedPlanetData.Columns.Add(new DataColumn("Position X"));

```

```

        storedPlanetData.Columns.Add(new DataColumn("Position Y"));
        storedPlanetData.Columns.Add(new DataColumn("Position Z"));
        storedPlanetData.Columns.Add(new DataColumn("Velocity Magnitude"));
        storedPlanetData.Columns.Add(new DataColumn("Velocity X"));
        storedPlanetData.Columns.Add(new DataColumn("Velocity Y"));
        storedPlanetData.Columns.Add(new DataColumn("Velocity Z"));
        storedPlanetData.Columns.Add(new DataColumn("Angular Momentum Magnitude"));
        storedPlanetData.Columns.Add(new DataColumn("Angular Momentum Direction X"));
        storedPlanetData.Columns.Add(new DataColumn("Angular Momentum Direction Y"));
        storedPlanetData.Columns.Add(new DataColumn("Angular Momentum Direction Z"));
        storedPlanetData.Columns.Add(new DataColumn("Energy"));
        storedPlanetData.Columns.Add(new DataColumn("Eccentricity Vector Magnitude"));
        storedPlanetData.Columns.Add(new DataColumn("Eccentricity Vector Direction X"));
        storedPlanetData.Columns.Add(new DataColumn("Eccentricity Vector Direction Y"));
        storedPlanetData.Columns.Add(new DataColumn("Eccentricity Vector Direction Z"));
    }

    /// <summary>
    /// calculates the angular momentum
    /// </summary>
    /// <param name="m1"></param>
    /// <returns></returns>
    public Vector angularMomentum( double m1 )
    {
        double mu = (m1 * mass) / (m1 + mass);
        return Vector.crossProduct( r_, v_ ) / mu;
    }

    /// <summary>
    /// calculates the energy of the planet
    /// </summary>
    /// <param name="bigPlanetMass"></param>
    /// <returns></returns>
    public double energy(double bigPlanetMass)
    {
        return .5 * mass * Math.Pow(v_.magnitude(), 2) - (1 * mass * bigPlanetMass / r_.magnitude());
    }

    /// <summary>
    /// calculates the eccentricity vector
    /// </summary>
    /// <param name="m1"></param>
    /// <returns></returns>
    public Vector eccentricityVector(double m1)
    {
        double G = 1;
        double M = m1 + mass;
        return Vector.crossProduct(v_, angularMomentum(m1)) / (G * M) - r_ / r_.magnitude();
    }
}
}

```

OrbitFinder.cs

```

using System;
using System.Collections.Generic;
using System.Text;

```



```

using System.Drawing;
using System.Data;

namespace IntegrationMethodApplication.SolutionClasses
{
    public class OrbitFinder
    {
        ProgramData parameters;
        OrbitModel orbitModel;
        Solution_Methods.AdamsBashford adams;
        Solution_Methods.Analytic_Solution analytic;

        //used to update the OrbitModel forum based on the new positon
        delegate void updateDelegate(object storedPlanetData, int solutionMethod);
        /// <summary>
        /// constructor for the class that receives instances of the ProgramData and orbitModel classes
        /// </summary>
        /// <param name="p"></param>
        /// <param name="orbitModel"></param>
        public OrbitFinder(ref ProgramData p, ref OrbitModel orbitModel)
        {
            this.parameters = p;
            this.orbitModel = orbitModel;
        }

        /// <summary>
        /// This function runs the simulation.
        /// It first initializes the methods.
        /// Then it continually stores the data, updates the position, and updates the model for each time step.
        /// </summary>
        public void movePlanet()
        {
            // used to store the current time
            double time = 0;
            //initializes the analytic and/or Adams Bashford Method
            initializeMethods(parameters.planets);
            //updates the position for the user inputed runlength
            while (time < parameters.runtime)
            {
                do
                {
                    //updates the position of each planet one time step
                    time += parameters.timeStep; //increases time
                    foreach (Body planet in parameters.planets)
                    {
                        StoreData(planet, time);
                        MovePlanet(planet, time);
                        updateModel(orbitModel, planet);
                    }
                }
                //an infinite loop is created if user wants to run forever or until the thread is suspended
                while (parameters.runForever);
            }
            //stores the data on the last run since StoreData is triggered before each run
            foreach (Body planet in parameters.planets)
                StoreData(planet, time);
        }
    }
}

```

```

    }
    /// <summary>
    /// initializes the adams bashford method and/or the analytic solution if needed
    /// </summary>
    /// <param name="planets"></param>
    public void initializeMethods(Body[] planets)
    {
        foreach (Body body in parameters.planets)
            if (body.method == SolutionMethods.ADAMS)
                adams = new IntegrationMethodApplication.Solution_Methods.AdamsBashford(parameters,
body);
            else if (body.method == SolutionMethods.ANALYTIC)
                analytic = new IntegrationMethodApplication.Solution_Methods.Analytic_Solution(body,
parameters.m1);
    }

    /// <summary>
    /// stores all data needed in the data table so it can later be displayed in the dataTexOutput forum
    /// </summary>
    /// <param name="planet"></param>
    /// <param name="time"></param>
    protected void StoreData ( Body planet, double time )
    {
        planet.storedPlanetData.Rows.Add();
        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Time"] = time -
parameters.timeStep;
        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Position Magnitude"] =
planet.r_.magnitude();
        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Position X"] = planet.r_.x;
        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Position Y"] = planet.r_.y;
        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Position Z"] = planet.r_.z;
        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Velocity Magnitude"] =
planet.v_.magnitude();
        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Velocity X"] =
planet.v_.x;
        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Velocity Y"] =
planet.v_.y;
        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Velocity Z"] =
planet.v_.z;
        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Angular Momentum
Magnitude"] = planet.angularMomentum(parameters.m1).magnitude();
        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Angular Momentum
Direction X"] = planet.angularMomentum(parameters.m1).x /
planet.angularMomentum(parameters.m1).magnitude();
        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Angular Momentum
Direction Y"] = planet.angularMomentum(parameters.m1).y /
planet.angularMomentum(parameters.m1).magnitude();
        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Angular Momentum
Direction Z"] = planet.angularMomentum(parameters.m1).z /
planet.angularMomentum(parameters.m1).magnitude();
        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Energy"] =
planet.energy(parameters.m1);
        Vector eccentricity = planet.eccentricityVector(parameters.m1);
        double mag = eccentricity.magnitude();
    }

```

```

        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Eccentricity Vector
Magnitude"] = mag;
        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Eccentricity Vector
Direction X"] = eccentricity.x / mag;
        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Eccentricity Vector
Direction Y"] = eccentricity.y / mag;
        planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 1]["Eccentricity Vector
Direction Z"] = eccentricity.z / mag;

    }
    /// <summary>
    /// updates the position of each planet based on the method it is using
    /// </summary>
    /// <param name="planet"></param>
    /// <param name="time"></param>
    protected void MovePlanet ( Body planet, double time )
    {
        switch (planet.method)
        {
            case SolutionMethods.ANALYTIC:
                analytic.analyticSolution(time, planet, parameters.m1);
                break;
            case SolutionMethods.EULERS:
                Solution_Methods.NumericMethods.eulerMethod(planet, parameters.m1,
parameters.timeStep);
                break;
            case SolutionMethods.RK2:
                Solution_Methods.NumericMethods.RK2Method(planet, parameters.m1,
parameters.timeStep);
                break;
            case SolutionMethods.RK4:
                Solution_Methods.NumericMethods.RK4Method(planet, parameters.m1,
parameters.timeStep);
                break;
            case SolutionMethods.ADAMS:
                adams.integrationMethod(planet.r_, planet);
                break;
            case SolutionMethods.GILLS:
                Solution_Methods.NumericMethods.GillsMethod(planet, parameters.m1,
parameters.timeStep);
                break;
            case SolutionMethods.EULERCRROMERS:
                Solution_Methods.NumericMethods.eulerCromerMethod(planet, parameters.m1,
parameters.timeStep);
                break;
            case SolutionMethods.LEAPFROG:
                Solution_Methods.NumericMethods.leapfrogMethod(planet, parameters.m1,
parameters.timeStep);
                break;
            case SolutionMethods.SYMPRK2:
                Solution_Methods.NumericMethods.sympRK2Method(planet, parameters.m1,
parameters.timeStep);
                break;
        }
    }
    /// <summary>

```

```

    /// function that calls the updateForum method
    /// </summary>
    /// <param name="storedPlanetData"></param>
    /// <param name="solutionMethod"></param>
    protected void UpdateHelper(object storedPlanetData, int solutionMethod)
    {
        orbitModel.updateForm((DataTable)storedPlanetData, solutionMethod);
    }
    /// <summary>
    /// updates the Orbit model forum if it is displayed
    /// </summary>
    /// <param name="orbitModel"></param>
    /// <param name="planet"></param>
    protected void updateModel ( OrbitModel orbitModel, Body planet )
    {
        if (orbitModel != null)
        {
            if (orbitModel.Visible == true)
            {
                try
                {
                    if (planet.storedPlanetData.Rows[planet.storedPlanetData.Rows.Count - 2] != null)
                    {
                        updateDelegate updateForum = UpdateHelper;
                        object[] updateParameters = new object[] { planet.storedPlanetData, (int)planet.method };
                        try
                        {
                            orbitModel.Invoke(updateForum, updateParameters);
                        }
                        catch { }
                    }
                }
                catch { }
            }
        }
    }
}

```

ParametersForm.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace IntegrationMethodApplication
{
    public partial class ParametersForm : Form
    {
        /// stores an instance of the starting data for the program
        private ProgramData parameters;
    }
}

```

```

/// <summary>
/// constructor that takes an instance of the ProgramData class
/// </summary>
/// <param name="parameters"></param>
public ParametersForm(ref ProgramData parameters)
{
    InitializeComponent();
    this.parameters = parameters;
}
/// <summary>
/// Button that closes the form and sets all the parameters based on the user input
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void reset_Click(object sender, EventArgs e)
{
    parameters.planets = null;
    parameters.planets = new Body[0];

    //creates one planet for each method selected and sets the planet's data according to user input
    for (int i = 0; i < checkedListBox1.Items.Count; i++)
    {
        if (checkedListBox1.GetItemChecked(i))
        {
            Array.Resize(ref parameters.planets, parameters.planets.Length + 1);
            parameters.planets[parameters.planets.Length - 1] = new Body();
            parameters.planets[parameters.planets.Length - 1].v_ = new Vector(
double.Parse(velocityXText.Text), double.Parse(velocityYText.Text), double.Parse(velocityZText.Text));
            parameters.planets[parameters.planets.Length - 1].r_ = new
Vector(double.Parse(xCordTextBox.Text), double.Parse(yCordTextBox.Text),
double.Parse(zCordTextBox.Text));
            parameters.planets[parameters.planets.Length - 1].method = (SolutionMethods)i;
            parameters.planets[parameters.planets.Length - 1].mass = double.Parse(massTextBox.Text);
        }
        parameters.m1 = double.Parse(m1TextBox.Text);
        parameters.timeStep = double.Parse(timeStepText.Text);
        parameters.runtime = double.Parse(runLengthText.Text);
        parameters.runForever = runForeverBox.Checked;

        //closes the forum
        Close();
    }
}
}

```

ParametersForm.Designer.cs

```

namespace IntegrationMethodApplication
{
    partial class ParametersForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;
    }
}

```

```

/// <summary>
/// Clean up any resources being used.
/// </summary>
/// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>

```

```

protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}

```

#region Windows Form Designer generated code

```

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.checkedListBox1 = new System.Windows.Forms.CheckedListBox();
    this.label1 = new System.Windows.Forms.Label();
    this.label2 = new System.Windows.Forms.Label();
    this.label3 = new System.Windows.Forms.Label();
    this.label4 = new System.Windows.Forms.Label();
    this.label5 = new System.Windows.Forms.Label();
    this.reset = new System.Windows.Forms.Button();
    this.label6 = new System.Windows.Forms.Label();
    this.label7 = new System.Windows.Forms.Label();
    this.label8 = new System.Windows.Forms.Label();
    this.label9 = new System.Windows.Forms.Label();
    this.label10 = new System.Windows.Forms.Label();
    this.label11 = new System.Windows.Forms.Label();
    this.timeStepText = new System.Windows.Forms.TextBox();
    this.printDocument1 = new System.Drawing.Printing.PrintDocument();
    this.runLengthText = new System.Windows.Forms.TextBox();
    this.runForeverBox = new System.Windows.Forms.CheckBox();
    this.label12 = new System.Windows.Forms.Label();
    this.velocityXText = new System.Windows.Forms.TextBox();
    this.velocityYText = new System.Windows.Forms.TextBox();
    this.velocityZText = new System.Windows.Forms.TextBox();
    this.massTextBox = new System.Windows.Forms.TextBox();
    this.xCordTextBox = new System.Windows.Forms.TextBox();
    this.yCordTextBox = new System.Windows.Forms.TextBox();
    this.zCordTextBox = new System.Windows.Forms.TextBox();
    this.m1TextBox = new System.Windows.Forms.TextBox();
    this.SuspendLayout();
    //
    // checkedListBox1
    //
    this.checkedListBox1.FormatEnabled = true;
    this.checkedListBox1.Items.AddRange(new object[] {
        "Analytic Solution",

```

```

        "Euler\'s Method",
        "Runge Kutta 2",
        "Runge Kutta 4",
        "Adams Bashford",
        "Gills",
        "Euler-Cromer",
        "LeapFrog Method",
        "Modified RK2" });
this.checkedListBox1.Location = new System.Drawing.Point(12, 63);
this.checkedListBox1.Name = "checkedListBox1";
this.checkedListBox1.Size = new System.Drawing.Size(144, 139);
this.checkedListBox1.TabIndex = 0;
//
// label1
//
this.label1.AutoSize = true;
this.label1.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.label1.Location = new System.Drawing.Point(-1, 37);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(180, 13);
this.label1.TabIndex = 1;
this.label1.Text = "Numerical Integration Methods";
//
// label2
//
this.label2.AutoSize = true;
this.label2.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.label2.Location = new System.Drawing.Point(272, 63);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(154, 13);
this.label2.TabIndex = 2;
this.label2.Text = "Planet Starting Conditions";
//
// label3
//
this.label3.AutoSize = true;
this.label3.Location = new System.Drawing.Point(259, 92);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(54, 13);
this.label3.TabIndex = 4;
this.label3.Text = "Velocity X";
//
// label4
//
this.label4.AutoSize = true;
this.label4.Location = new System.Drawing.Point(259, 118);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(54, 13);
this.label4.TabIndex = 6;
this.label4.Text = "Velocity Y";
//
// label5
//
this.label5.AutoSize = true;

```

```

this.label5.Location = new System.Drawing.Point(259, 170);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(32, 13);
this.label5.TabIndex = 8;
this.label5.Text = "Mass";
//
// reset
//
this.reset.Location = new System.Drawing.Point(312, 359);
this.reset.Name = "reset";
this.reset.Size = new System.Drawing.Size(136, 26);
this.reset.TabIndex = 9;
this.reset.Text = "Reset Graphical Display";
this.reset.UseVisualStyleBackColor = true;
this.reset.Click += new System.EventHandler(this.reset_Click);
//
// label6
//
this.label6.AutoSize = true;
this.label6.Location = new System.Drawing.Point(259, 196);
this.label6.Name = "label6";
this.label6.Size = new System.Drawing.Size(74, 13);
this.label6.TabIndex = 11;
this.label6.Text = "X - Coordinate";
//
// label7
//
this.label7.AutoSize = true;
this.label7.Location = new System.Drawing.Point(259, 222);
this.label7.Name = "label7";
this.label7.Size = new System.Drawing.Size(74, 13);
this.label7.TabIndex = 14;
this.label7.Text = "Y - Coordinate";
//
// label8
//
this.label8.AutoSize = true;
this.label8.Location = new System.Drawing.Point(259, 248);
this.label8.Name = "label8";
this.label8.Size = new System.Drawing.Size(74, 13);
this.label8.TabIndex = 15;
this.label8.Text = "Z - Coordinate";
//
// label9
//
this.label9.AutoSize = true;
this.label9.Location = new System.Drawing.Point(259, 144);
this.label9.Name = "label9";
this.label9.Size = new System.Drawing.Size(54, 13);
this.label9.TabIndex = 17;
this.label9.Text = "Velocity Z";
//
// label10
//
this.label10.AutoSize = true;
this.label10.Location = new System.Drawing.Point(259, 278);

```



```

this.label10.Name = "label10";
this.label10.Size = new System.Drawing.Size(55, 13);
this.label10.TabIndex = 17;
this.label10.Text = "Time Step";
//
// label11
//
this.label11.AutoSize = true;
this.label11.Location = new System.Drawing.Point(228, 304);
this.label11.Name = "label11";
this.label11.Size = new System.Drawing.Size(100, 13);
this.label11.TabIndex = 19;
this.label11.Text = "Center Object Mass";
//
// timeStepText
//
this.timeStepText.Location = new System.Drawing.Point(339, 271);
this.timeStepText.Name = "timeStepText";
this.timeStepText.Size = new System.Drawing.Size(100, 20);
this.timeStepText.TabIndex = 20;
this.timeStepText.Text = "1";
//
// runLengthText
//
this.runLengthText.Location = new System.Drawing.Point(339, 323);
this.runLengthText.Name = "runLengthText";
this.runLengthText.Size = new System.Drawing.Size(100, 20);
this.runLengthText.TabIndex = 21;
this.runLengthText.Text = "1000";
//
// runForeverBox
//
this.runForeverBox.AutoSize = true;
this.runForeverBox.Location = new System.Drawing.Point(12, 222);
this.runForeverBox.Name = "runForeverBox";
this.runForeverBox.Size = new System.Drawing.Size(85, 17);
this.runForeverBox.TabIndex = 22;
this.runForeverBox.Text = "Run Forever";
this.runForeverBox.UseVisualStyleBackColor = true;
//
// label12
//
this.label12.AutoSize = true;
this.label12.Location = new System.Drawing.Point(259, 330);
this.label12.Name = "label12";
this.label12.Size = new System.Drawing.Size(63, 13);
this.label12.TabIndex = 23;
this.label12.Text = "Run Length";
//
// velocityXText
//
this.velocityXText.Location = new System.Drawing.Point(339, 89);
this.velocityXText.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
this.velocityXText.Name = "velocityXText";
this.velocityXText.Size = new System.Drawing.Size(100, 20);
this.velocityXText.TabIndex = 24;

```

```

this.velocityXText.Text = "-7";
//
// velocityYText
//
this.velocityYText.Location = new System.Drawing.Point(339, 114);
this.velocityYText.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
this.velocityYText.Name = "velocityYText";
this.velocityYText.Size = new System.Drawing.Size(100, 20);
this.velocityYText.TabIndex = 25;
this.velocityYText.Text = "0";
//
// velocityZText
//
this.velocityZText.Location = new System.Drawing.Point(339, 141);
this.velocityZText.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
this.velocityZText.Name = "velocityZText";
this.velocityZText.Size = new System.Drawing.Size(100, 20);
this.velocityZText.TabIndex = 26;
this.velocityZText.Text = "0";
//
// massTextBox
//
this.massTextBox.Location = new System.Drawing.Point(339, 166);
this.massTextBox.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
this.massTextBox.Name = "massTextBox";
this.massTextBox.Size = new System.Drawing.Size(100, 20);
this.massTextBox.TabIndex = 27;
this.massTextBox.Text = "1";
//
// xCordTextBox
//
this.xCordTextBox.Location = new System.Drawing.Point(339, 193);
this.xCordTextBox.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
this.xCordTextBox.Name = "xCordTextBox";
this.xCordTextBox.Size = new System.Drawing.Size(100, 20);
this.xCordTextBox.TabIndex = 28;
this.xCordTextBox.Text = "0";
//
// yCordTextBox
//
this.yCordTextBox.Location = new System.Drawing.Point(339, 220);
this.yCordTextBox.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
this.yCordTextBox.Name = "yCordTextBox";
this.yCordTextBox.Size = new System.Drawing.Size(100, 20);
this.yCordTextBox.TabIndex = 29;
this.yCordTextBox.Text = "100";
//
// zCordTextBox
//
this.zCordTextBox.Location = new System.Drawing.Point(338, 245);
this.zCordTextBox.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
this.zCordTextBox.Name = "zCordTextBox";
this.zCordTextBox.Size = new System.Drawing.Size(100, 20);
this.zCordTextBox.TabIndex = 30;
this.zCordTextBox.Text = "0";
//

```

```

// m1TextBox
//
this.m1TextBox.Location = new System.Drawing.Point(339, 299);
this.m1TextBox.Name = "m1TextBox";
this.m1TextBox.Size = new System.Drawing.Size(100, 20);
this.m1TextBox.TabIndex = 31;
this.m1TextBox.Text = "4700";
//
// ParametersForm
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(482, 397);
this.Controls.Add(this.m1TextBox);
this.Controls.Add(this.zCordTextBox);
this.Controls.Add(this.yCordTextBox);
this.Controls.Add(this.xCordTextBox);
this.Controls.Add(this.massTextBox);
this.Controls.Add(this.velocityZText);
this.Controls.Add(this.velocityYText);
this.Controls.Add(this.velocityXText);
this.Controls.Add(this.label12);
this.Controls.Add(this.runForeverBox);
this.Controls.Add(this.runLengthText);
this.Controls.Add(this.timeStepText);
this.Controls.Add(this.label11);
this.Controls.Add(this.label9);
this.Controls.Add(this.label8);
this.Controls.Add(this.label7);
this.Controls.Add(this.label6);
this.Controls.Add(this.reset);
this.Controls.Add(this.label5);
this.Controls.Add(this.label4);
this.Controls.Add(this.label3);
this.Controls.Add(this.label2);
this.Controls.Add(this.label1);
this.Controls.Add(this.checkedListBox1);
this.Controls.Add(this.label10);
this.Name = "ParametersForm";
this.Text = "Evaluating Numerical Integration Schemes";
this.ResumeLayout(false);
this.PerformLayout();

}

#endregion

```

```

private System.Windows.Forms.CheckedListBox checkedListBox1;
private System.Windows.Forms.Label label1;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.Label label3;
private System.Windows.Forms.Label label4;
private System.Windows.Forms.Label label5;
private System.Windows.Forms.Button reset;
private System.Windows.Forms.Label label6;
private System.Windows.Forms.Label label7;

```

```

private System.Windows.Forms.Label label8;
private System.Windows.Forms.Label label9;
private System.Windows.Forms.Label label10;
private System.Windows.Forms.Label label11;
private System.Windows.Forms.TextBox timeStepText;
private System.Drawing.Printing.PrintDocument printDocument1;
private System.Windows.Forms.TextBox runLengthText;
private System.Windows.Forms.CheckBox runForeverBox;
private System.Windows.Forms.Label label12;
private System.Windows.Forms.TextBox velocityXText;
private System.Windows.Forms.TextBox velocityYText;
private System.Windows.Forms.TextBox velocityZText;
private System.Windows.Forms.TextBox massTextBox;
private System.Windows.Forms.TextBox xCordTextBox;
private System.Windows.Forms.TextBox yCordTextBox;
private System.Windows.Forms.TextBox zCordTextBox;
private System.Windows.Forms.TextBox m1TextBox;
}
}

```

OrbitModel.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Data;
using System.Text;
using System.Windows.Forms;
using System.Threading;

namespace IntegrationMethodApplication.SolutionClasses
{
    public partial class OrbitModel : Form
    {
        //used to zoom in and out
        double Transform;
        //used for transforming the cordinates into a 2D representation of 3D
        #region trig Values for Angles
        double cosAngleOne;
        double sinAngleOne;
        double cosAngleTwo;
        double sinAngleTwo;
        double cosAngleThree;
        double sinAngleThree;
        #endregion
        // stores the values for the transformation angles and each time one is changed
        // it recalculates the needed trig values
        #region TrasformationAngles
        double Angle1;
        public double angle1
        {

```

```

    get
    {
        return Angle1;
    }
    set
    {
        Angle1 = value;
        cosAngleOne = Math.Cos(angle1);
        sinAngleOne = Math.Sin(angle1);
    }
}
double Angle2;
public double angle2
{
    get
    {
        return Angle2;
    }
    set
    {
        Angle2 = value;
        cosAngleTwo = Math.Cos(angle2);
        sinAngleTwo = Math.Sin(angle2);
    }
}
double Angle3;
public double angle3
{
    get
    {
        return Angle3;
    }
    set
    {
        Angle3 = value;
        cosAngleThree = Math.Cos(angle3);
        sinAngleThree = Math.Sin(angle3);
    }
}
}
#endregion
//data structure that stores all the information about the planets
ProgramData p;

public OrbitModel(ref ProgramData p)
{
    //initially draws the forum
    InitializeComponent();

    //gets an instance to the "programdata" class
    this.p = p;

    //initially sets zoom values
    Transform = 1;

    // initially sets transform angles
    angle1 = 0;

```

```

        angle2 = 0;
        angle3 = 0;
    }

    private void OrbitModel_Paint(object sender, PaintEventArgs e)
    {
        //set up form
        Graphics display = e.Graphics;
        display.Clear(Color.Black);
        display.Transform.Scale(2000, 2000);
        display.TranslateTransform((500 / 2 + 25), (500 / 2 - 20));
        display.ScaleTransform((float)Transform, -(float)Transform);
        Pen pathPen = new Pen(Color.Green, 1f);

        display.FillEllipse(new SolidBrush(Color.Yellow), -15, -15, 30, 30); // sun
        if (p.planets != null)
        {
            for (int i = 0; i < p.planets.Length; i++) // planets
            {
                int solutionType = (int)p.planets[i].method;
                // draw paths
                drawPath(display, solutionType, p.planets[i].storedPlanetData);

                if (p.planets[i].r_ != null)
                {
                    // get an appropriate colored brush
                    SolidBrush solutionColoredBrush = new SolidBrush(Color.Black);
                    if (solutionType == 0) { solutionColoredBrush = new SolidBrush(Color.Blue); }
                    if (solutionType == 1) { solutionColoredBrush = new SolidBrush(Color.Red); }
                    if (solutionType == 2) { solutionColoredBrush = new SolidBrush(Color.Green); }
                    if (solutionType == 3) { solutionColoredBrush = new SolidBrush(Color.HotPink); }
                    if (solutionType == 4) { solutionColoredBrush = new SolidBrush(Color.Yellow); }
                    if (solutionType == 5) { solutionColoredBrush = new SolidBrush(Color.Azure); }
                    if (solutionType == 6) { solutionColoredBrush = new SolidBrush(Color.Orange); }
                    if (solutionType == 7) { solutionColoredBrush = new SolidBrush(Color.Turquoise); }
                    if (solutionType == 8) { solutionColoredBrush = new SolidBrush(Color.Violet); }
                    if (solutionType == 9) { solutionColoredBrush = new SolidBrush(Color.Violet); }
                    if (solutionType == 10) { solutionColoredBrush = new SolidBrush(Color.Violet); }
                    // draw planet
                    Point planetLocation = convertCords(p.planets[i].r_, cosAngleOne, sinAngleOne,
cosAngleTwo, sinAngleTwo, cosAngleThree, sinAngleThree);
                    display.FillEllipse(solutionColoredBrush, planetLocation.X - 10, planetLocation.Y - 10, 20,
20);
                }
            }
        }

        // draw the axis
        drawAxis(display);
    }

    private void drawSun(Graphics display)
    {
        SolidBrush sunBrush = new SolidBrush(Color.Yellow);
        display.FillEllipse(sunBrush, 500 / 2, (500 - 100) / 2, 50, 50);
    }

```

```

private void drawBody(Graphics display, string fileExtention, int xcord, int ycord, int width, int
height)
{
    display.DrawImage(Image.FromFile(fileExtention), xcord, ycord, width, height);
}

private void OrbitModel_MouseWheel(object sender, MouseEventArgs e)
{
    if (e.Delta < 0)
    {
        Transform *= .9;
    }
    else
    {
        Transform *= 1.1;
    }
    Refresh();
}

private void OrbitModel_FormClosing(object sender, FormClosingEventArgs e)
{
    e.Cancel = true;
    this.Hide();
}

private Point convertCords(Vector point3D,
double cosAngleOne, double sinAngleOne,
double cosAngleTwo, double sinAngleTwo,
double cosAngleThree, double sinAngleThree )
{
    Point point = new Point();
    Vector point0 = new Vector();
    Vector point1 = new Vector();

    point0.x = point3D.x;
    point0.y = point3D.y * cosAngleOne + point3D.z * sinAngleOne;
    point0.z = point3D.z * cosAngleOne - point3D.y * Math.Sin(angle1);

    point1.x = point0.x * cosAngleTwo - point0.z * sinAngleTwo;
    point1.y = point0.y;
    point1.z = point0.z * cosAngleTwo + point0.x * sinAngleTwo;

    point.X = (int)(point1.x * cosAngleThree + point1.y * sinAngleThree);
    point.Y = (int)(point1.y * cosAngleThree - point1.x * sinAngleThree);

    return point;
}

private void OrbitModel_KeyDown(object sender, KeyEventArgs e)
{
    //change 3D transformations
    if (e.KeyCode == Keys.Q)
        angle1 += .05;
    if (e.KeyCode == Keys.W)
        angle1 -= .05;
    if (e.KeyCode == Keys.A)

```

```

        angle2 += .05;
    if (e.KeyCode == Keys.S)
        angle2 -= .05;
    if (e.KeyCode == Keys.Z)
        angle3 += .05;
    if (e.KeyCode == Keys.X)
        angle3 -= .05;
    //change zoom
    if (e.KeyCode == Keys.E)
        Transform *= .9;
    if (e.KeyCode == Keys.R)
        Transform *= 1.1;
    //update with new values
    Refresh();
}

public void updateForm(DataTable storedData, int solutionType)
{
    if (this.WindowState != FormWindowState.Minimized)
    {
        // get new location
        Vector newPlanetLocation = new Vector();
        newPlanetLocation.x = double.Parse((string)storedData.Rows[storedData.Rows.Count -
1]["Position X"]);
        newPlanetLocation.y = double.Parse((string)storedData.Rows[storedData.Rows.Count -
1]["Position Y"]);
        newPlanetLocation.z = double.Parse((string)storedData.Rows[storedData.Rows.Count -
1]["Position Z"]);

        //get old location
        Vector oldPlanetLocation = new Vector();
        oldPlanetLocation.x = double.Parse((string)storedData.Rows[storedData.Rows.Count -
2]["Position X"]);
        oldPlanetLocation.y = double.Parse((string)storedData.Rows[storedData.Rows.Count -
2]["Position Y"]);
        oldPlanetLocation.z = double.Parse((string)storedData.Rows[storedData.Rows.Count -
2]["Position Z"]);

        //set up drawing surface
        Graphics display = CreateGraphics();
        display.Transform.Scale(2000, 2000);
        display.TranslateTransform((500 / 2 + 25), (500 / 2 - 20));
        display.ScaleTransform((float)Transform, -(float)Transform);

        //draw initial new location
        Point newLocation = convertCords(newPlanetLocation, cosAngleOne, sinAngleOne,
            cosAngleTwo, sinAngleTwo, cosAngleThree, sinAngleThree);
        GraphicsPath newLocationPath = new GraphicsPath();
        newLocationPath.AddEllipse(newLocation.X - 10, newLocation.Y - 10, 20, 20);
        Region newRegion = new Region(newLocationPath);

        //erase old location
        Point oldLocation = convertCords(oldPlanetLocation, cosAngleOne, sinAngleOne,
            cosAngleTwo, sinAngleTwo, cosAngleThree, sinAngleThree);
        GraphicsPath old = new GraphicsPath();
        old.AddEllipse(oldLocation.X - 15, oldLocation.Y - 15, 30, 30);
    }
}

```



```

    Region oldPlanet = new Region(old);
    oldPlanet.Exclude(new Rectangle(newLocation.X - 10, newLocation.Y - 10, 20, 20));
    display.FillRegion(new SolidBrush(Color.Black), oldPlanet);
    //display.FillEllipse(new SolidBrush(Color.Yellow), -25, -25, 50, 50);

    //get appropriate colored brush
    SolidBrush solutionColoredBrush = new SolidBrush(Color.Black);
    if (solutionType == 0) { solutionColoredBrush = new SolidBrush(Color.Blue); }
    if (solutionType == 1) { solutionColoredBrush = new SolidBrush(Color.Red); }
    if (solutionType == 2) { solutionColoredBrush = new SolidBrush(Color.Green); }
    if (solutionType == 3) { solutionColoredBrush = new SolidBrush(Color.HotPink); }
    if (solutionType == 4) { solutionColoredBrush = new SolidBrush(Color.Yellow); }
    if (solutionType == 5) { solutionColoredBrush = new SolidBrush(Color.Azure); }
    if (solutionType == 6) { solutionColoredBrush = new SolidBrush(Color.Orange); }
    if (solutionType == 7) { solutionColoredBrush = new SolidBrush(Color.Turquoise); }
    if (solutionType == 8) { solutionColoredBrush = new SolidBrush(Color.Violet); }
    if (solutionType == 9) { solutionColoredBrush = new SolidBrush(Color.Violet); }
    if (solutionType == 10) { solutionColoredBrush = new SolidBrush(Color.Violet); }
    //use brush to draw planet
    display.FillRegion(solutionColoredBrush, newRegion);
    // erase any old missed pixels
    display.DrawEllipse(new Pen(Color.Black, 5f), newLocation.X - 10, newLocation.Y - 10, 23,
23);

    // draw path
    drawPath(display, solutionType, storedData);
    drawAxis(display);
}
}
private void drawAxis(Graphics display)
{
    Pen bluePen = new Pen(Brushes.White, 1f);

    Point point1 = convertCords(new Vector(0, 0, 100), cosAngleOne, sinAngleOne,
        cosAngleTwo, sinAngleTwo, cosAngleThree, sinAngleThree);
    Point point2 = convertCords(new Vector(0, 0, -100), cosAngleOne, sinAngleOne,
        cosAngleTwo, sinAngleTwo, cosAngleThree, sinAngleThree);
    display.DrawLine(bluePen, point1, point2);
    point1.Y *= -1;
    display.ScaleTransform(1, -1);
    display.DrawString("Z", new Font("Times New Roman", 12), Brushes.White, point1);
    display.ScaleTransform(1, -1);

    Point point7 = convertCords(new Vector(100, 0, 0), cosAngleOne, sinAngleOne,
        cosAngleTwo, sinAngleTwo, cosAngleThree, sinAngleThree);
    Point point3 = convertCords(new Vector(-100, 0, 0), cosAngleOne, sinAngleOne,
        cosAngleTwo, sinAngleTwo, cosAngleThree, sinAngleThree);
    display.DrawLine(bluePen, point7, point3);
    point7.Y *= -1;
    display.ScaleTransform(1, -1);
    display.DrawString("X", new Font("Times New Roman", 12), Brushes.White, point7);
    display.ScaleTransform(1, -1);

    Point point4 = convertCords(new Vector(0, 100, 0), cosAngleOne, sinAngleOne,
        cosAngleTwo, sinAngleTwo, cosAngleThree, sinAngleThree);
    Point point5 = convertCords(new Vector(0, -100, 0), cosAngleOne, sinAngleOne,

```

```

        cosAngleTwo, sinAngleTwo, cosAngleThree, sinAngleThree);
display.DrawLine(bluePen, point4, point5);
point4.Y *= -1;
display.ScaleTransform(1, -1);
display.DrawString("Y", new Font("Times New Roman", 12), Brushes.White, point4);
display.ScaleTransform(1, -1);
}
private void drawPath(Graphics display, int solutionType, DataTable storedData)
{
    lock (storedData)
    {
        //get an appropriately colored pen
        Pen solutionColoredPen = new Pen(Color.HotPink);
        if (solutionType == 0) { solutionColoredPen = new Pen(Color.Blue); }
        if (solutionType == 1) { solutionColoredPen = new Pen(Color.Red); }
        if (solutionType == 2) { solutionColoredPen = new Pen(Color.Green); }
        if (solutionType == 3) { solutionColoredPen = new Pen(Color.HotPink); }
        if (solutionType == 4) { solutionColoredPen = new Pen(Color.Yellow); }
        if (solutionType == 5) { solutionColoredPen = new Pen(Color.Azure); }
        if (solutionType == 6) { solutionColoredPen = new Pen(Color.Orange); }
        if (solutionType == 7) { solutionColoredPen = new Pen(Color.Turquoise); }
        if (solutionType == 8) { solutionColoredPen = new Pen(Color.Violet); }
        if (solutionType == 9) { solutionColoredPen = new Pen(Color.Violet); }
        if (solutionType == 10) { solutionColoredPen = new Pen(Color.Violet); }

        Point prevPoint;
        Point nextPoint;
        if (storedData.Rows.Count != 0)
        {
            Vector locationFirst = new Vector();
            locationFirst.x = (int)double.Parse(storedData.Rows[0]["Position X"].ToString());
            locationFirst.y = (int)double.Parse(storedData.Rows[0]["Position Y"].ToString());
            locationFirst.z = (int)double.Parse(storedData.Rows[0]["Position Z"].ToString());

            prevPoint = convertCords(locationFirst, cosAngleOne, sinAngleOne,
                                    cosAngleTwo, sinAngleTwo, cosAngleThree, sinAngleThree);
            nextPoint = new Point();

            for (int i = 1; i < storedData.Rows.Count; i++)
            {
                if (storedData.Rows[i]["Position Y"] != null && storedData.Rows[i]["Position X"] != null)
                {
                    try
                    {
                        Vector location = new Vector();
                        location.x = (int)double.Parse(storedData.Rows[i]["Position X"].ToString());
                        location.y = (int)double.Parse(storedData.Rows[i]["Position Y"].ToString());
                        location.z = (int)double.Parse(storedData.Rows[i]["Position Z"].ToString());
                        nextPoint = convertCords(location, cosAngleOne, sinAngleOne,
                                                cosAngleTwo, sinAngleTwo, cosAngleThree, sinAngleThree);
                        display.DrawLine(solutionColoredPen, prevPoint, nextPoint);
                        prevPoint = nextPoint;
                    }
                    catch { }
                }
            }
        }
    }
}

```

```

    }
    }
    }
    }
    }
}

```

OrbitModel.Designer.cs

namespace IntegrationMethodApplication.SolutionClasses

```

{
    partial class OrbitModel
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.timer1 = new System.Windows.Forms.Timer(this.components);
            this.SuspendLayout();
            //
            // OrbitModel
            //
            this.ClientSize = new System.Drawing.Size(554, 469);
            this.DoubleBuffered = true;
            this.ForeColor = System.Drawing.SystemColors.ControlText;
            this.Location = new System.Drawing.Point(250, 25);
            this.Name = "OrbitModel";
            this.StartPosition = System.Windows.Forms.FormStartPosition.Manual;

```

```

        this.Text = "OrbitModel";
        this.MouseWheel += new
System.Windows.Forms.MouseEventHandler(this.OrbitModel_MouseWheel);
        this.Paint += new System.Windows.Forms.PaintEventHandler(this.OrbitModel_Paint);
        this.FormClosing += new
System.Windows.Forms.FormClosingEventHandler(this.OrbitModel_FormClosing);
        this.KeyDown += new System.Windows.Forms.KeyEventHandler(this.OrbitModel_KeyDown);
        this.ResumeLayout(false);

    }

    #endregion

    public System.Windows.Forms.Timer timer1;
}
}

```

DataTextOutput.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace IntegrationMethodApplication.Analysis
{
    public partial class DataTextOutput : Form
    {
        /// <summary>
        /// constructor that creates the display of the forum
        /// </summary>
        /// <param name="progData"></param>
        public DataTextOutput(ProgramData progData)
        {
            InitializeComponent(progData);
        }

        /// <summary>
        /// This is triggered on a resize event and it enlarges data tables and tabs
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void DataTextOutput_Resize(object sender, EventArgs e)
        {
            if (dataGrid != null)
            {
                this.tabControl1.Size = new System.Drawing.Size(this.Width, this.Height - 30);
                foreach (DataGridView data in dataGrid)
                {
                    data.Size = new System.Drawing.Size(this.Width, this.Height - 30);
                }
            }
        }
    }
}

```

```
}
```

DataTextOutput.Designer.cs

namespace IntegrationMethodApplication.Analysis

```
{  
    partial class DataTextOutput  
    {  
        /// <summary>  
        /// Required designer variable.  
        /// </summary>  
        private System.ComponentModel.IContainer components = null;  
  
        /// <summary>  
        /// Clean up any resources being used.  
        /// </summary>  
        /// <param name="disposing">true if managed resources should be disposed; otherwise,  
false.</param>  
        protected override void Dispose(bool disposing)  
        {  
            if (disposing && (components != null))  
            {  
                components.Dispose();  
            }  
            base.Dispose(disposing);  
        }  
    }  
}
```

#region Windows Form Designer generated code

```
/// <summary>  
/// Required method for Designer support - do not modify  
/// the contents of this method with the code editor.  
/// </summary>  
private void InitializeComponent(ProgramData progData)  
{  
    this.tabControl1 = new System.Windows.Forms.TabControl();  
  
    this.SuspendLayout();  
    //  
    // tabControl1  
    //  
    this.tabControl1.Location = new System.Drawing.Point(0, 0);  
    this.tabControl1.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);  
    this.tabControl1.Name = "tabControl1";  
    this.tabControl1.SelectedIndex = 0;  
    this.tabControl1.Size = new System.Drawing.Size(391, 323);  
    this.tabControl1.TabIndex = 0;  
    //  
    //dataGrid  
    //  
    dataGrid = new System.Windows.Forms.DataGridView[progData.planets.Length];  
    //  
    // creates a tab and a data table for each method  
    //  
    for (int i = 0; i < progData.planets.Length; i++)
```

```

{
    //
    // TabPages
    //
    System.Windows.Forms.TabPage tab = new System.Windows.Forms.TabPage();
    this.tabControl1.Controls.Add(tab);
    tab.Location = new System.Drawing.Point(4, 25);
    tab.Name = progData.planets[i].method.ToString();
    tab.Padding = new System.Windows.Forms.Padding(3);
    tab.Size = new System.Drawing.Size(this.Width, this.Height - 30);
    tab.TabIndex = i;
    tab.Text = progData.planets[i].method.ToString();
    tab.UseVisualStyleBackColor = true;
    //
    // DataGrids
    //
    dataGrid[i] = new System.Windows.Forms.DataGridView();
    dataGrid[i].ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
    dataGrid[i].Dock = System.Windows.Forms.DockStyle.Fill;
    dataGrid[i].Location = new System.Drawing.Point(3, 3);
    dataGrid[i].Name = "thingy" + i.ToString();
    dataGrid[i].Size = new System.Drawing.Size(280, 230);
    dataGrid[i].TabIndex = i + 6;
    dataGrid[i].DataSource = progData.planets[i].storedPlanetData;
    tab.Controls.Add(dataGrid[i]);
}
//
// DataTextOutput
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(388, 281);
this.Controls.Add(this.tabControl1);
this.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
this.MaximumSize = new System.Drawing.Size(675, 3000);
this.Name = "DataTextOutput";
this.Text = "Form1";
this.Resize += new System.EventHandler(this.DataTextOutput_Resize);
this.ResumeLayout(false);

}

#endregion

private System.Windows.Forms.TabControl tabControl1;
private System.Windows.Forms.DataGridView[] dataGrid;
}
}

```

AnalyticSolution.cs

```

using System;
using System.Collections.Generic;

```

```

using System.Text;

namespace IntegrationMethodApplication.Solution_Methods
{
    class Analytic_Solution
    {
        double timeAdjust;
        double period;
        //mass sum
        double M;
        // eccentricity vector
        Vector e_;
        //angular momentum vector
        Vector L_;
        double a;
        double b;
        //mechanical energy
        double energy;
        double G = 1;
        // reduced mass
        double mu;
        Vector unitX;
        Vector unitY;
        Vector unitZ;

        /// <summary>
        /// quantities calculated once to determin ellipse
        /// </summary>
        /// <param name="planet"></param>
        /// <param name="m1"></param>
        public Analytic_Solution(Body planet, double m1)
        {

            M = m1 + planet.mass;
            mu = m1 * planet.mass / M;

            L_ = Vector.crossProduct(planet.r_, planet.v_) * mu;
            energy = .5 * mu * planet.v_.magnitude() * planet.v_.magnitude()
                - G * m1 * planet.mass / planet.r_.magnitude();
            e_ = Vector.crossProduct(planet.v_, L_) / (G * M * mu) - planet.r_ / planet.r_.magnitude();

            // determines the unit vectors of eccentricity vector, angular momentum vector, and the orthaganal
            // vector to them. This defines the corodinate system within the ellipse
            unitX = e_ / e_.magnitude();
            unitZ = L_ / L_.magnitude();
            unitY = Vector.crossProduct(unitZ, unitX);

            double cosTheta = Vector.dotProduct(planet.r_, e_) / (planet.r_.magnitude() * e_.magnitude());
            //angle between the eccentricity vector and the position vector, its the true anomaly at time 0
            a = planet.r_.magnitude() * (1 + e_.magnitude() * cosTheta) / (1 - e_.magnitude() *
            e_.magnitude()); //semi-major axis
            b = a * Math.Sqrt(1 - e_.magnitude() * e_.magnitude()); //semi-minor axis

            period = 2 * Math.PI * Math.Sqrt(Math.Pow(a, 3) / (G * M)); //calculates the period of the orbit

            //determins eccentric anomaly

```

```

double eAnomaly;
double coseAnomaly = ((e_.magnitude() + cosTheta) / (1 + e_.magnitude() * cosTheta));
coseAnomaly %= Math.PI;
//disambiguates the angle based on planet location
if (Vector.dotProduct(Vector.crossProduct(e_, planet.r_), unitZ) < 0)
    eAnomaly = Math.Acos(coseAnomaly);
else
    eAnomaly = -Math.Acos(coseAnomaly);

//calculates mean anomaly
double mAnomaly = eAnomaly - e_.magnitude() * Math.Sin(eAnomaly);
//calculates time since perihelion passage
timeAdjust = mAnomaly * period / (2 * Math.PI);

}
/// <summary>
/// updates the position of the planet with respect to time
/// </summary>
/// <param name="t"></param>
/// <param name="planet"></param>
/// <param name="m1"></param>
public void analyticSolution(double t, Body planet, double m1)
{
    //calculates mean anomaly at given time
    double mAnomaly = (2 * Math.PI * ((t - timeAdjust) / period));
    //modulates to keep precision and reduce computing time
    mAnomaly %= 2 * Math.PI;
    double nu; //true anomaly
    //solves Kepler's Equation numerically w/ precision of 1E-14
    double eAnomaly = 1;
    double eAnomalyNew = 1.1;
    while (Math.Abs(eAnomaly - eAnomalyNew) > 1E-14)
    {
        eAnomaly = eAnomalyNew;
        eAnomalyNew = mAnomaly + e_.magnitude() * Math.Sin(eAnomaly);
    }
    //calculates true anomaly
    nu = 2 * Math.Atan((Math.Sqrt((1 + e_.magnitude()) / (1 - e_.magnitude()))) * Math.Tan(eAnomaly /
2)));
    //calculates distance from center of mass
    double r = a * (1 - e_.magnitude() * Math.Cos(eAnomaly));
    //determines coordinates with respect to the ellipse
    Vector posElliptical = new Vector(r * Math.Cos(nu), r * Math.Sin(nu), 0);
    //rotates into standard coordinate system
    planet.r_ = posElliptical.x * unitX + posElliptical.y * unitY + posElliptical.z * unitZ;
    //calculates magnitude of velocity from energy
    double vMag = Math.Sqrt((energy + G * m1 * planet.mass / planet.r_.magnitude()) / (.5 * mu));

    //Phi is the angle between the position vector and the velocity vector
    double sinPhi = (1 + e_.magnitude() * Math.Cos(nu)) / Math.Sqrt(1 + e_.magnitude() *
e_.magnitude() + 2 * e_.magnitude() * Math.Cos(nu));
    double cosPhi = -(e_.magnitude() * Math.Sin(nu)) / Math.Sqrt(1 + e_.magnitude() * e_.magnitude()
+ 2 * e_.magnitude() * Math.Cos(nu));

    //disambiguates velocity

```



```

double velocityAngle;
if (sinPhi > 0)
{
    if (cosPhi > 0)
        velocityAngle = Math.Asin(sinPhi);
    else
        velocityAngle = Math.Acos(cosPhi);
}
else
{
    if (cosPhi > 0)
        velocityAngle = Math.Asin(sinPhi);
    else
        velocityAngle = Math.PI * 2 - Math.Asin(sinPhi);
}
//velocity in corodinales in refrence to the elipse
Vector velocityEliptic = new Vector(vMag * Math.Cos(velocityAngle + nu), vMag *
Math.Sin(velocityAngle + nu), 0);
//rotates into standard corodinate system
planet.v_ = velocityEliptic.x * unitX + velocityEliptic.y * unitY + velocityEliptic.z * unitZ;
}
}
}

```

AdamsBashford.cs

```

using System;
using System.Collections.Generic;
using System.Text;

namespace IntegrationMethodApplication.Solution_Methods
{
    class AdamsBashford
    {
        ProgramData data;
        //is the planet being updated
        Body planet;
        // saves previous locations
        Vector[] points;
        // saves previous velocities
        Vector[] velocities;
        // saves previous accelerations
        Vector[] gravity;
        // used in initialization
        int timeStepCounter;

        /// <summary>
        /// constructor that initializes the method
        /// </summary>
        /// <param name="data"></param>
        /// <param name="planet"></param>
        public AdamsBashford(ProgramData data, Body planet)
        {
            //used for the first 4 points
            timeStepCounter = 4;
            //gets needed instances of classes

```

```

    this.data = data;
    this.planet = planet;
    Body planetCopy = planet;

    points = new Vector[4];
    velocities = new Vector[4];
    gravity = new Vector[4];

    //initializes the method
    start(planetCopy);
}
/// <summary>
/// updates the position each time step when it is called
/// </summary>
/// <param name="currentPoint"></param>
/// <param name="planet"></param>
public void integrationMethod( Vector currentPoint, Body planet )
{
    // the first 4 points are already calculated so it outputs those for the first 4 points
    if (timeStepCounter != 0)
    {
        int step = 4 - timeStepCounter;
        planet.r_ = points[step];
        planet.v_ = velocities[step];
        timeStepCounter--;
    }
    //calculates new position based on the 4th order adams bashford
    else
    {
        velocities[0] = velocities[1];
        velocities[1] = velocities[2];
        velocities[2] = velocities[3];
        velocities[3] = planet.v_;

        gravity[0] = gravity[1];
        gravity[1] = gravity[2];
        gravity[2] = gravity[3];
        gravity[3] = NumericMethods.gravityVector(planet.mass, data.m1, planet.r_);

        planet.r_ += data.timeStep * ((55 / 24) * velocities[3] - (59 / 24) * velocities[2] + (37 / 24) *
velocities[1] - (3 / 8) * velocities[0]);
        planet.v_ += data.timeStep * ((55 / 24) * gravity[3] - (59 / 24) * gravity[2] + (37 / 24) *
gravity[1] - (3 / 8) * gravity[0]);
    }
}
/// <summary>
/// used in the initialization of the method.
/// Calculates the first 4 data sets using the Runge-Katta 4 method.
/// </summary>
protected void start(Body planetCopy)
{
    gravity[0] = NumericMethods.gravityVector(planetCopy.mass, data.m1, planetCopy.r_);
    NumericMethods.RK4Method(planetCopy, data.m1, data.timeStep);
    velocities[0] = planetCopy.v_;
    points[0] = planetCopy.r_;
}

```

```

        gravity[1] = NumericMethods.gravityVector(planetCopy.mass, data.m1, planetCopy.r_);
        NumericMethods.RK4Method(planetCopy, data.m1, data.timeStep);
        velocities[1] = planetCopy.v_;
        points[1] = planetCopy.r_;

        gravity[2] = NumericMethods.gravityVector(planetCopy.mass, data.m1, planetCopy.r_);
        NumericMethods.RK4Method(planetCopy, data.m1, data.timeStep);
        velocities[2] = planetCopy.v_;
        points[2] = planetCopy.r_;

        gravity[3] = NumericMethods.gravityVector(planetCopy.mass, data.m1, planetCopy.r_);
        NumericMethods.RK4Method(planetCopy, data.m1, data.timeStep);
        velocities[3] = planetCopy.v_;
        points[3] = planetCopy.r_;
    }
}
}

```

NumericMethods.cs

```

using System;
using System.Collections.Generic;
using System.Text;

namespace IntegrationMethodApplication.Solution_Methods
{
    class NumericMethods
    {
        /// <summary>
        /// Euler Method
        /// </summary>
        /// <param name="planet"></param>
        /// <param name="centerPlanetMass"></param>
        /// <param name="timeStep"></param>
        public static void eulerMethod(Body planet, double centerPlanetMass, double timeStep)
        {
            Vector gravity = gravityVector(planet.mass, centerPlanetMass, planet.r_) / planet.mass;
            planet.r_ += (planet.v_ * timeStep);
            planet.v_ += gravity * timeStep;
        }
        /// <summary>
        /// Euler Cromer Method which is a symplectic method
        /// </summary>
        /// <param name="planet"></param>
        /// <param name="centerPlanetMass"></param>
        /// <param name="timeStep"></param>
        public static void eulerCromerMethod(Body planet, double centerPlanetMass, double timeStep)
        {
            //symplectic integrator
            Vector gravity = gravityVector(planet.mass, centerPlanetMass, planet.r_) / planet.mass;
            planet.v_ += gravity * timeStep;
            planet.r_ += (planet.v_ * timeStep);
        }
        /// <summary>
        /// gills method - similar to the Runge-Katta 4 except it has different coefficients
        /// </summary>
    }
}

```

```

/// <param name="planet"></param>
/// <param name="centerPlanetMass"></param>
/// <param name="timeStep"></param>
public static void GillsMethod(Body planet, double centerPlanetMass, double timeStep)
{
    Vector gravityk1 = gravityVector(planet.mass, centerPlanetMass, planet.r_) / planet.mass;
    Vector velocityk1 = planet.v_;

    Vector gravityk2 = gravityVector(planet.mass, centerPlanetMass, planet.r_ + timeStep * velocityk1
/ 2) / planet.mass;
    Vector velocityk2 = planet.v_ + timeStep * gravityk1 / 2;

    Vector gravityk3 = gravityVector(planet.mass, centerPlanetMass, planet.r_ + timeStep * (.5 *
(Math.Sqrt(2) - 1) * velocityk1 + (1 - .5 * Math.Sqrt(2)) * velocityk2)) / planet.mass;
    Vector velocityk3 = planet.v_ + timeStep * (.5 * (Math.Sqrt(2) - 1) * gravityk1 + (1 - .5 *
Math.Sqrt(2)) * gravityk2);

    Vector gravityk4 = gravityVector(planet.mass, centerPlanetMass, planet.r_ + timeStep * (-.5 *
Math.Sqrt(2) * velocityk2 + (1 + .5 * Math.Sqrt(2)) * velocityk3)) / planet.mass;
    Vector velocityk4 = planet.v_ + timeStep * (-.5 * Math.Sqrt(2) * gravityk2 + (1 + .5 *
Math.Sqrt(2)) * gravityk3);

    planet.r_ += timeStep * (velocityk1 + (2 - Math.Sqrt(2)) * velocityk2 + (2 + Math.Sqrt(2)) *
velocityk3 + velocityk4) / 6;
    planet.v_ += timeStep * (gravityk1 + (2 - Math.Sqrt(2)) * gravityk2 + (2 + Math.Sqrt(2)) *
gravityk3 + gravityk4) / 6;
}
/// <summary>
/// Leapfrog method - calculates the slope 2 times
/// </summary>
/// <param name="planet"></param>
/// <param name="centerPlanetMass"></param>
/// <param name="timeStep"></param>
public static void leapfrogMethod(Body planet, double centerPlanetMass, double timeStep)
{
    Vector gravity = gravityVector(planet.mass, centerPlanetMass, planet.r_) / planet.mass;
    planet.r_ += (planet.v_ * timeStep) + .5 * gravity * timeStep * timeStep;
    Vector nextGrav = gravityVector(planet.mass, centerPlanetMass, planet.r_) / planet.mass;
    planet.v_ += .5 * (gravity + nextGrav) * timeStep;
}
/// <summary>
/// Runge-Katta 2 method - calculates the slope 2 times
/// </summary>
/// <param name="planet"></param>
/// <param name="centerPlanetMass"></param>
/// <param name="timeStep"></param>
public static void RK2Method(Body planet, double centerPlanetMass, double timeStep)
{
    Vector gravityk1 = gravityVector(planet.mass, centerPlanetMass, planet.r_) / planet.mass;
    Vector velocityk1 = planet.v_;

    Vector gravityk2 = gravityVector(planet.mass, centerPlanetMass, planet.r_ + timeStep * velocityk1
/ 2) / planet.mass;
    Vector velocityk2 = planet.v_ + timeStep * gravityk1 / 2;

    planet.r_ += velocityk2 * timeStep;

```

```

    planet.v_ += gravityk2 * timeStep;
}
public static void RK4Method(Body planet, double centerPlanetMass, double timeStep)
{
    Vector gravityk1 = gravityVector(planet.mass, centerPlanetMass, planet.r_) / planet.mass;
    Vector velocityk1 = planet.v_;

    Vector gravityk2 = gravityVector(planet.mass, centerPlanetMass, planet.r_ + timeStep * velocityk1
/ 2) / planet.mass;
    Vector velocityk2 = planet.v_ + timeStep * gravityk1 / 2;

    Vector gravityk3 = gravityVector(planet.mass, centerPlanetMass, planet.r_ + timeStep * velocityk2
/ 2) / planet.mass;
    Vector velocityk3 = planet.v_ + timeStep * gravityk2 / 2;

    Vector gravityk4 = gravityVector(planet.mass, centerPlanetMass, planet.r_ + timeStep * velocityk3)
/ planet.mass;
    Vector velocityk4 = planet.v_ + timeStep * gravityk3;

    planet.r_ += timeStep * (velocityk1 + 2 * velocityk2 + 2 * velocityk3 + velocityk4) / 6;
    planet.v_ += timeStep * (gravityk1 + 2 * gravityk2 + 2 * gravityk3 + gravityk4) / 6;
}
/// <summary>
/// This is our own method that works off of the RK2 and uses a velocity calculated on the current
position instead of the previous
/// </summary>
/// <param name="planet"></param>
/// <param name="centerPlanetMass"></param>
/// <param name="timeStep"></param>
public static void sympRK2Method(Body planet, double centerPlanetMass, double timeStep)
{
    Vector velocityk1 = planet.v_;
    Vector gravityk1 = gravityVector(planet.mass, centerPlanetMass, planet.r_ + timeStep * velocityk1
/ 2) / planet.mass;

    Vector velocityk2 = planet.v_ + timeStep * gravityk1 / 2;
    Vector gravityk2 = gravityVector(planet.mass, centerPlanetMass, planet.r_ + timeStep * velocityk1
/ 2) / planet.mass;

    planet.v_ += gravityk2 * timeStep;
    planet.r_ += velocityk2 * timeStep;
}
/// <summary>
/// calculates gravity for all the different methods.
/// </summary>
/// <param name="mass1"></param>
/// <param name="mass2"></param>
/// <param name="position"></param>
/// <returns></returns>
public static Vector gravityVector(double mass1, double mass2, Vector position)
{
    Vector gravity = new Vector();
    double G = 1;
    double distance = position.magnitude();
    gravity.x = -G * mass1 * mass2 * position.x / Math.Pow(distance, 3);
    gravity.y = -G * mass1 * mass2 * position.y / Math.Pow(distance, 3);
}

```

```

        gravity.z = -G * mass1 * mass2 * position.z / Math.Pow(distance, 3);
        return gravity;
    }
}
}

```

Vectors.cs

```

using System;
using System.Collections.Generic;
using System.Text;

namespace IntegrationMethodApplication
{
    public class Vector
    {
        public double x;
        public double y;
        public double z;

        public Vector(double xLength, double yLength, double zLength)
        {
            this.x = xLength;
            this.y = yLength;
            this.z = zLength;
        }

        public Vector(double angleXY, double angleZ, double magnitude, bool use)
        {
            x = Math.Cos(angleXY) * magnitude;
            y = Math.Sin(angleXY) * magnitude;
            z = Math.Sin(angleZ - (Math.PI / 2)) * magnitude;
        }

        /// <summary>
        /// default constructor
        /// </summary>
        public Vector() { }

        /// <summary>
        /// addition operator
        /// </summary>
        /// <param name="vectOne"></param>
        /// <param name="vectTwo"></param>
        /// <returns></returns>
        public static Vector operator +(Vector vectOne, Vector vectTwo)
        {
            Vector newVect = new Vector();
            newVect.x = vectOne.x + vectTwo.x;
            newVect.y = vectOne.y + vectTwo.y;
            newVect.z = vectOne.z + vectTwo.z;
            return newVect;
        }
    }
}

```

```

/// <summary>
/// subtraction operator
/// </summary>
/// <param name="vectOne"></param>
/// <param name="vectTwo"></param>
/// <returns></returns>
public static Vector operator -(Vector vectOne, Vector vectTwo)
{
    Vector newVect = new Vector();
    newVect.x = vectOne.x - vectTwo.x;
    newVect.y = vectOne.y - vectTwo.y;
    newVect.z = vectOne.z - vectTwo.z;
    return newVect;
}

/// <summary>
/// scaling operator
/// </summary>
/// <param name="vectOne"></param>
/// <param name="num"></param>
/// <returns></returns>
public static Vector operator / (Vector vectOne, double num)
{
    Vector newVect = new Vector();
    newVect.x = vectOne.x / num;
    newVect.y = vectOne.y / num;
    newVect.z = vectOne.z / num;
    return newVect;
}

/// <summary>
/// scaling operator
/// </summary>
/// <param name="num"></param>
/// <param name="vectOne"></param>
/// <returns></returns>
public static Vector operator *(double num, Vector vectOne)
{
    Vector newVect = new Vector();
    newVect.x = vectOne.x * num;
    newVect.y = vectOne.y * num;
    newVect.z = vectOne.z * num;
    return newVect;
}

/// <summary>
/// scaling operator
/// </summary>
/// <param name="vectOne"></param>
/// <param name="num"></param>
/// <returns></returns>
public static Vector operator *(Vector vectOne, double num)
{
    Vector newVect = new Vector();
    newVect.x = vectOne.x * num;
    newVect.y = vectOne.y * num;
    newVect.z = vectOne.z * num;
    return newVect;
}

```

```

    }
    /// <summary>
    /// magnitude function
    /// </summary>
    /// <returns></returns>
    public double magnitude()
    {
        return Math.Sqrt(x * x + y * y + z * z);
    }
    /// <summary>
    /// dot product function
    /// </summary>
    /// <param name="vectOne"></param>
    /// <param name="VectTwo"></param>
    /// <returns></returns>
    public static double dotProduct(Vector vectOne, Vector VectTwo)
    {
        return ((vectOne.x * VectTwo.x) + (vectOne.y * VectTwo.y) + (vectOne.z * VectTwo.z));
    }
    /// <summary>
    /// cross product function
    /// </summary>
    /// <param name="vectOne"></param>
    /// <param name="VectTwo"></param>
    /// <returns></returns>
    public static Vector crossProduct(Vector vectOne, Vector VectTwo)
    {
        Vector newVect = new Vector();
        newVect.x = (vectOne.y * VectTwo.z) - (vectOne.z * VectTwo.y);
        newVect.y = (vectOne.z * VectTwo.x) - (vectOne.x * VectTwo.z);
        newVect.z = (vectOne.x * VectTwo.y) - (vectOne.y * VectTwo.x);
        return newVect;
    }
}
}

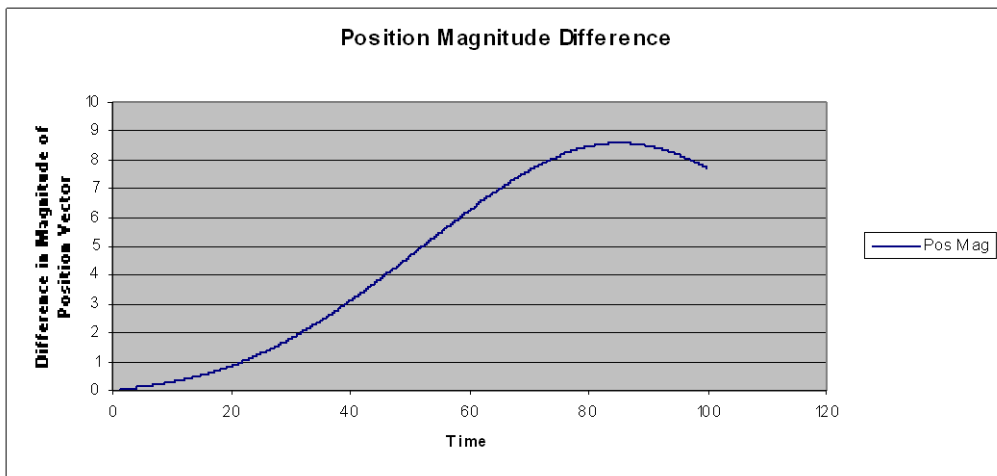
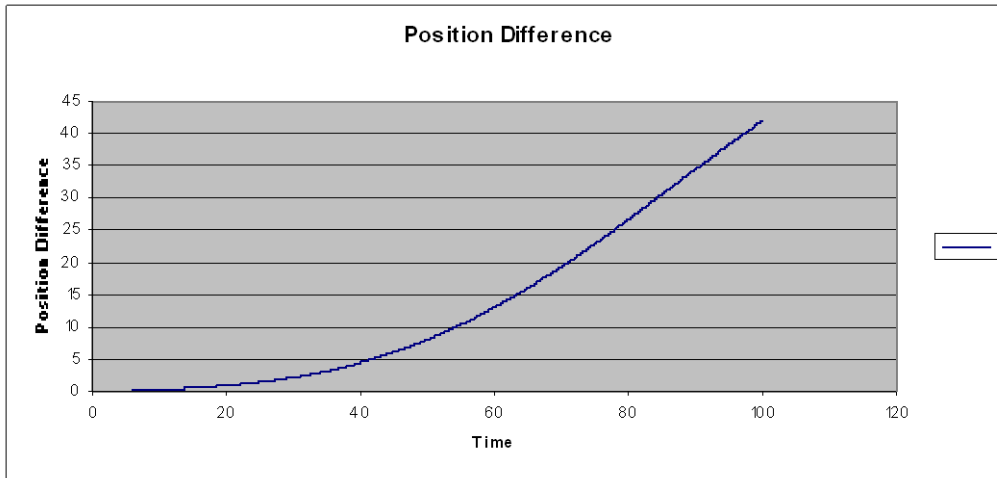
```

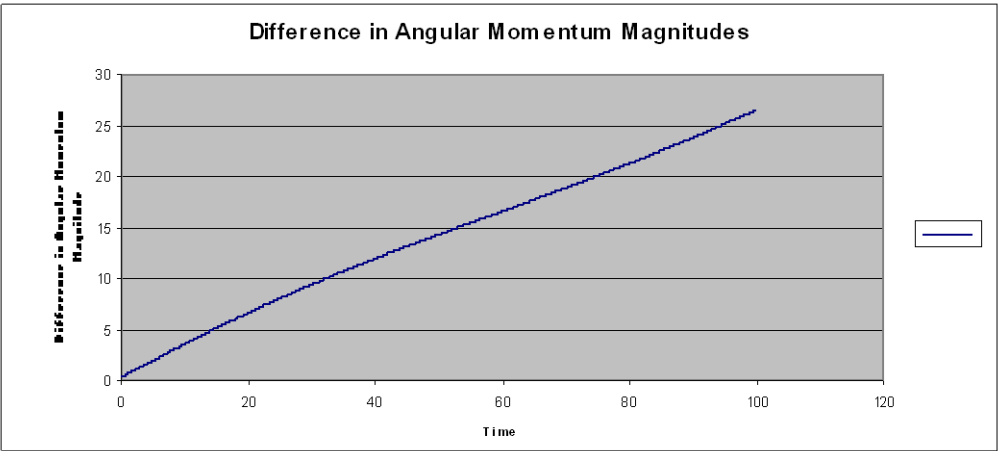
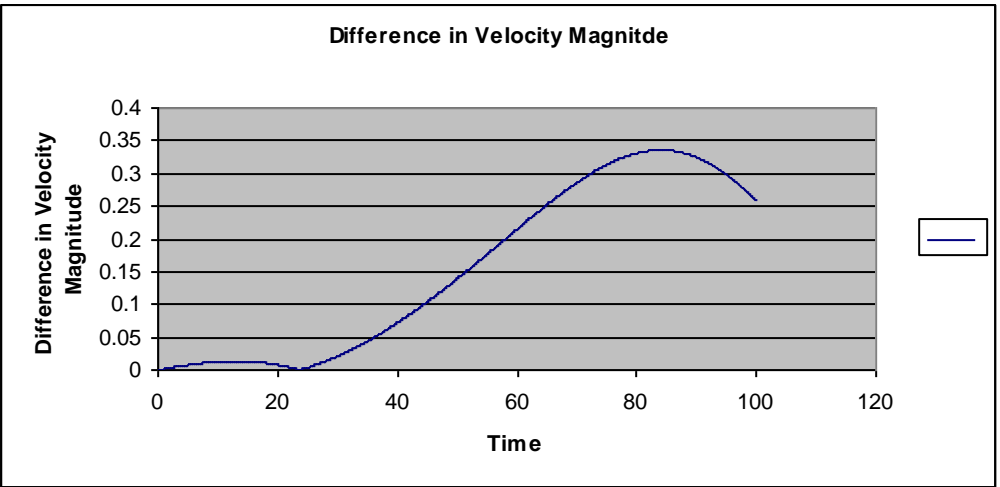
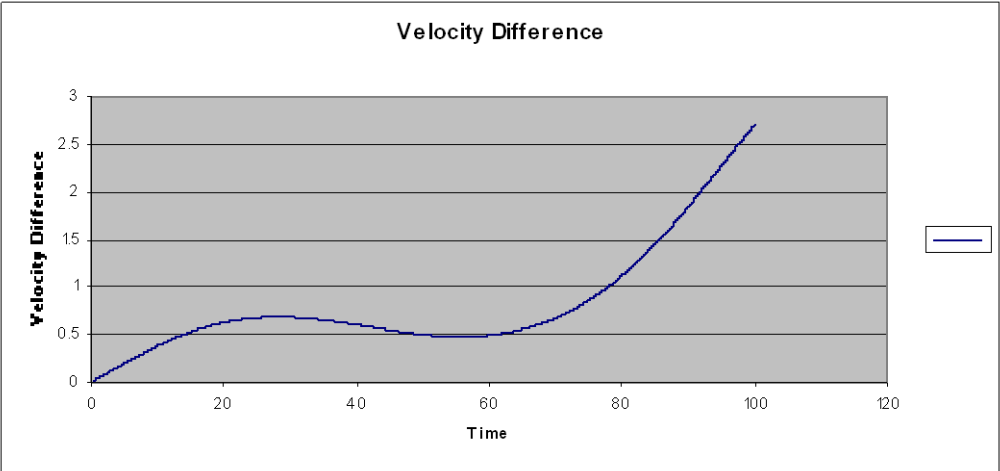

Appendix II: Additional Graphs

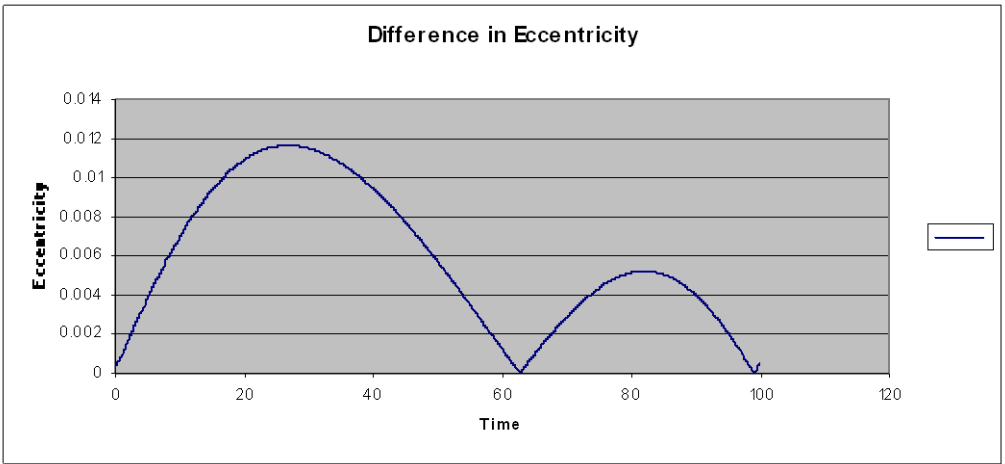
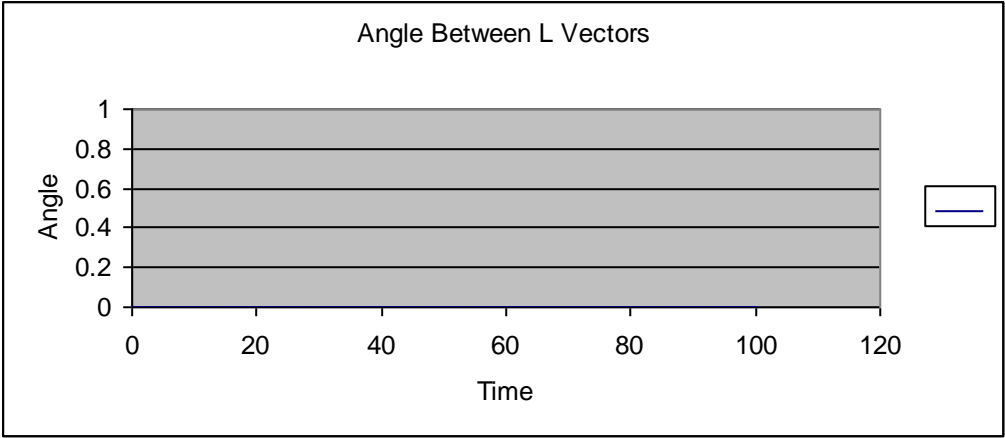
NOTE: Three sets of graphical data with are included with each method. The three sets are results from time steps 0.1,2, and 8 respectively. Within each set, there are 9 graphs that illustrate the conservation of various quantities.

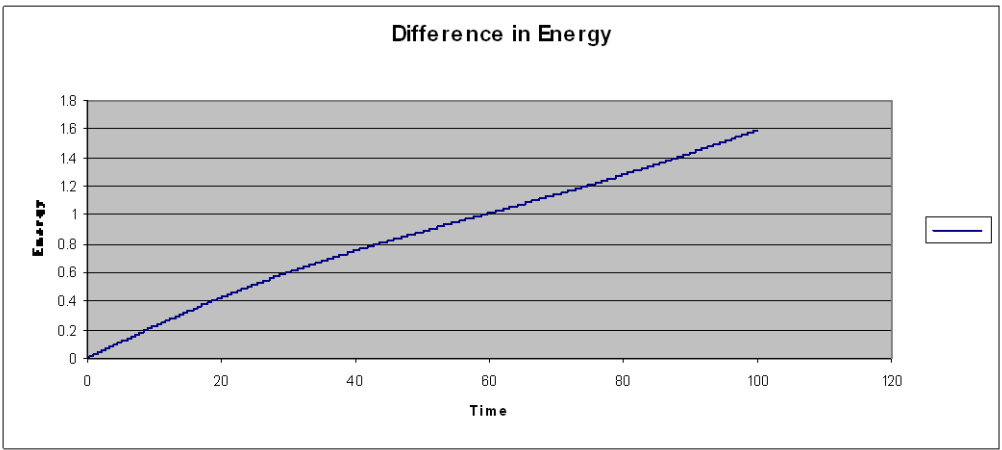
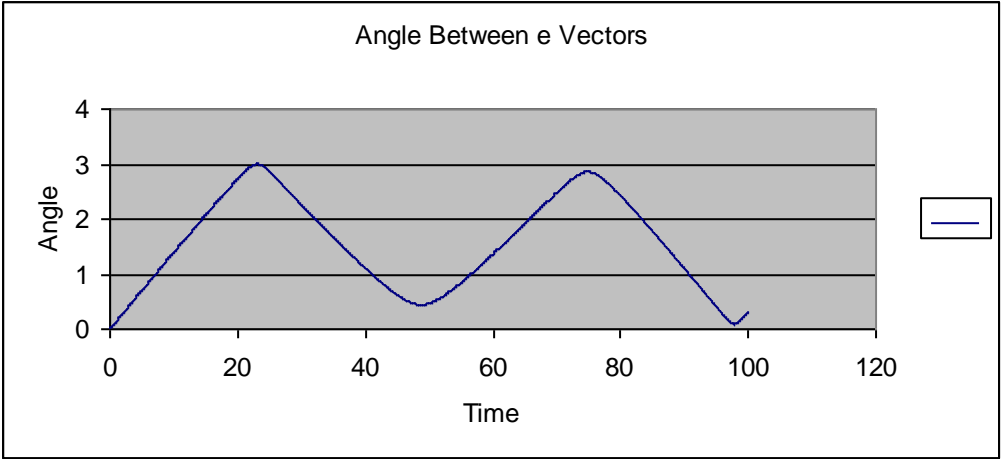
Euler's Method

Time Step: .1

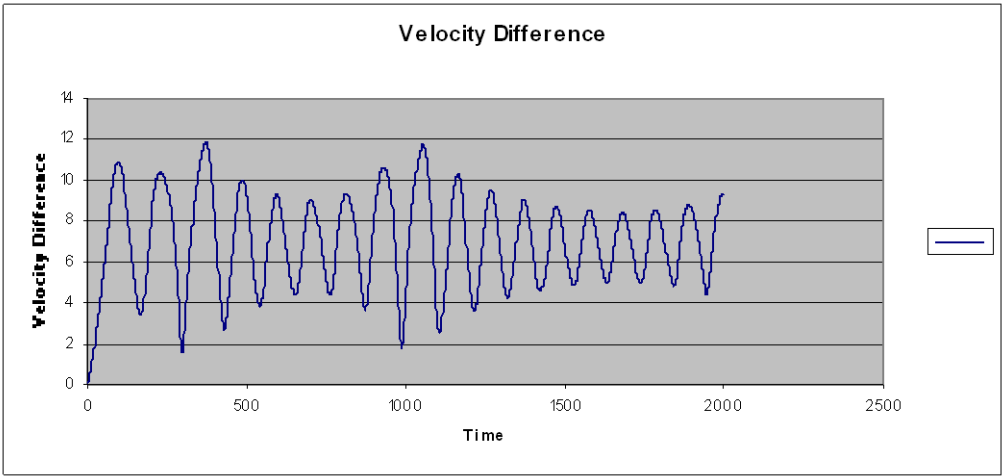
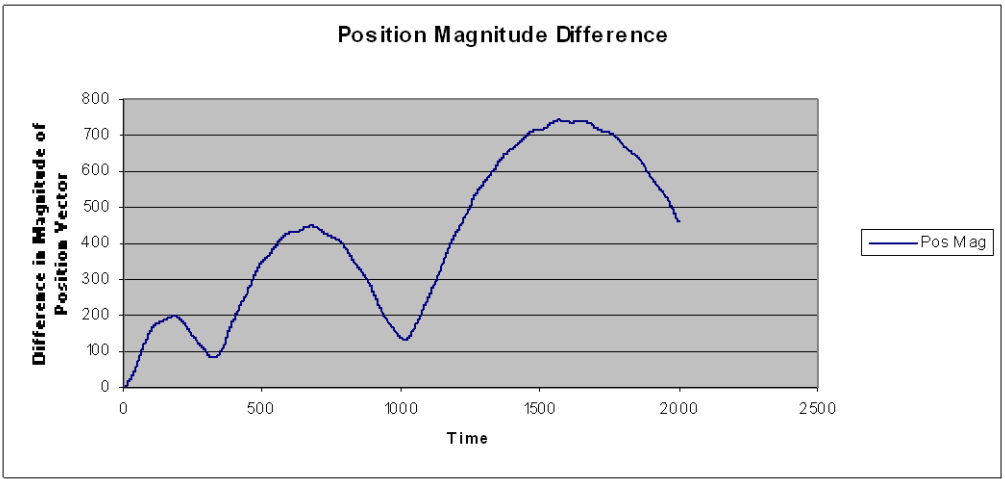
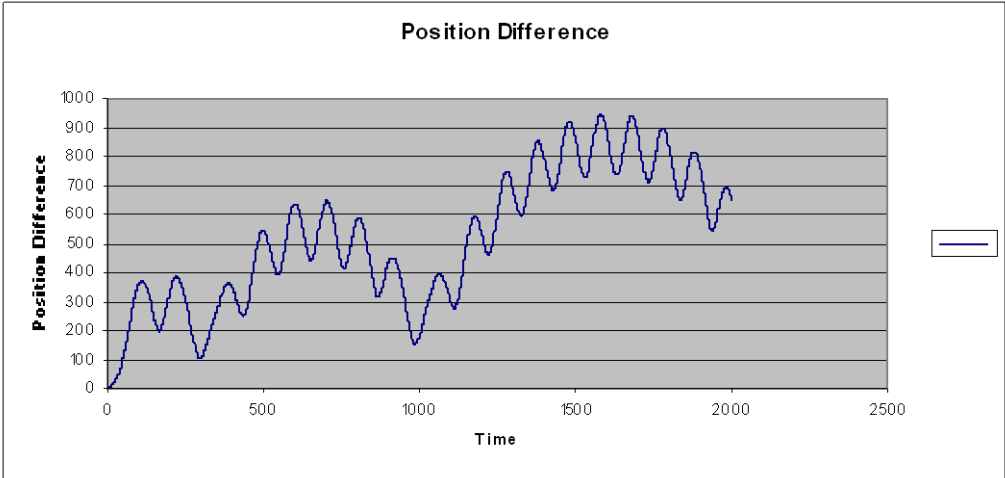


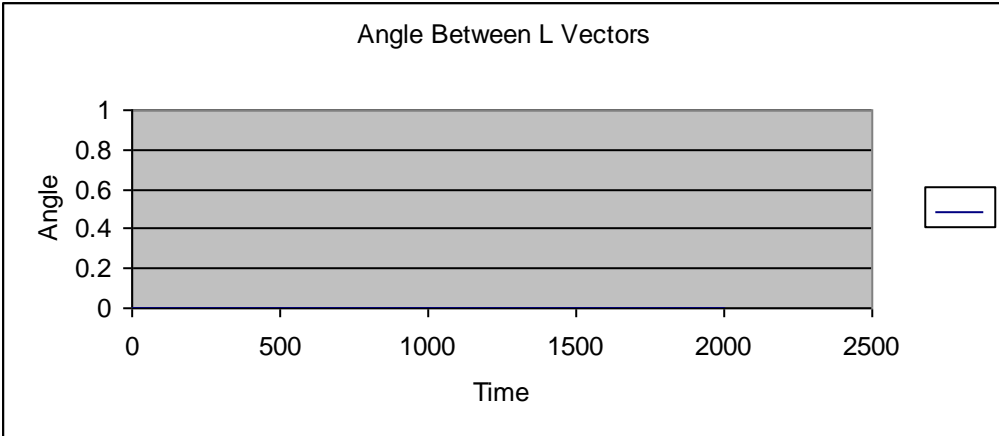
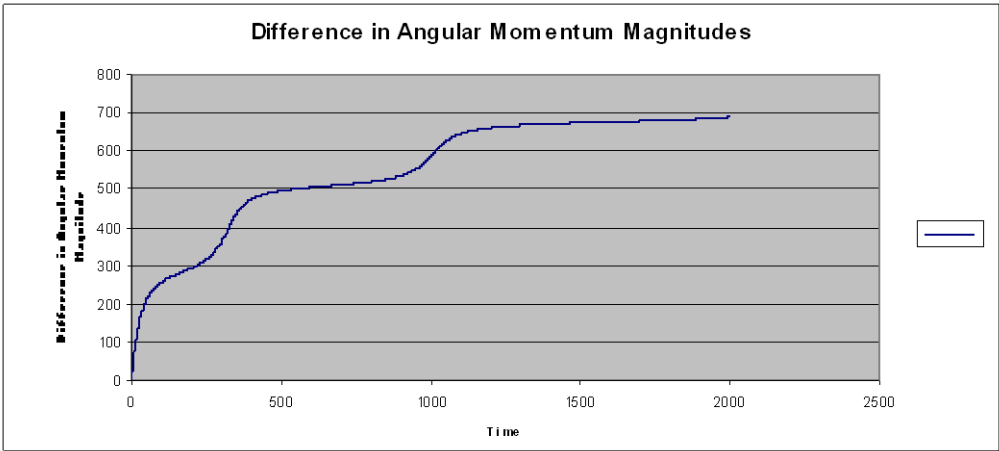
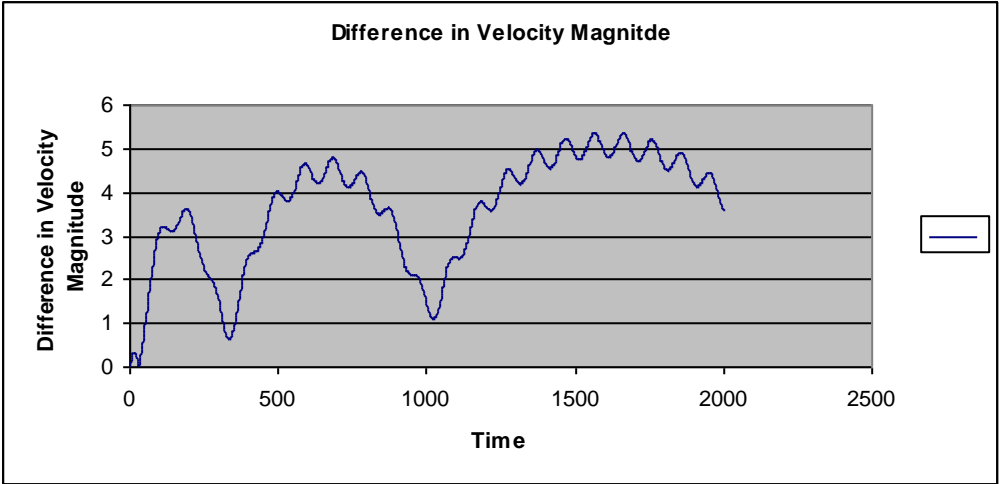


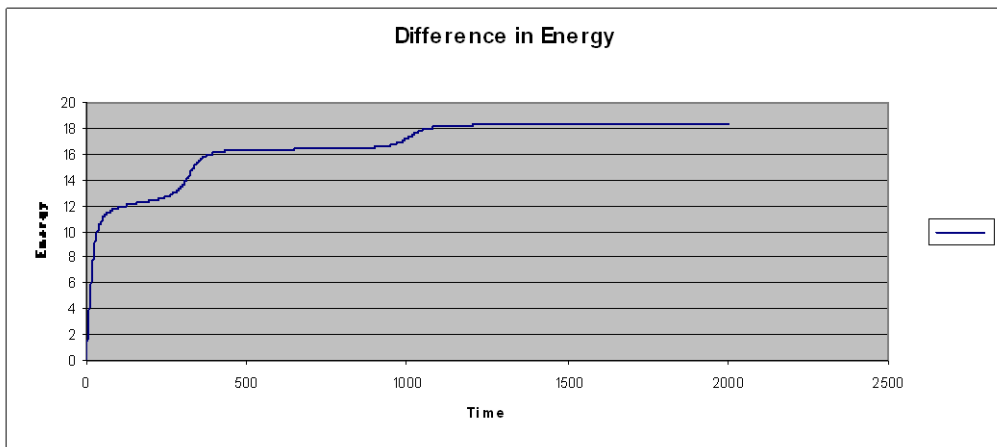
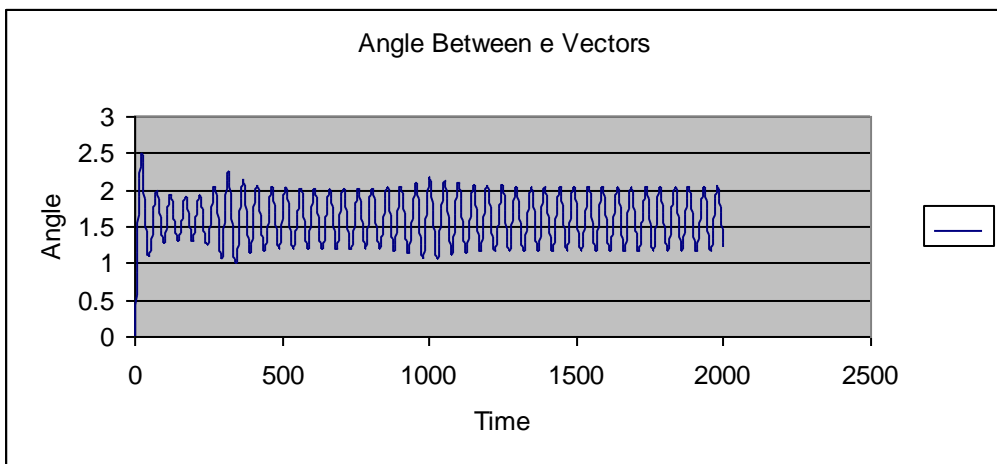
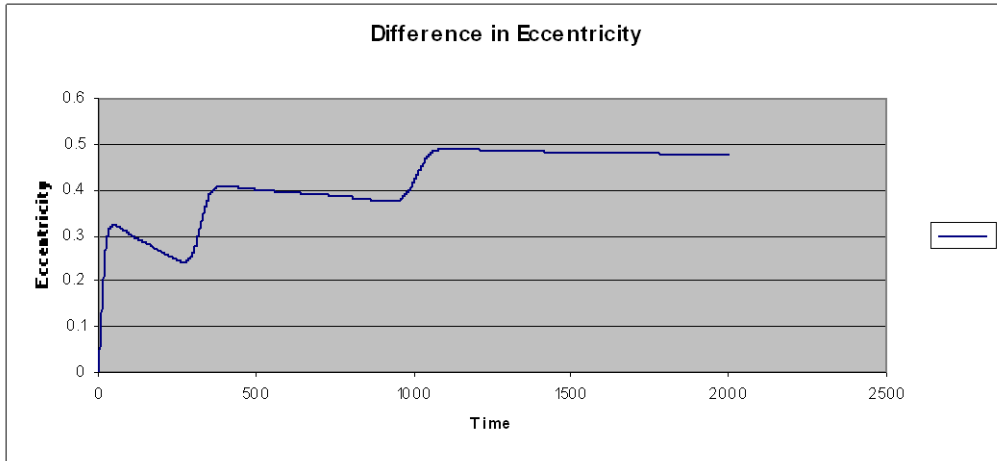




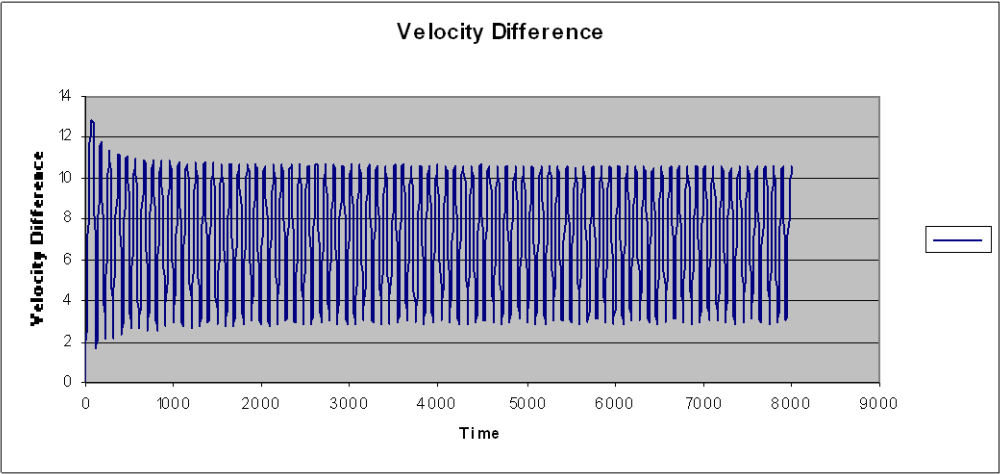
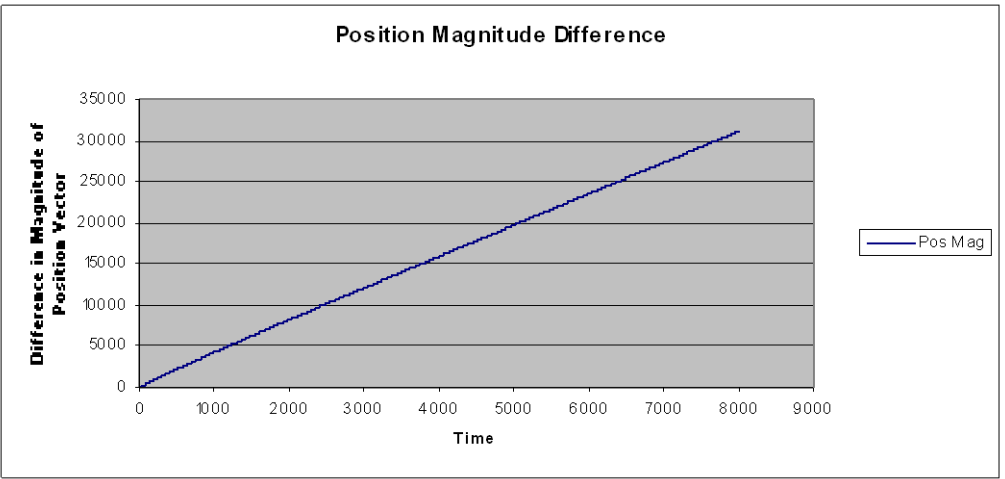
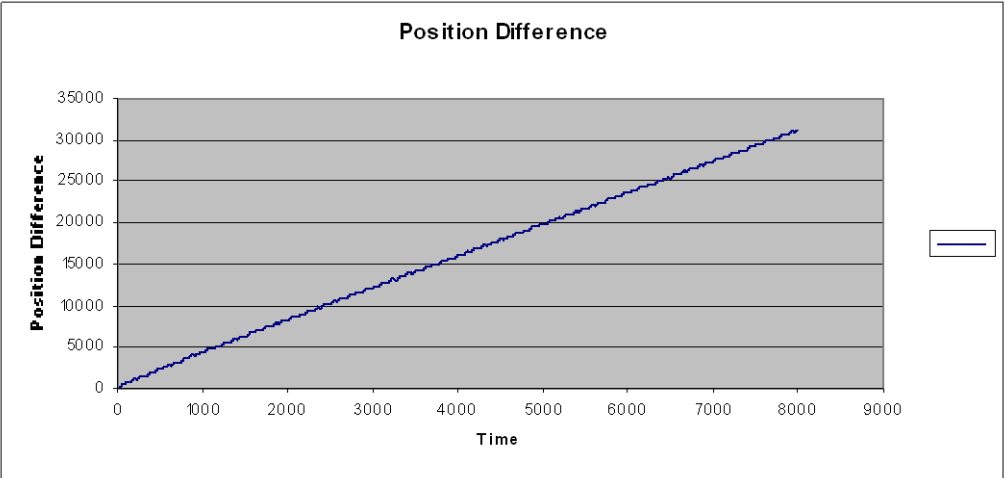
Time Step: 2

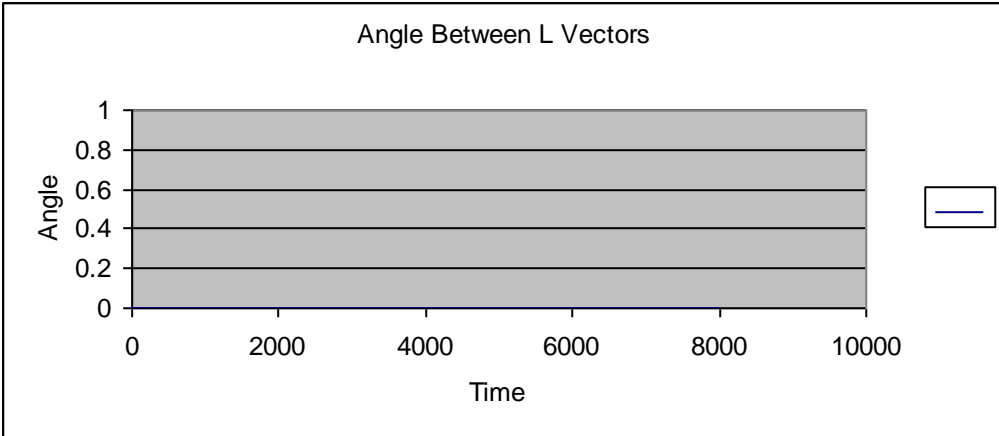
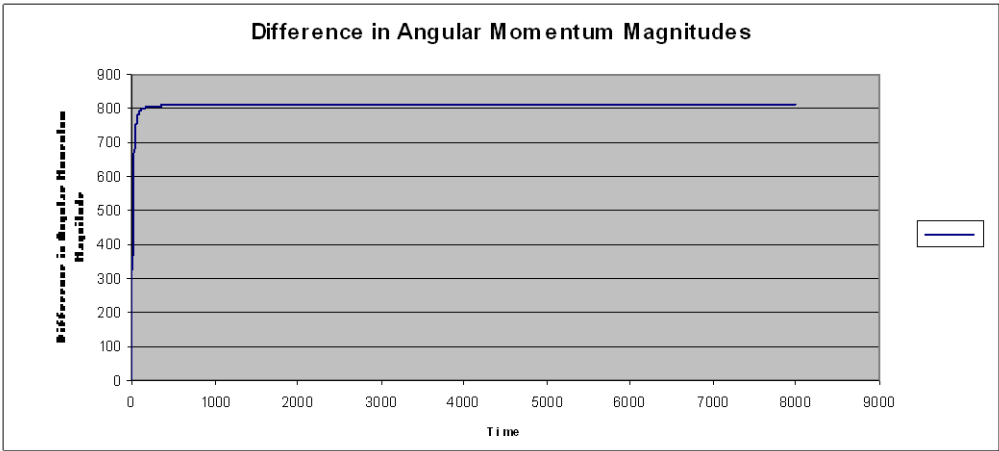
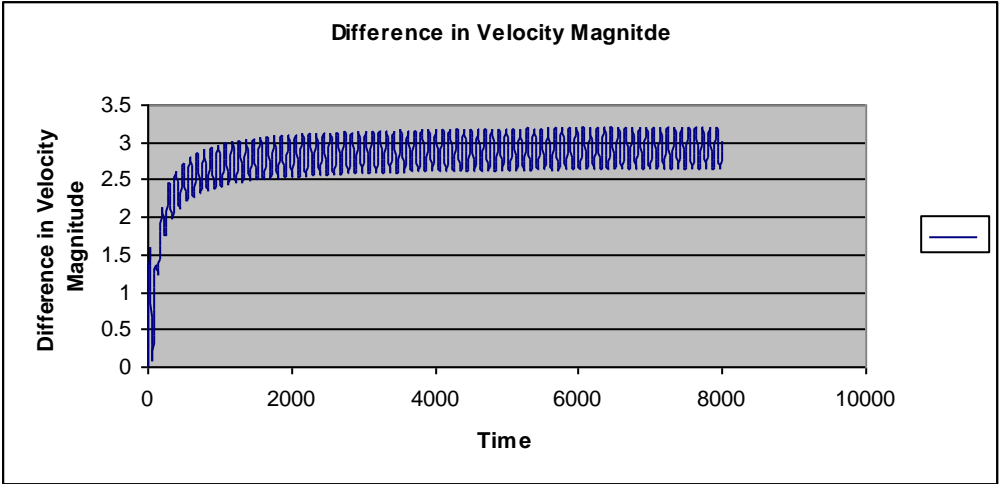


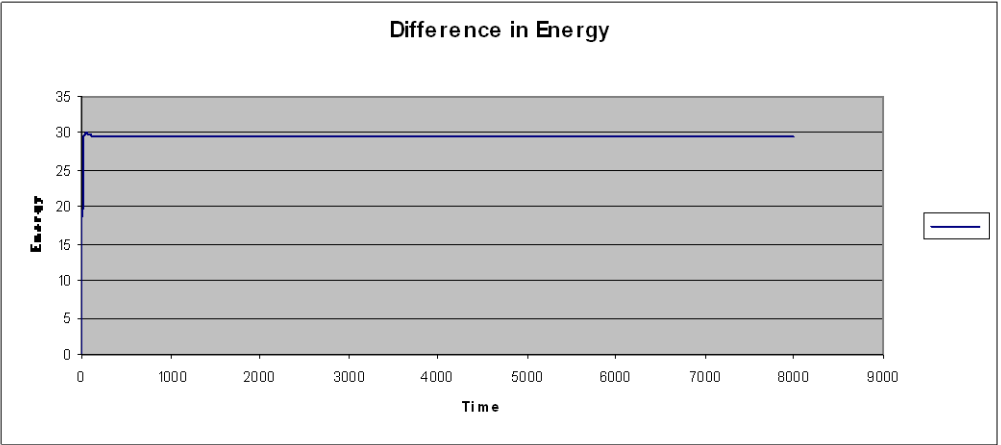
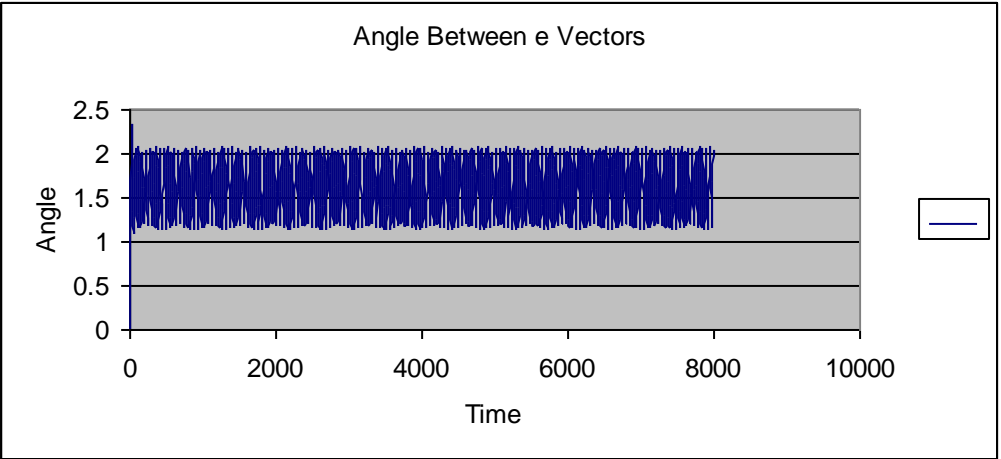
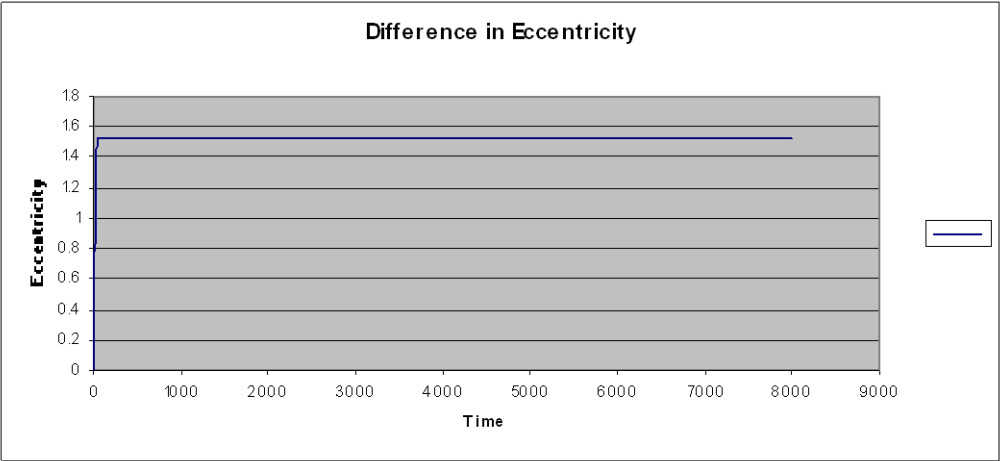




Time Step: 8

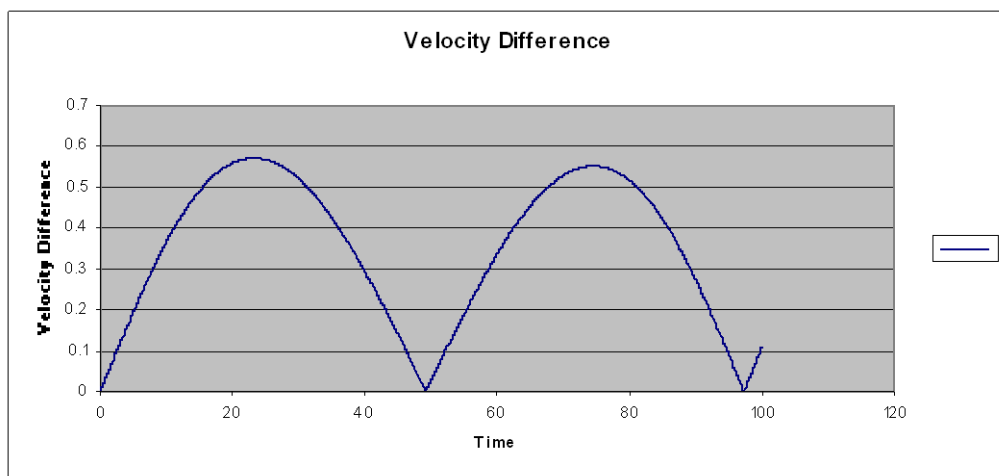
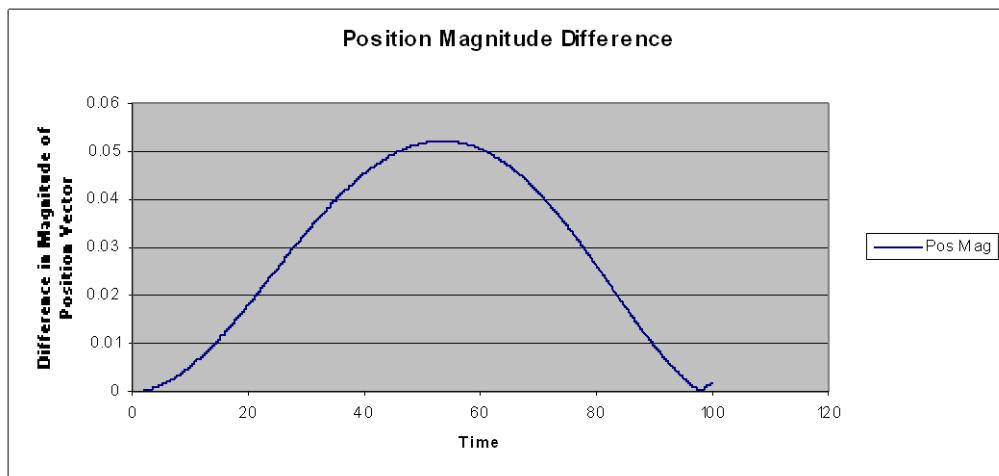
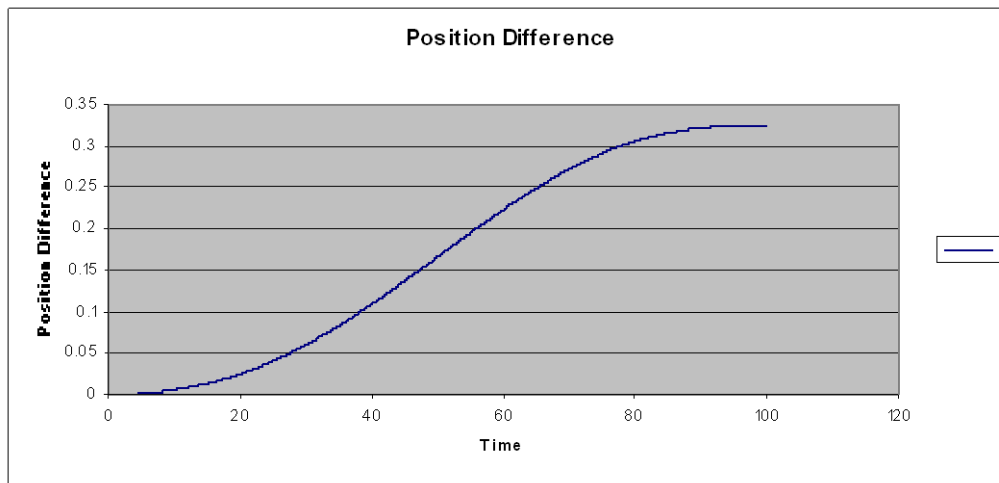


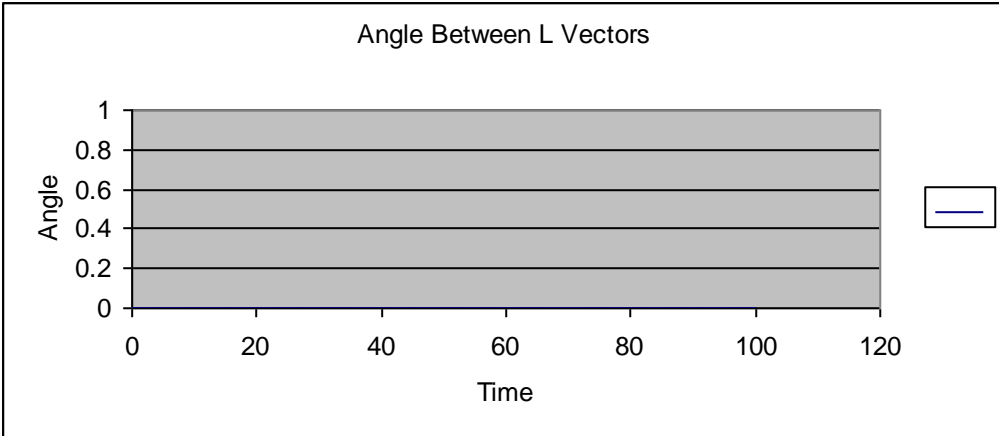
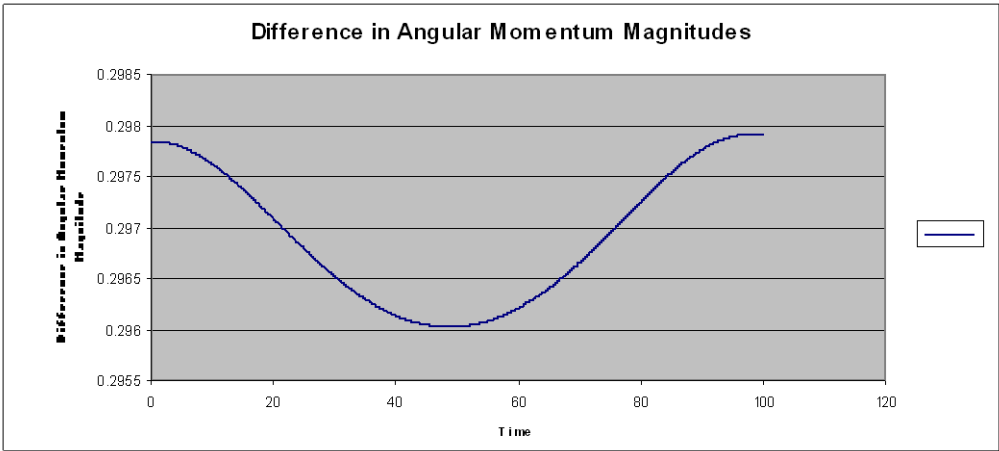
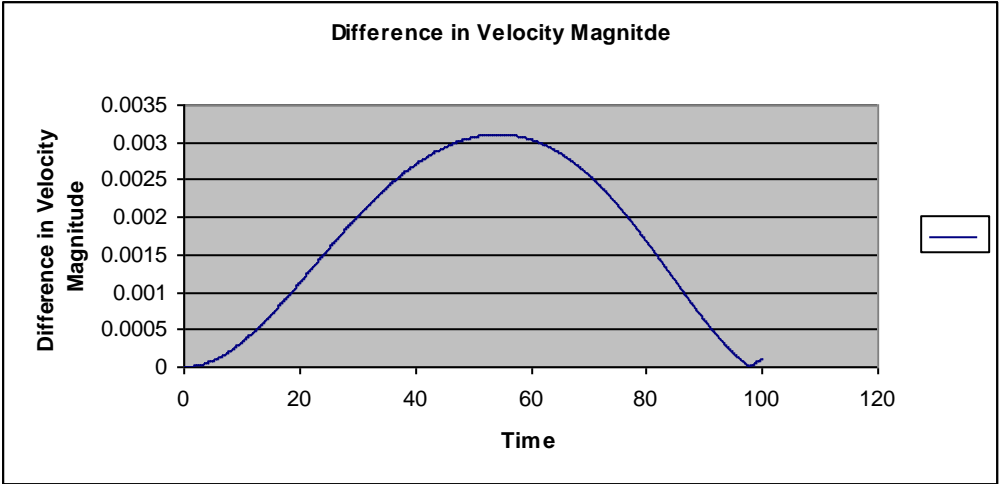


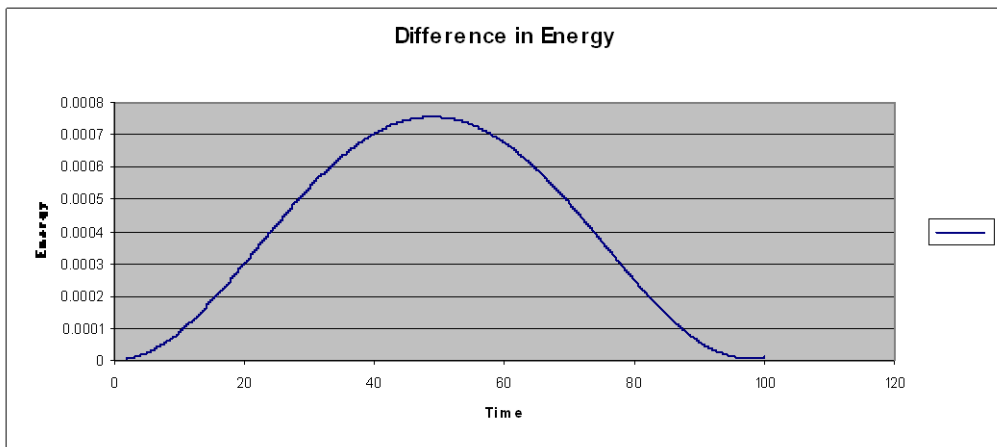
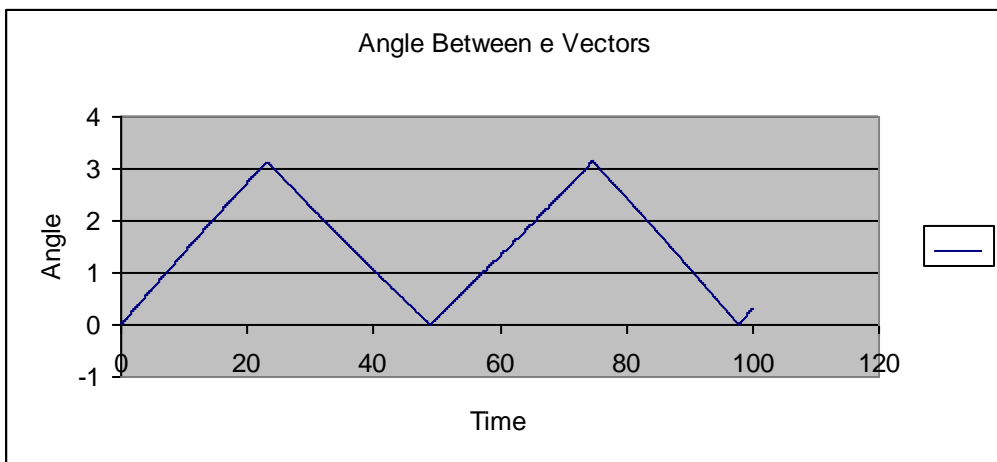
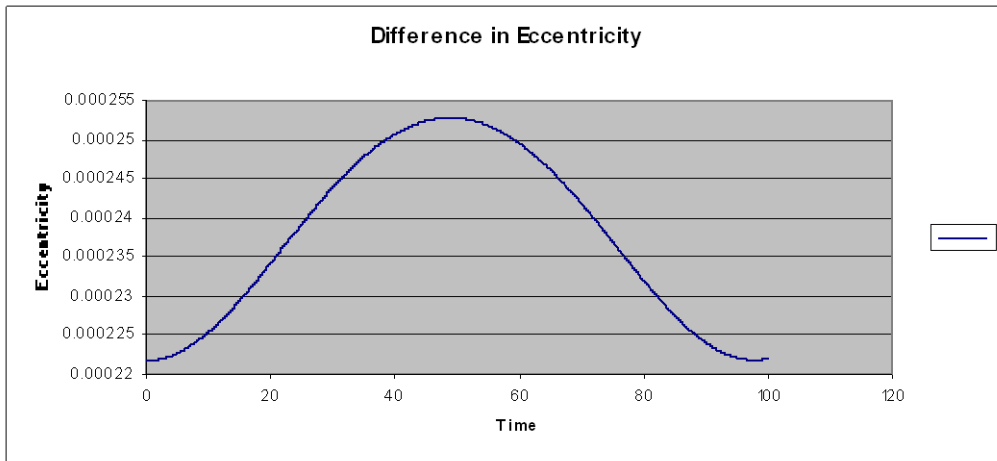


Runge-Kutta 2

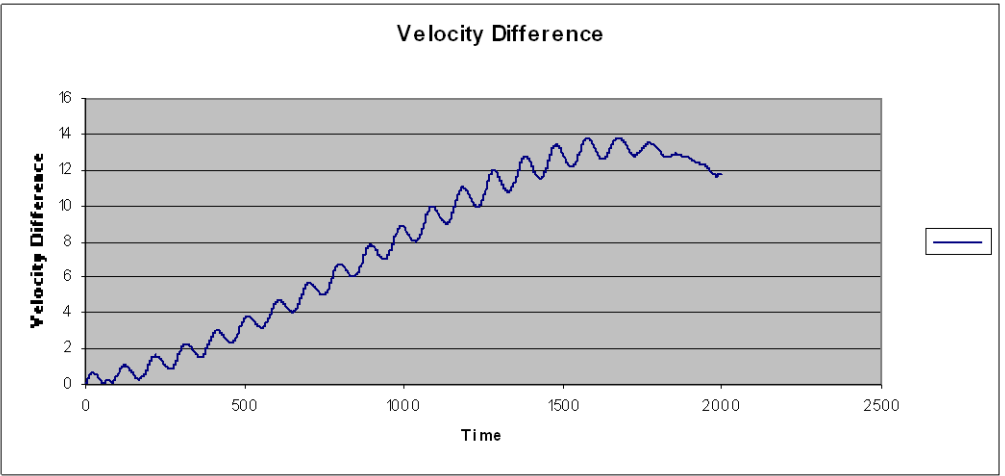
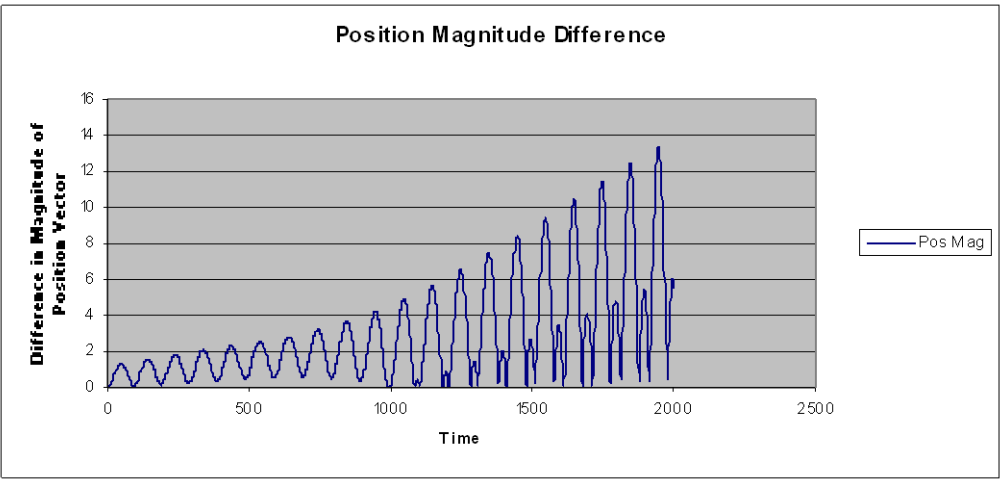
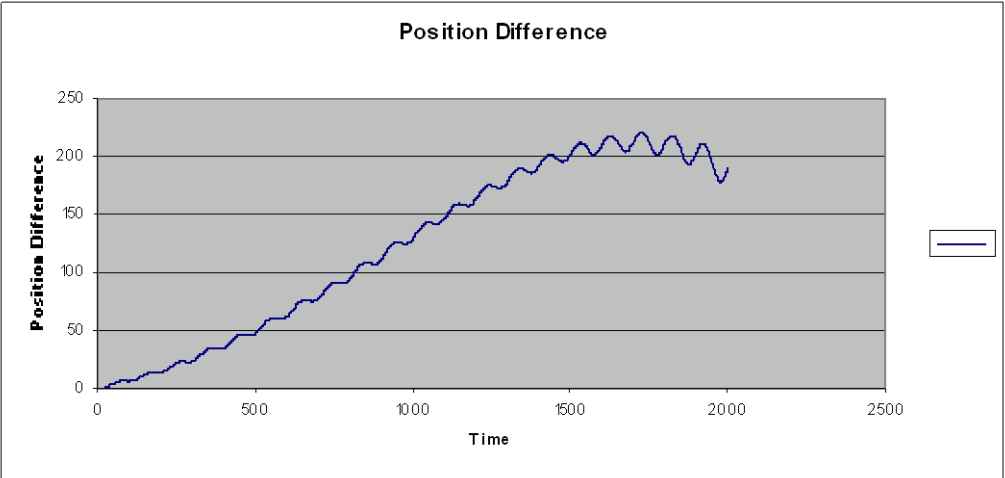
Time Step: .1

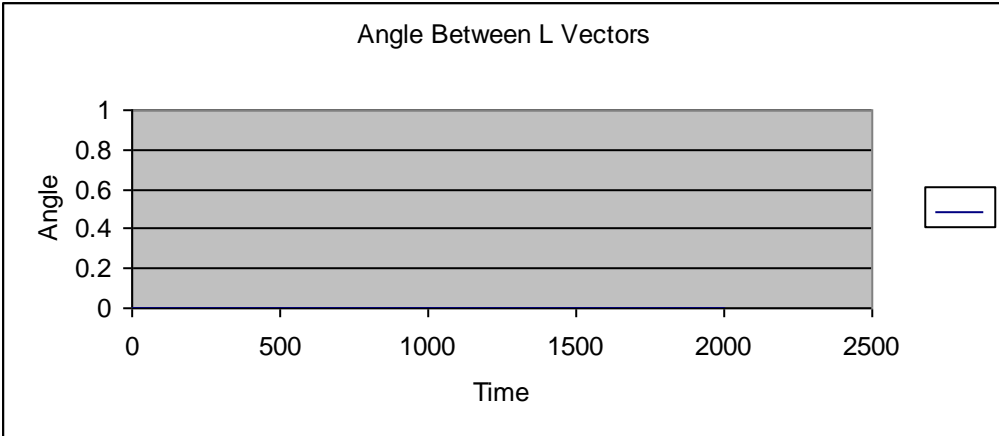
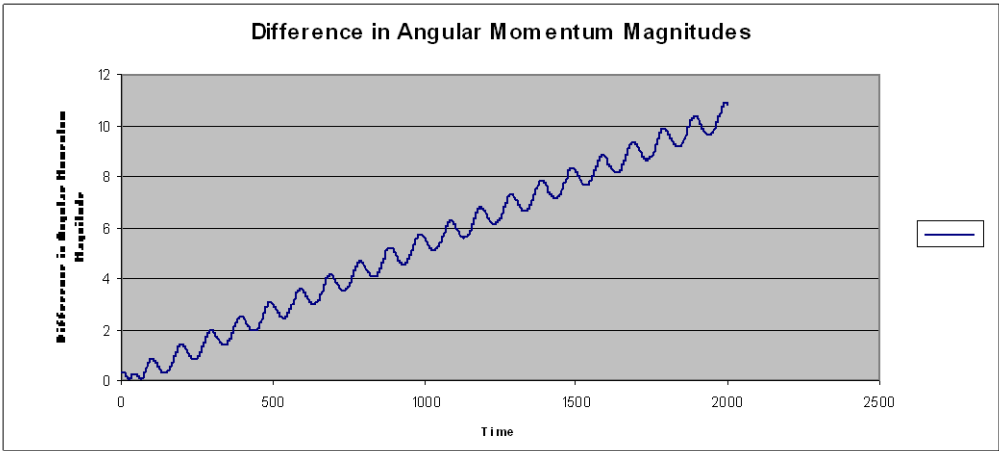
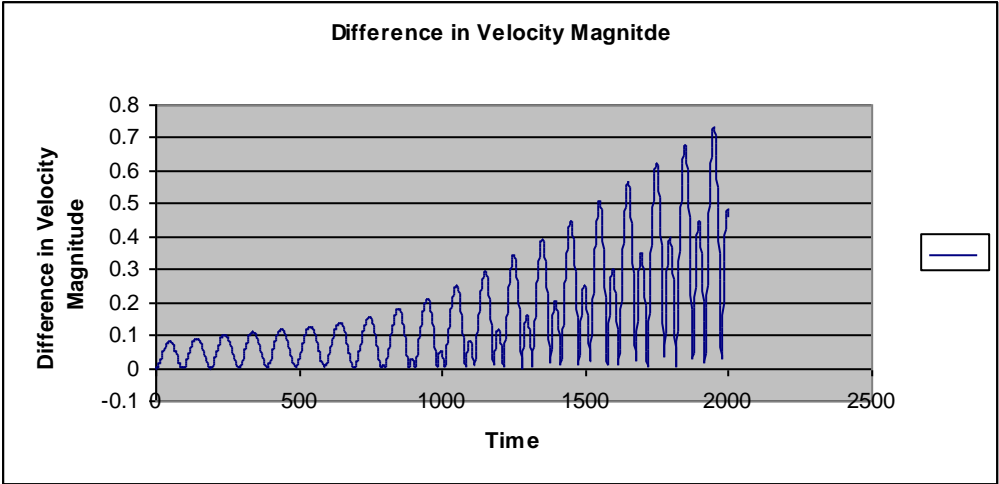


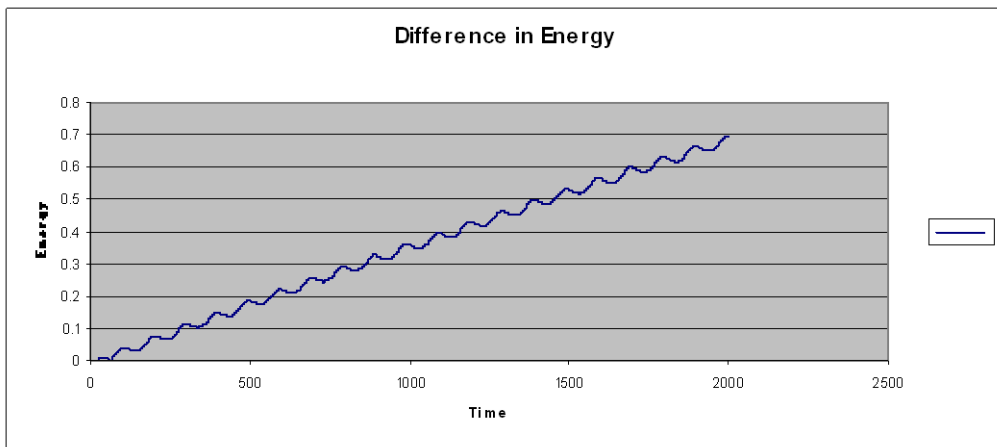
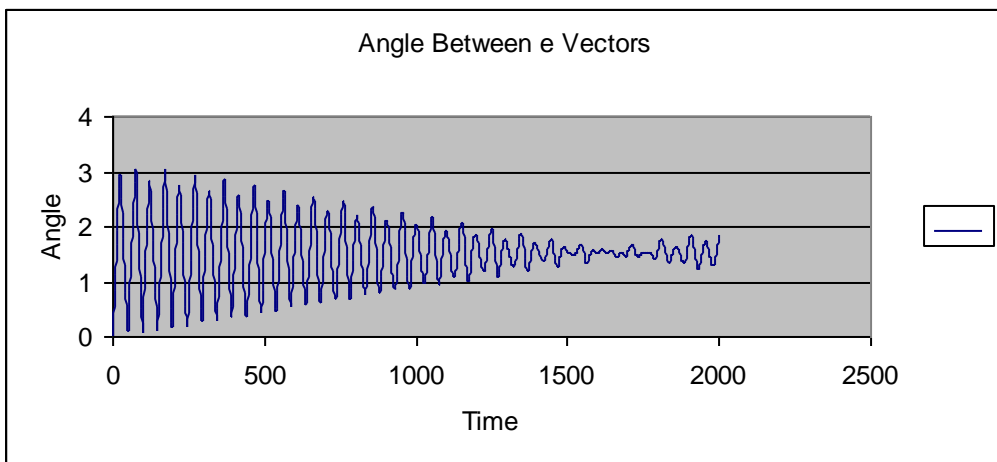
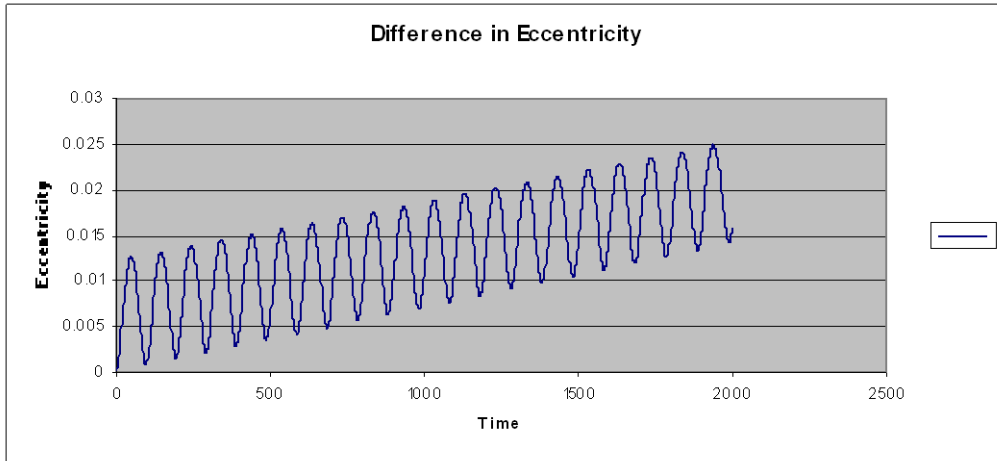




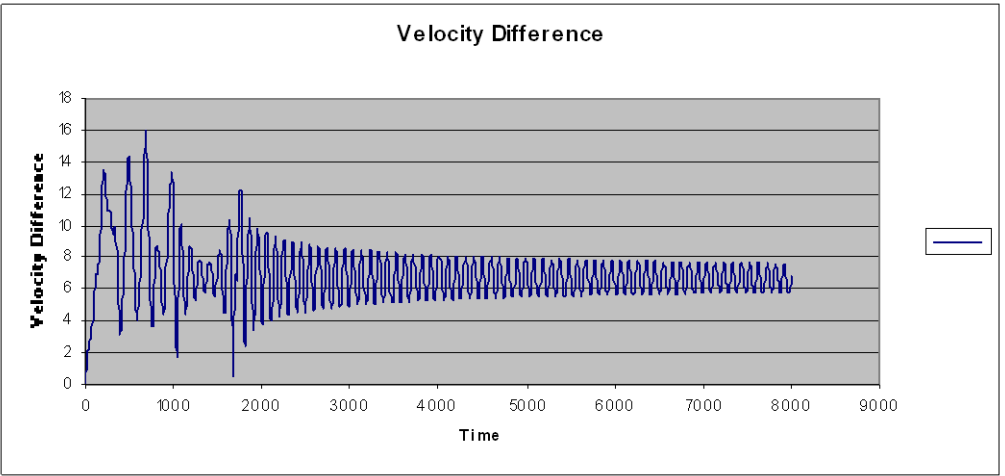
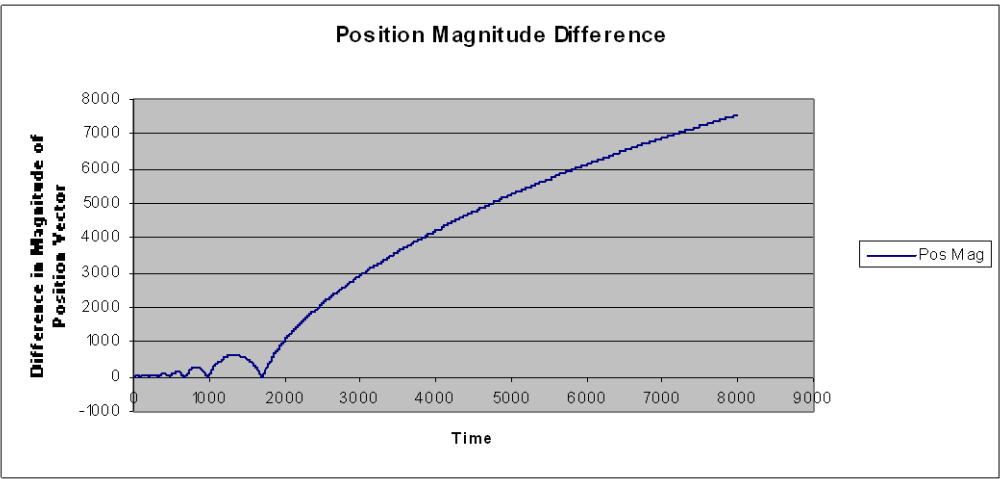
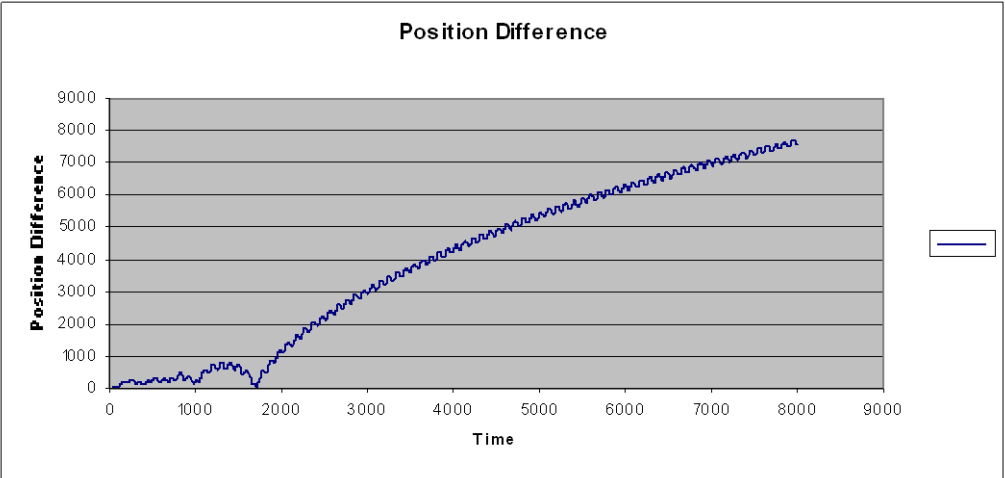
Time Step: 2

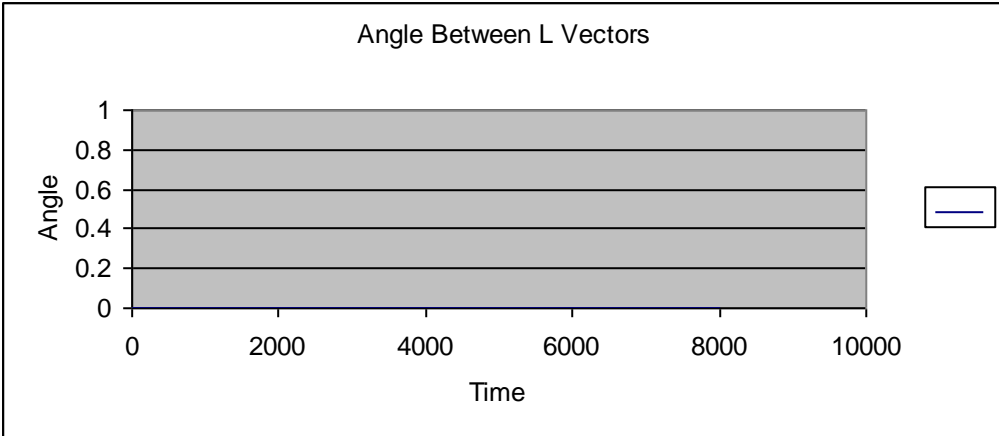
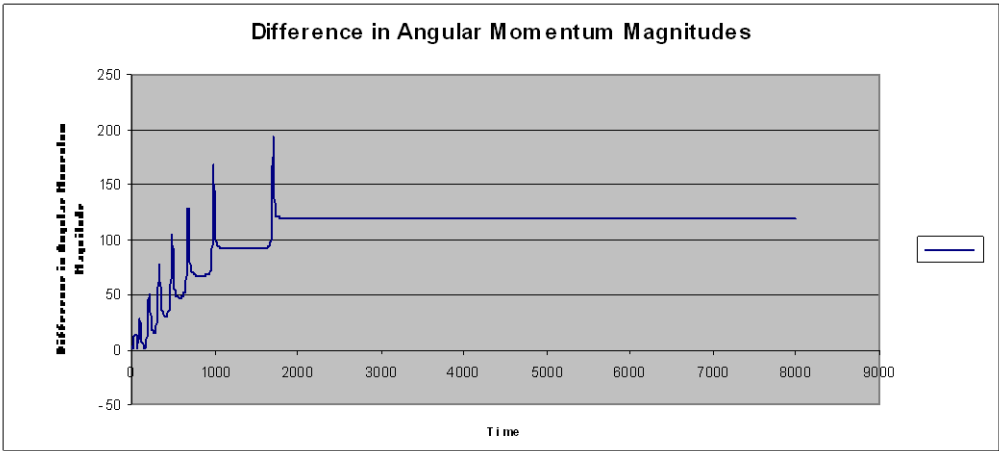
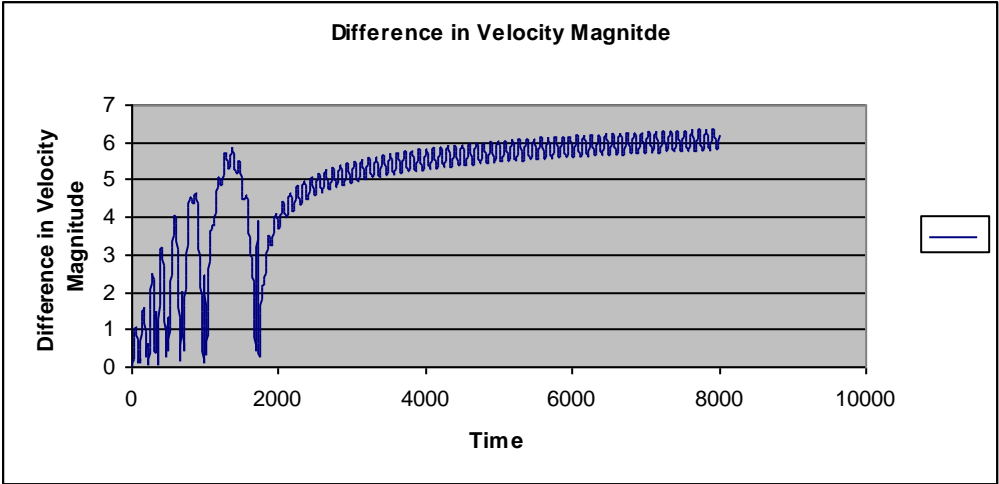


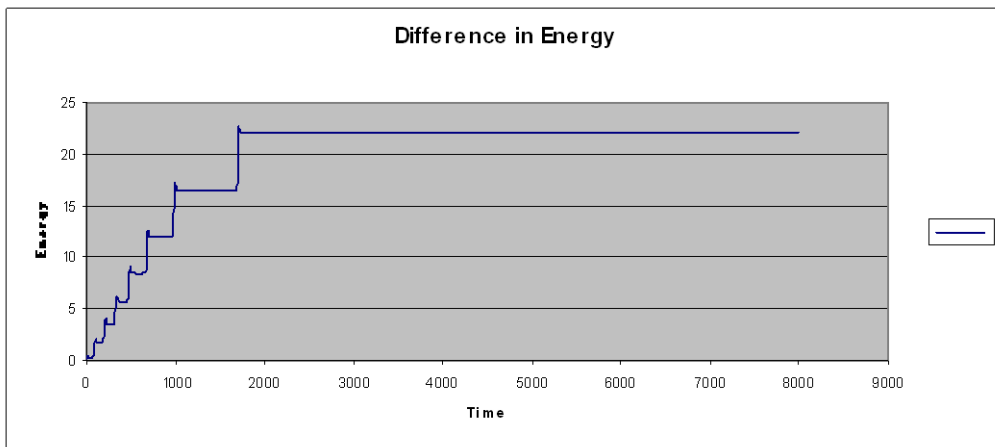
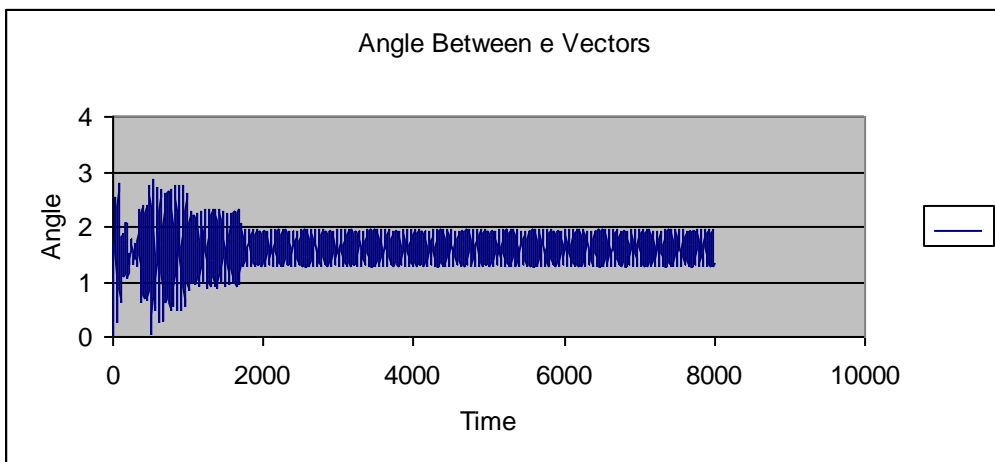
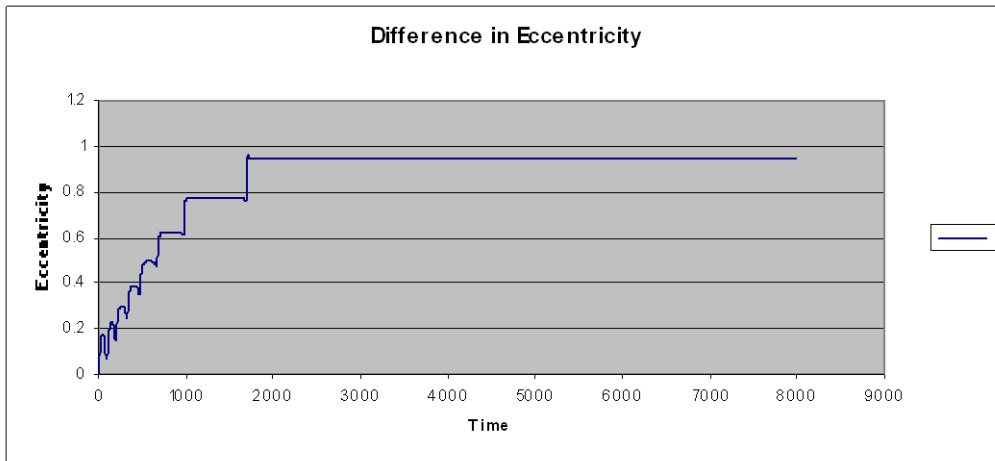




Time Step: 8

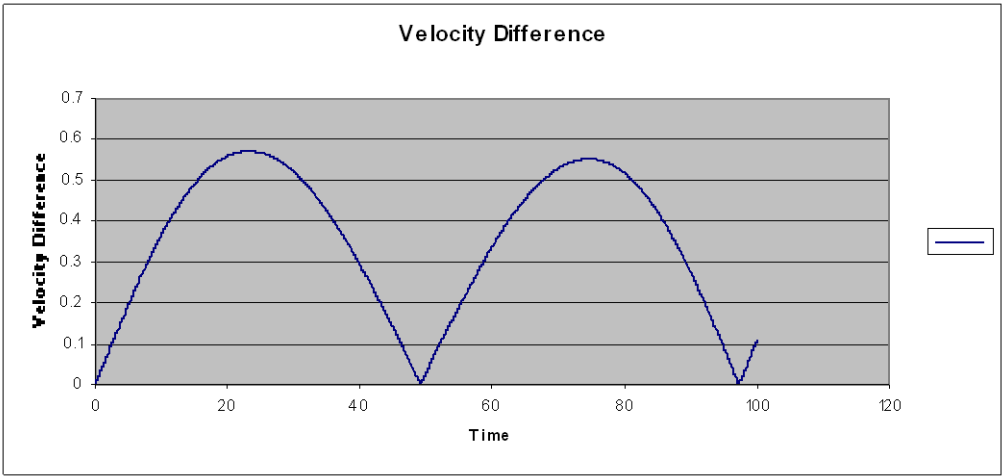
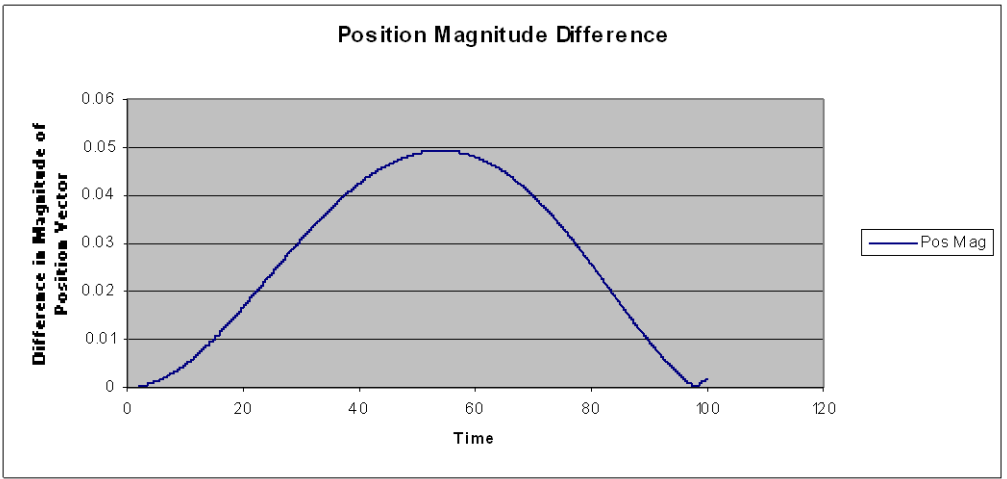


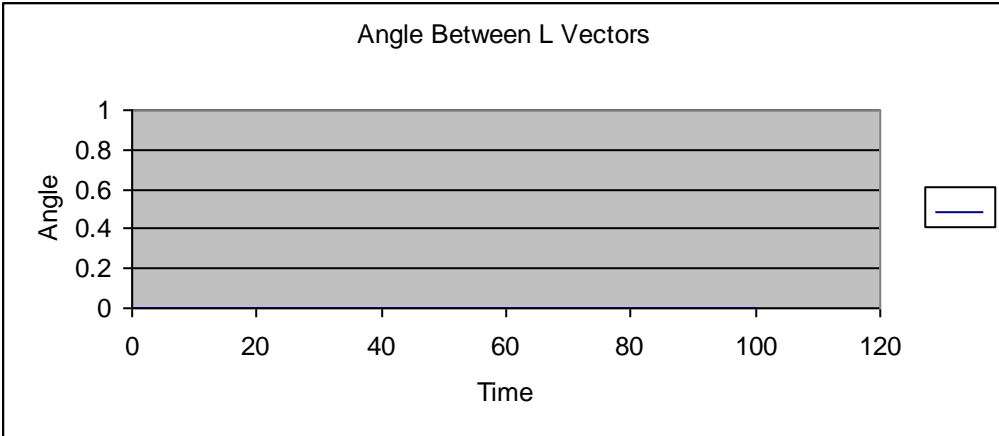
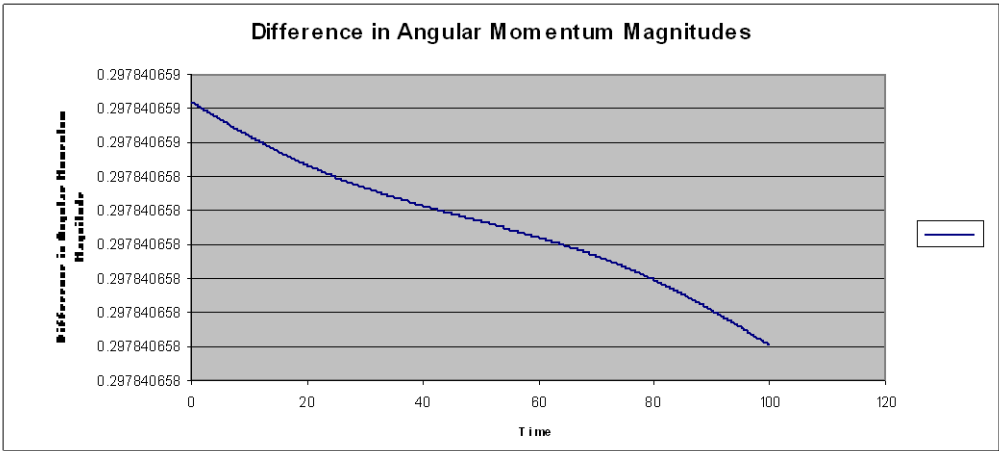
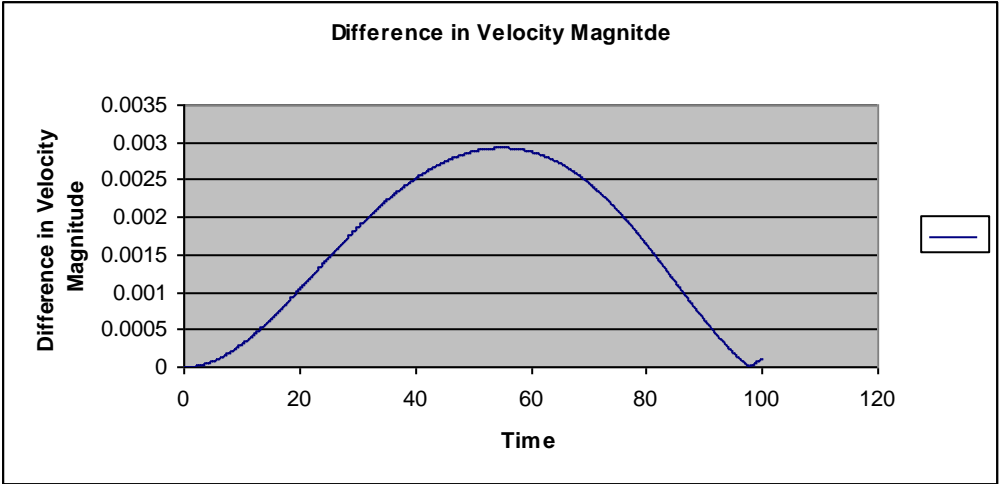


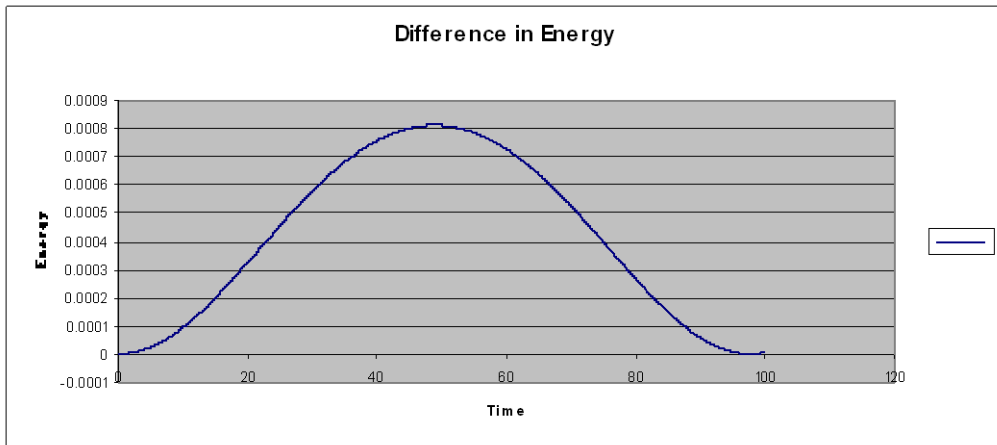
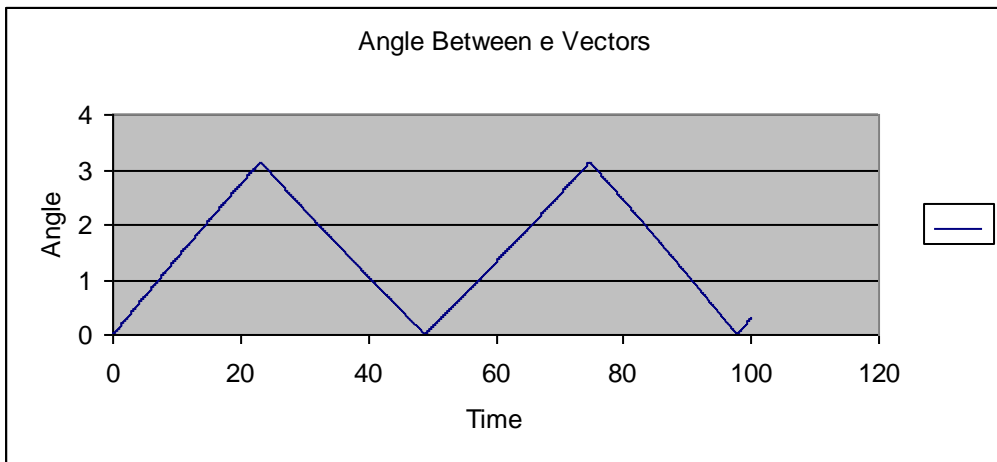
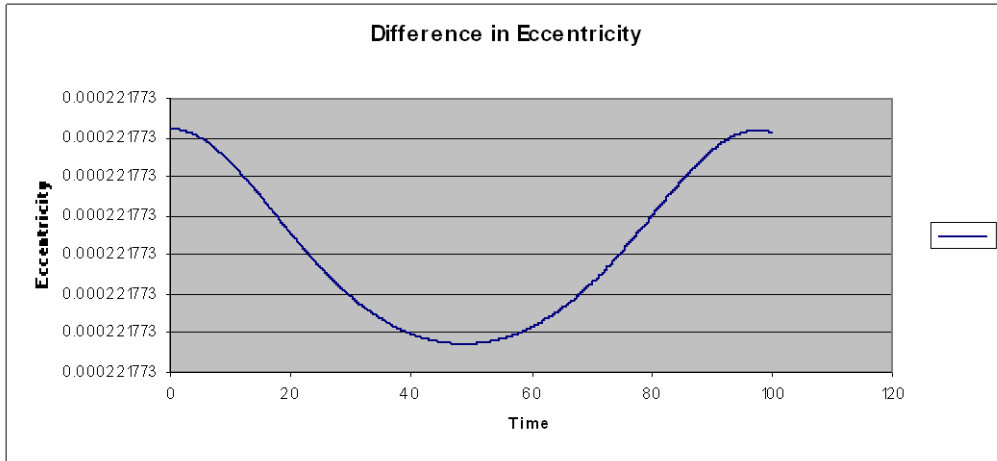


Runge-Kutta 4

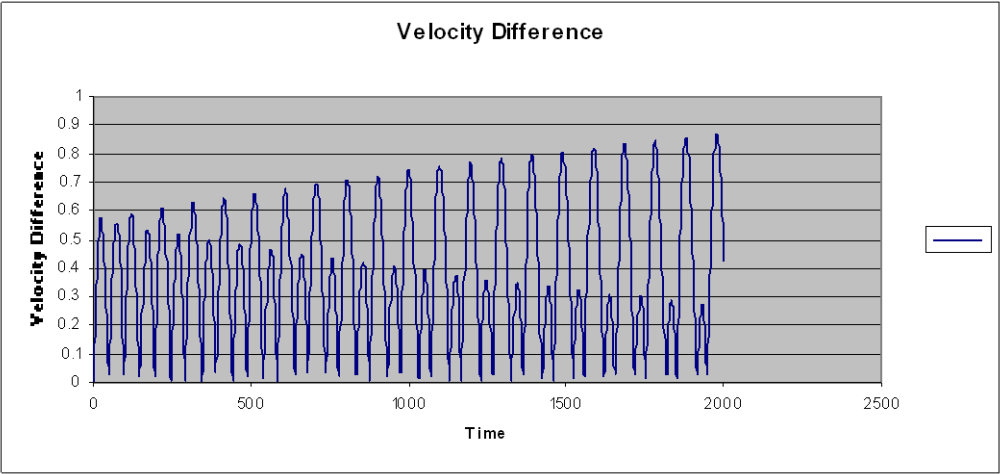
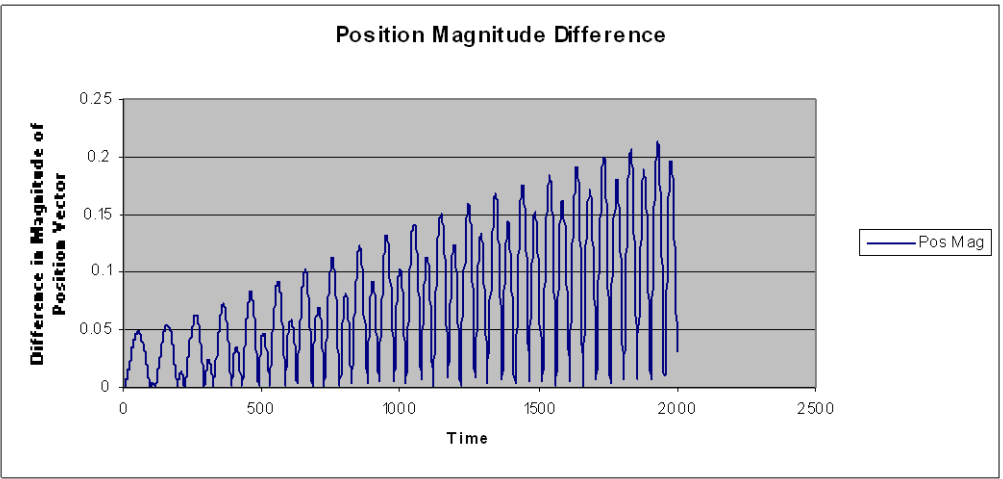
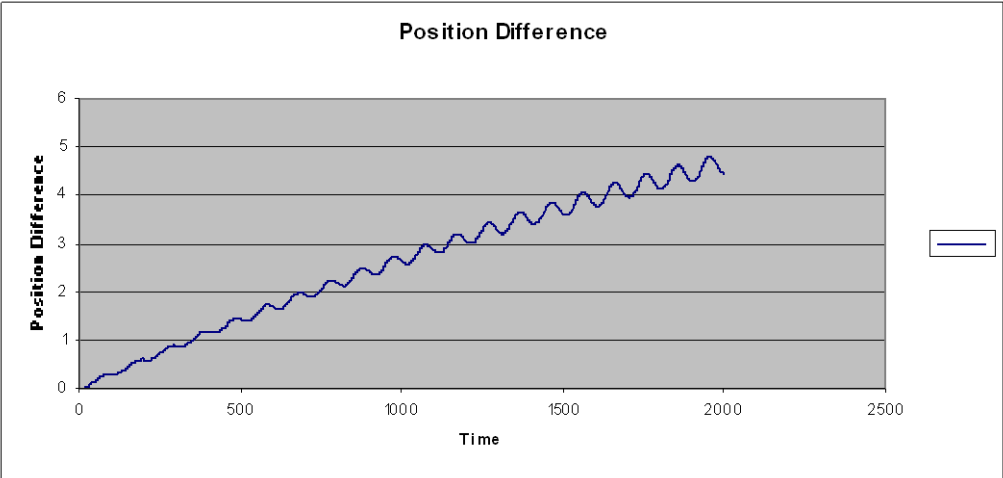
Time Step: .1

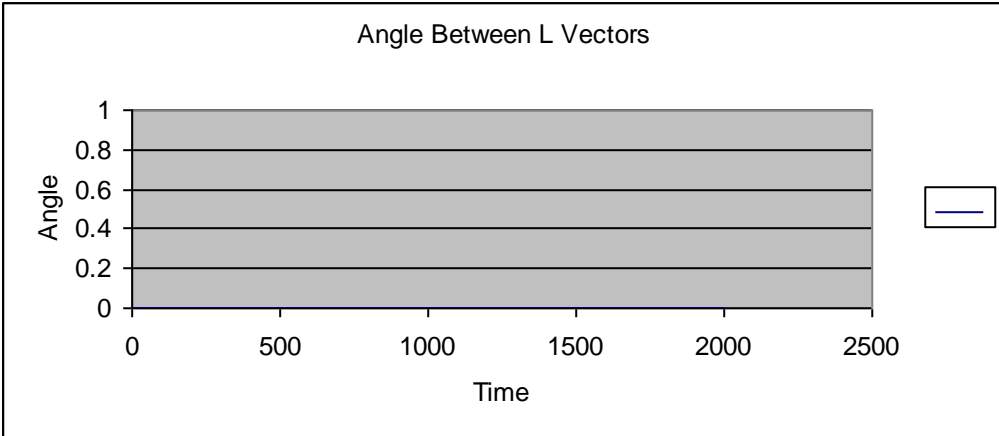
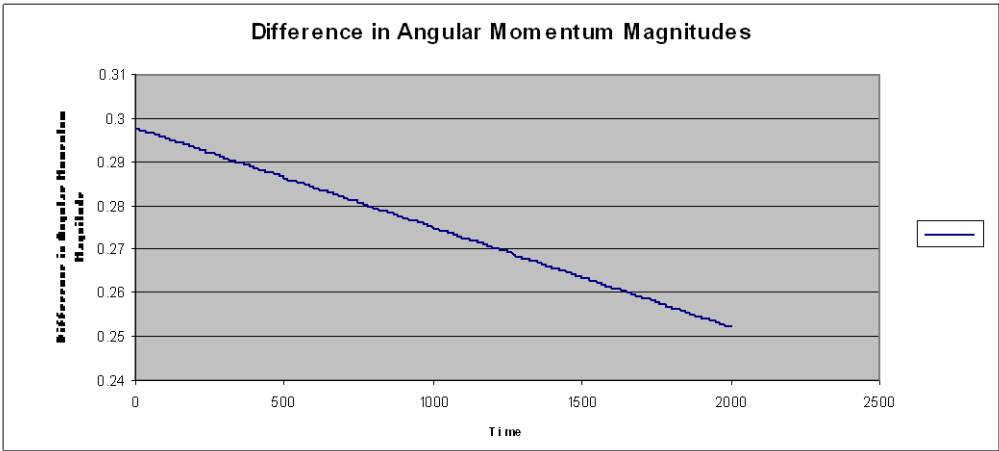
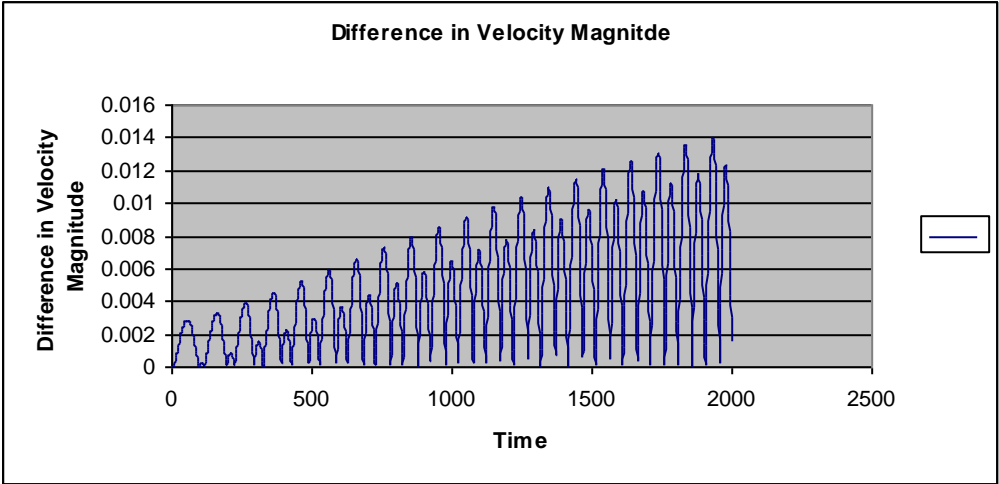


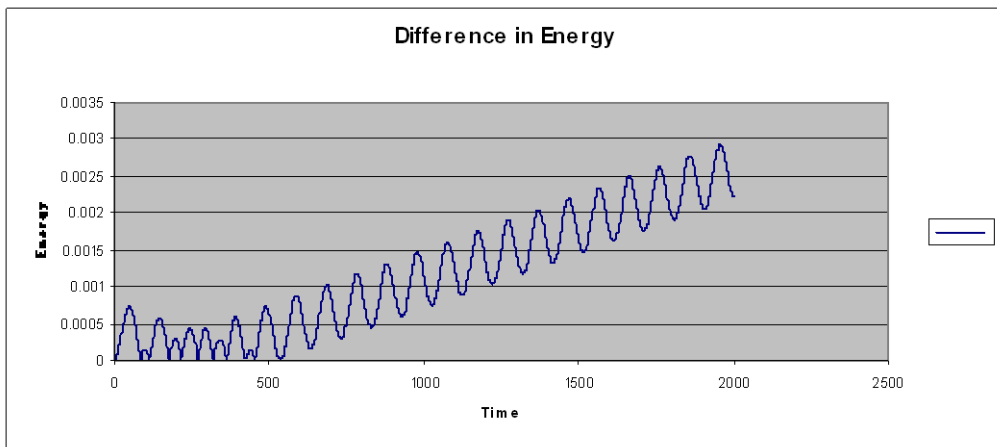
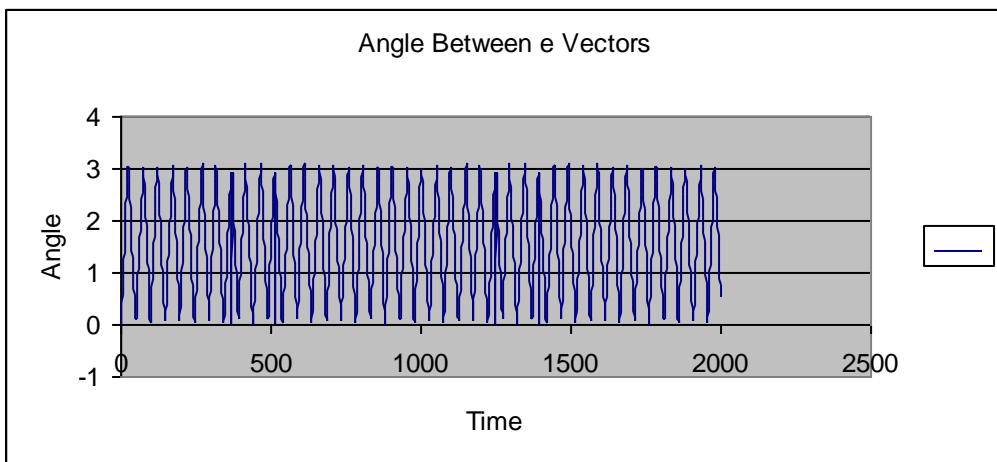
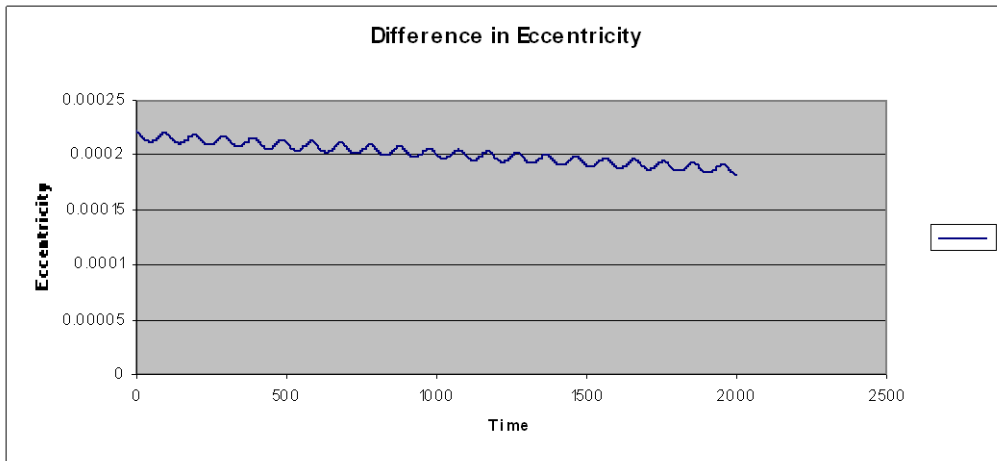




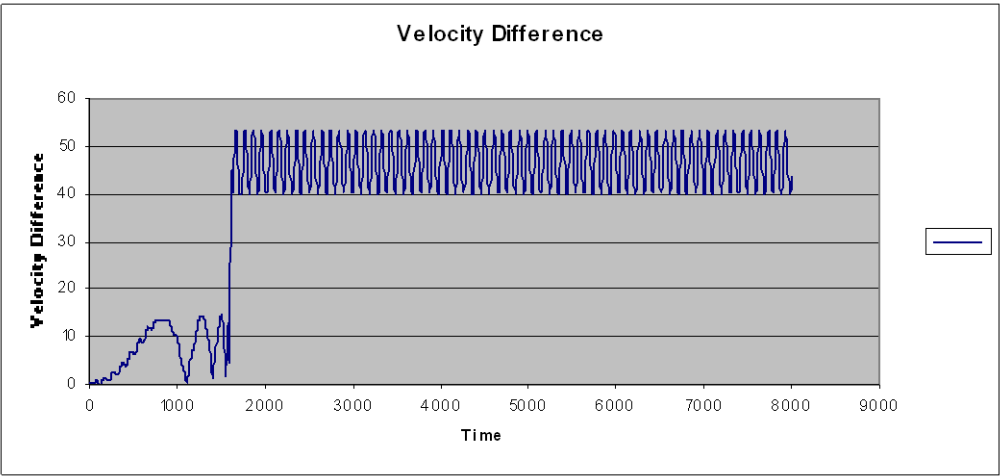
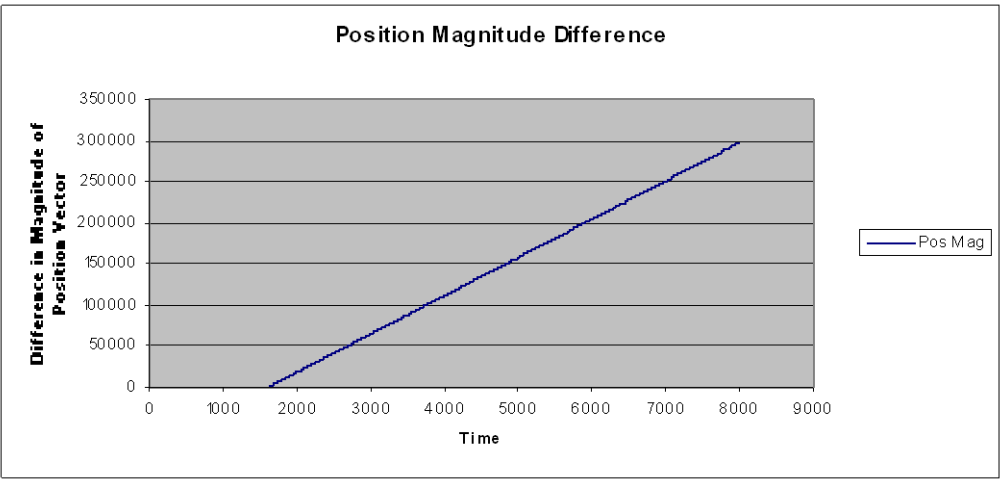
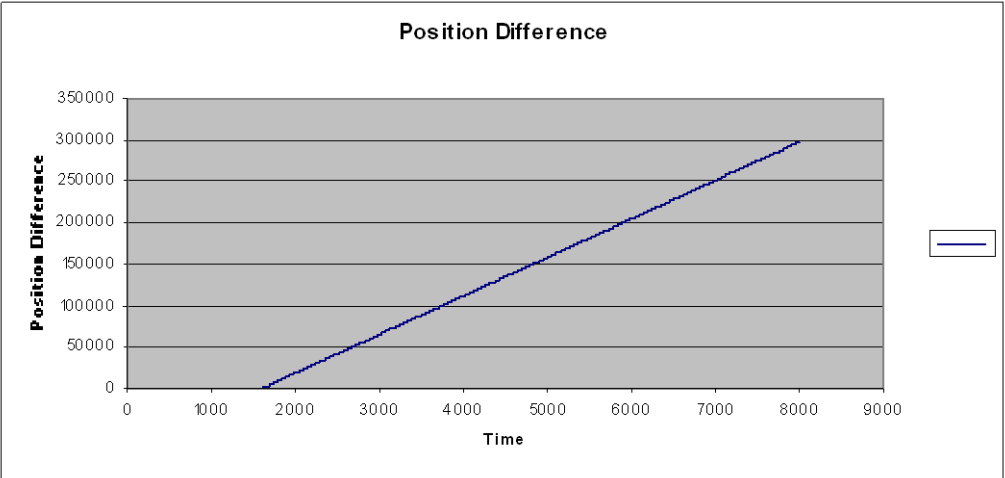
Time Step: 2

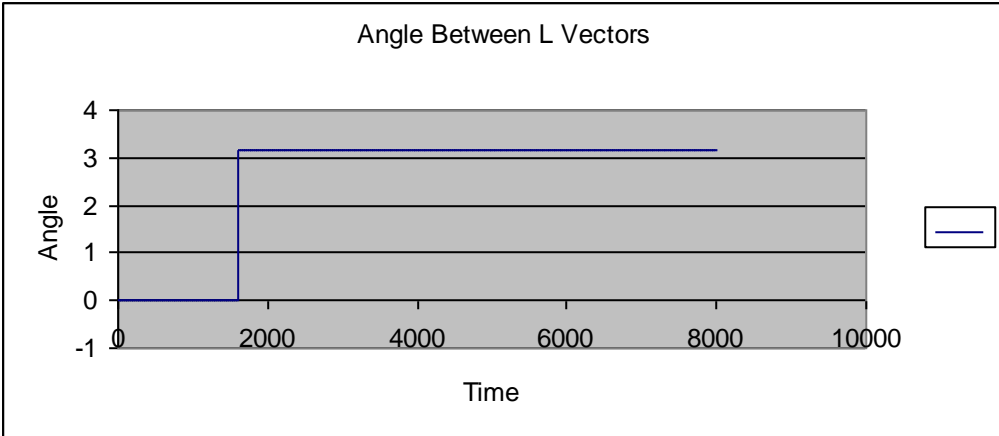
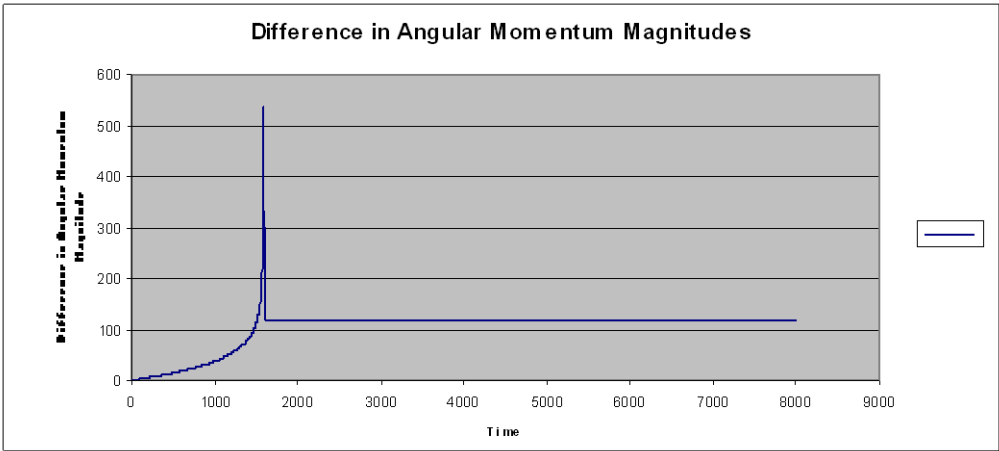
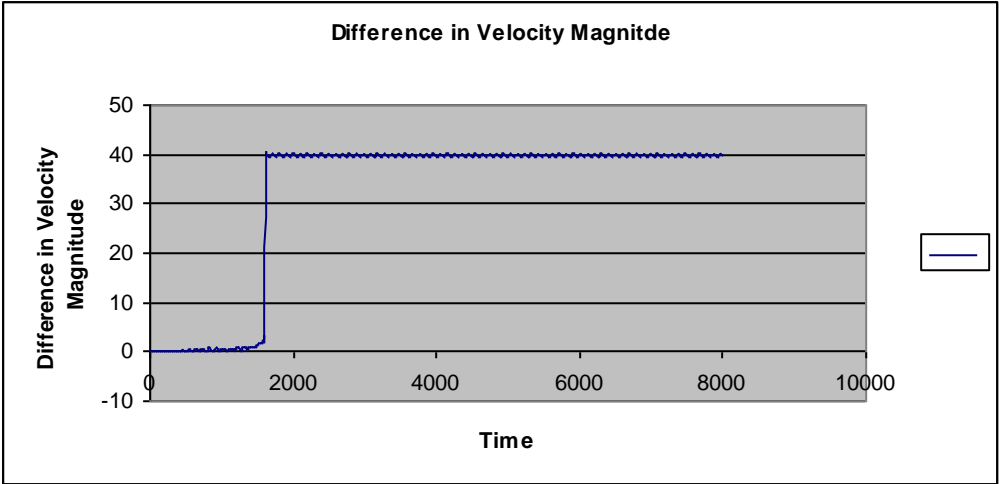


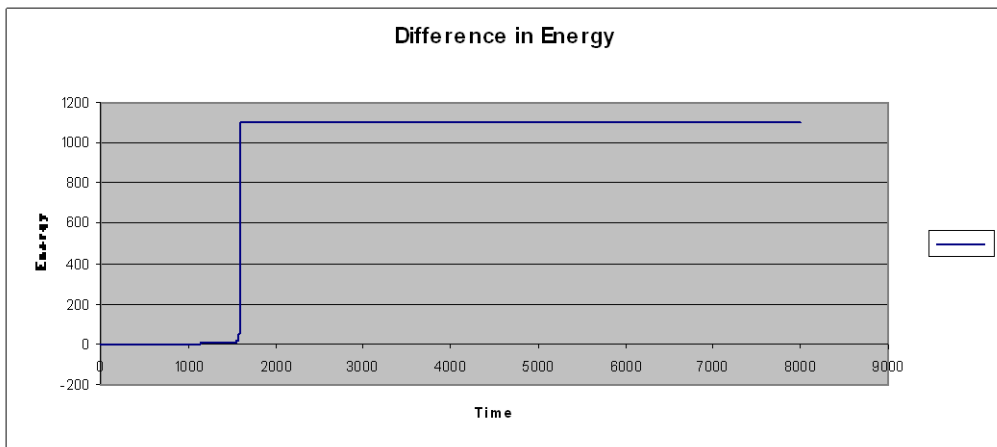
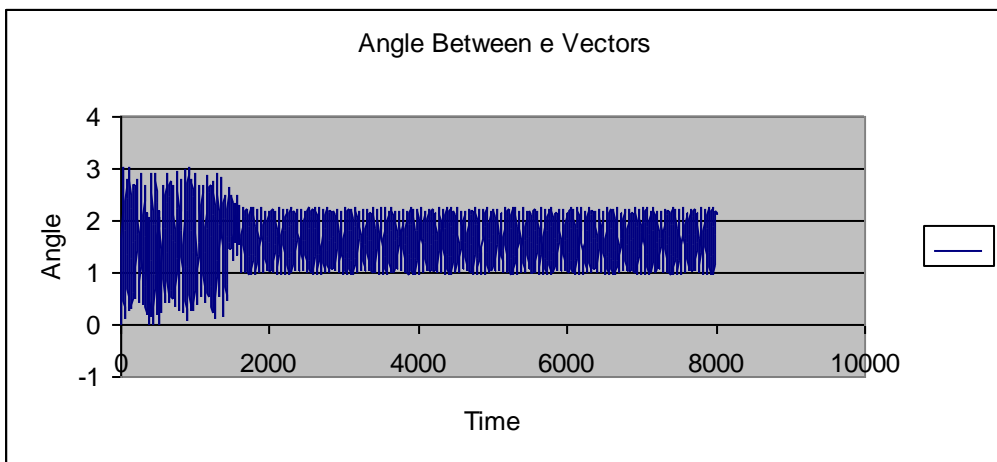
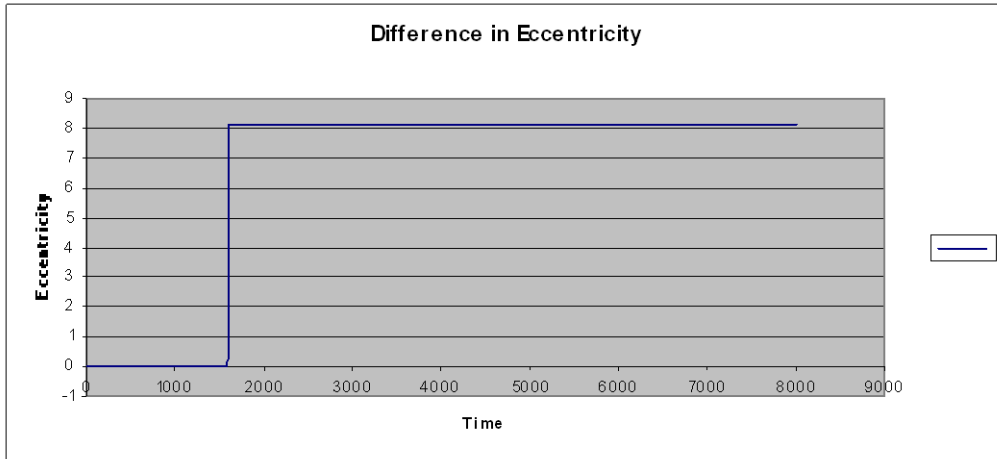




Time Step: 8

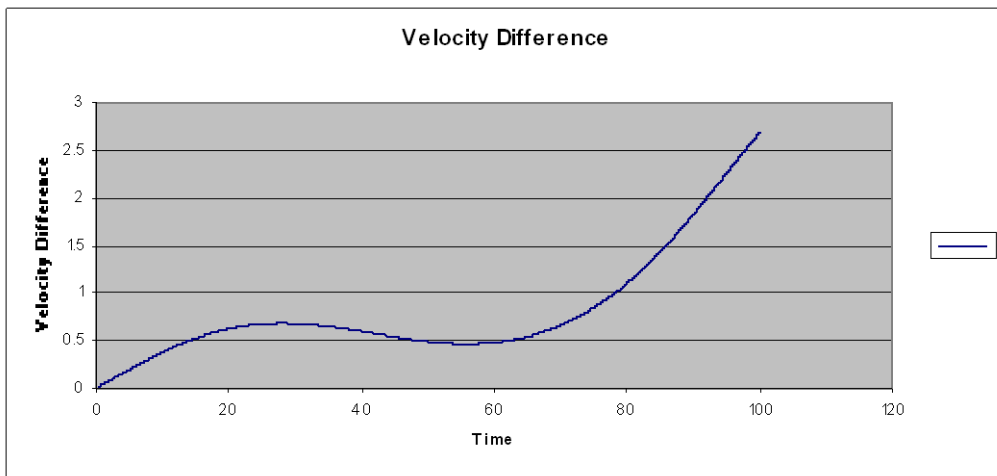
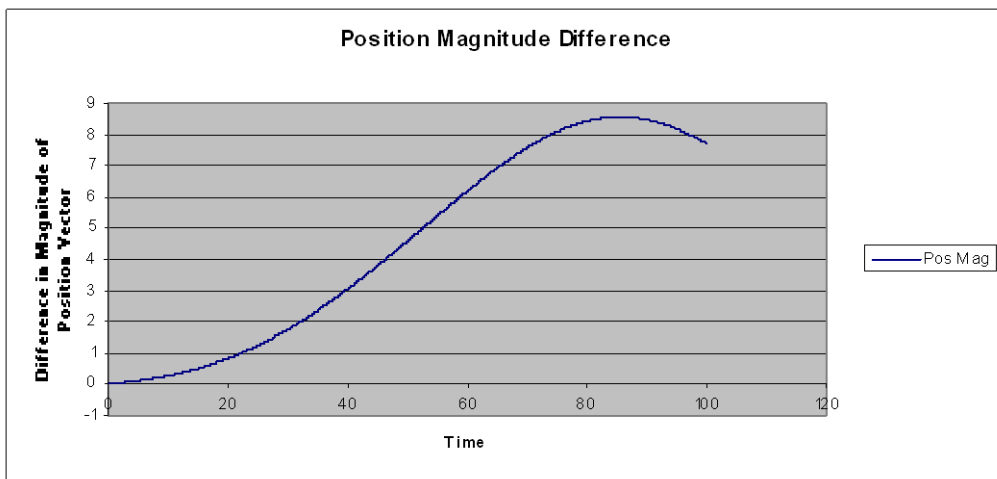
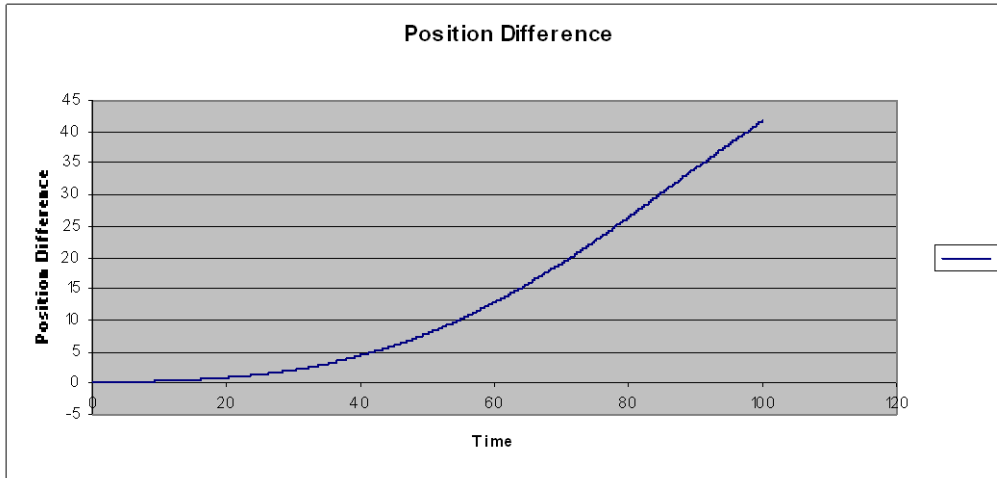


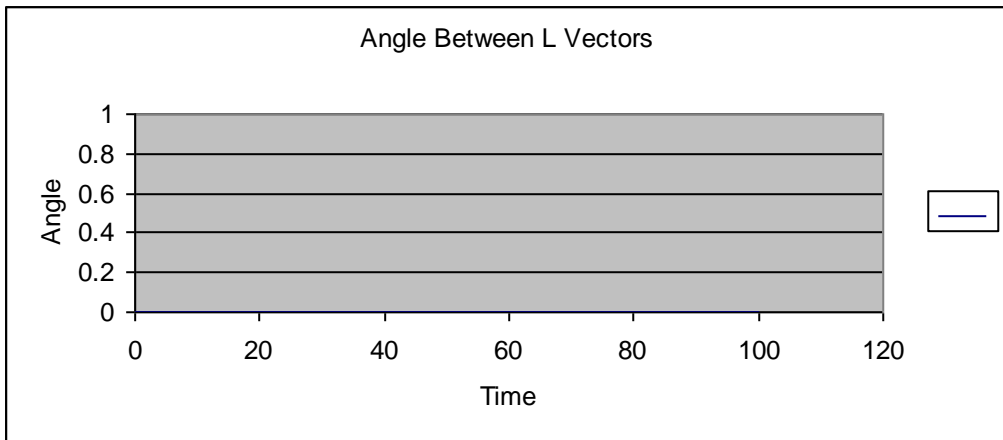
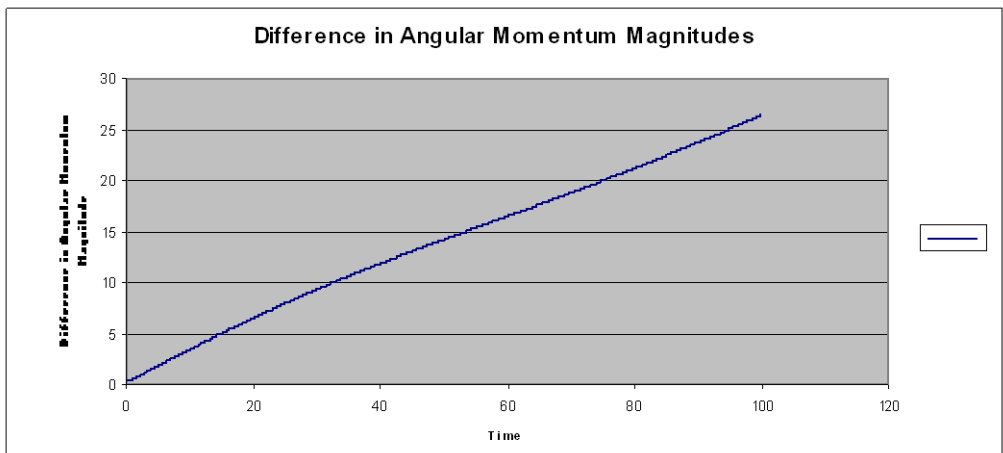
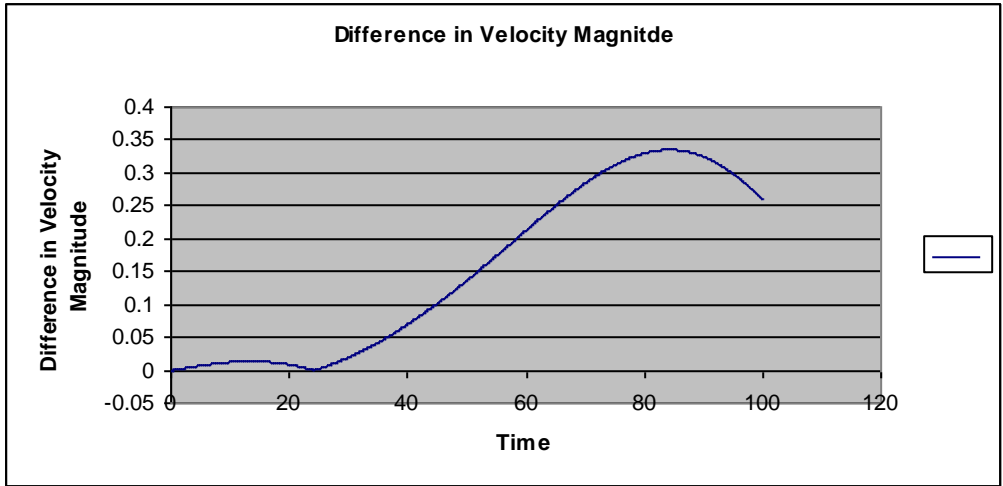


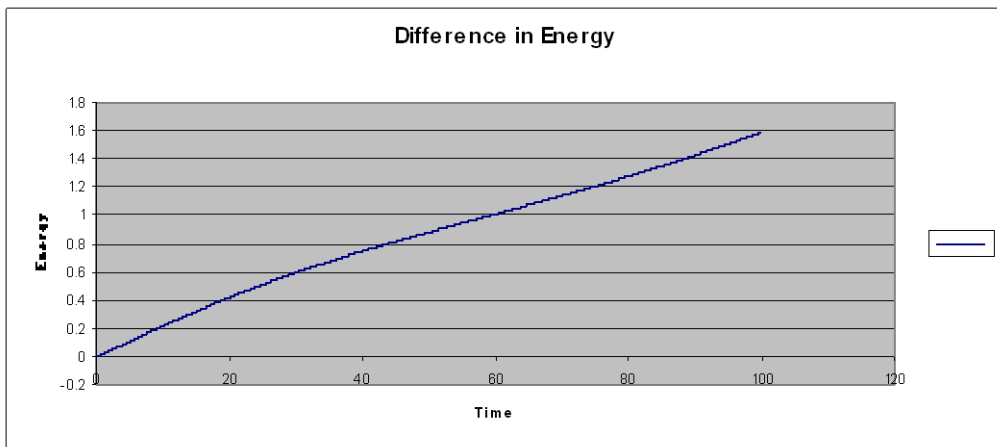
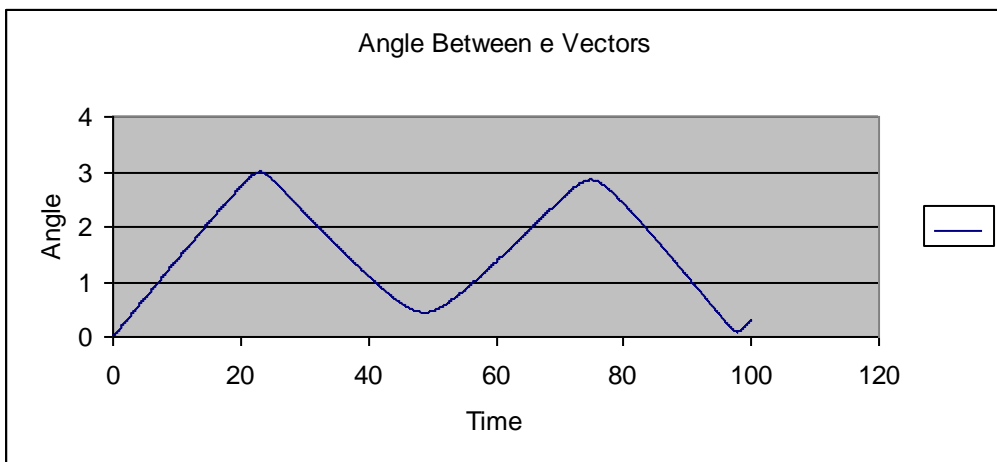
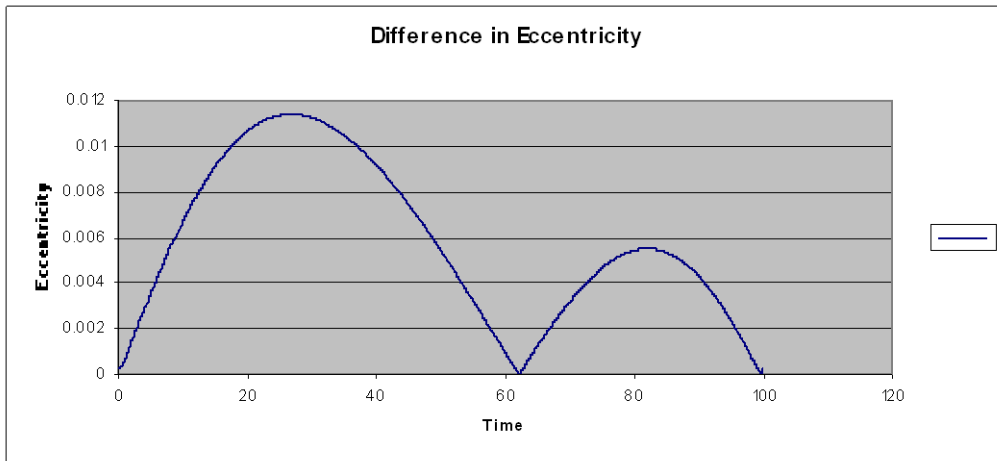


Adams-Bashford

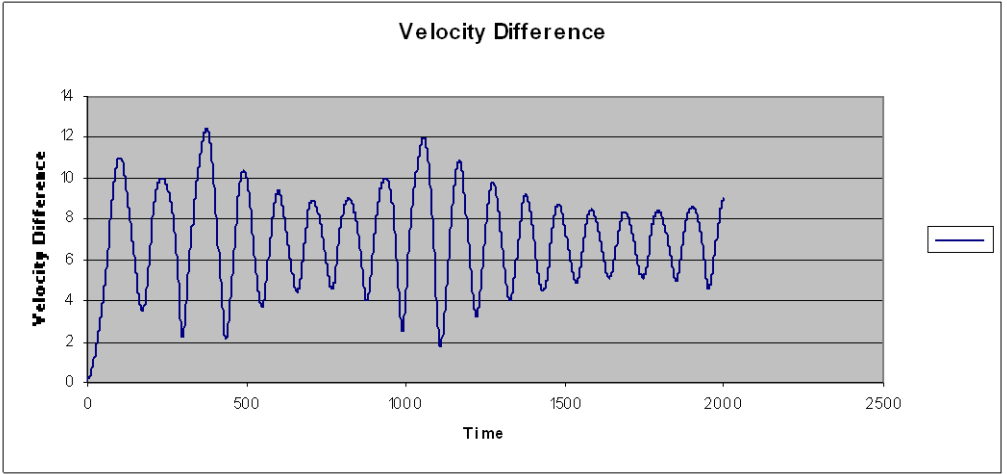
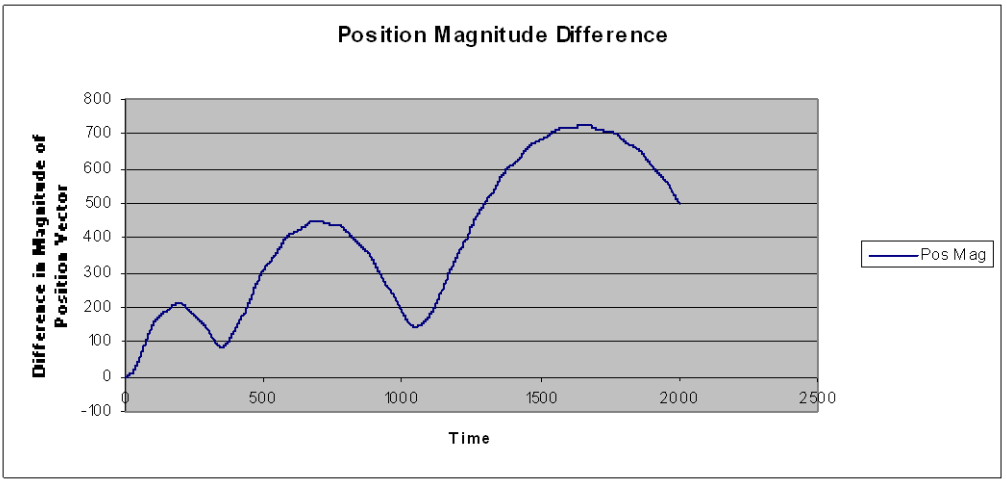
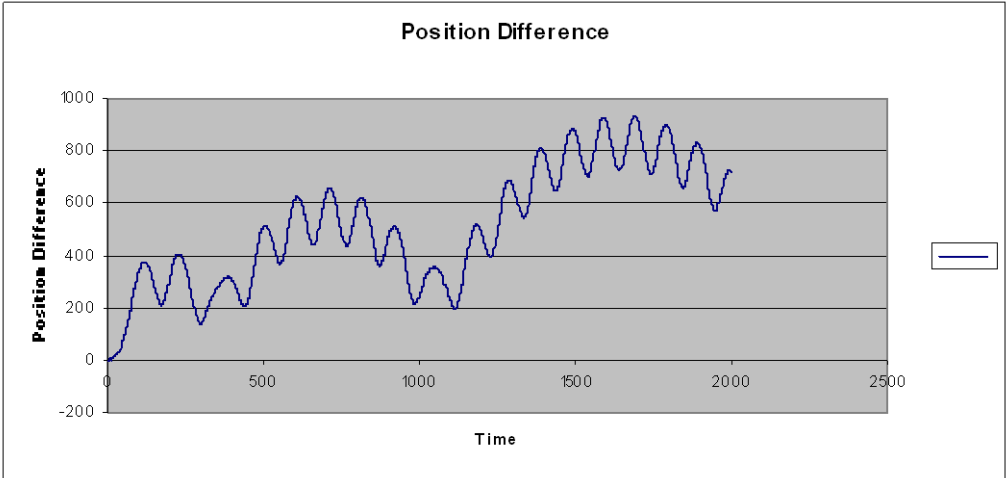
Time Step: .1

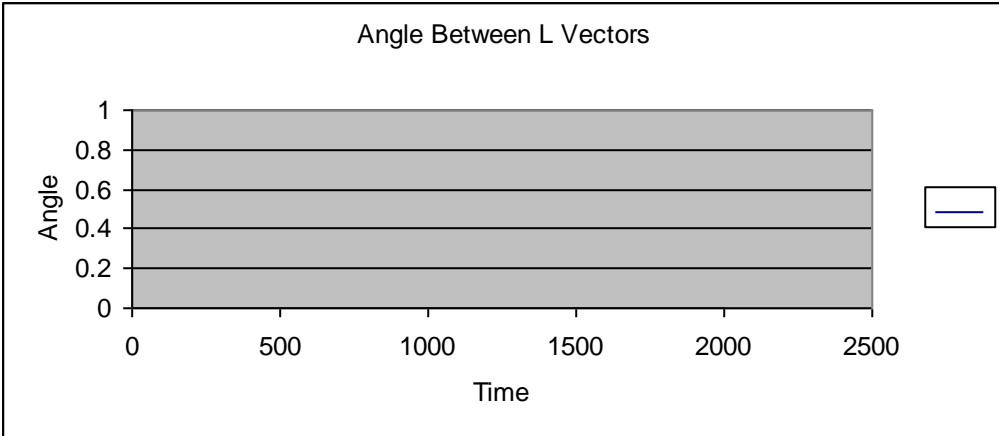
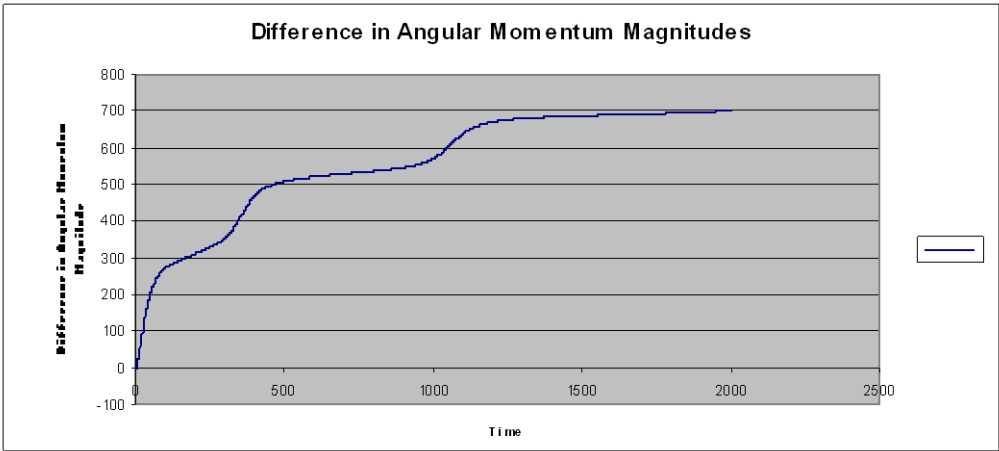
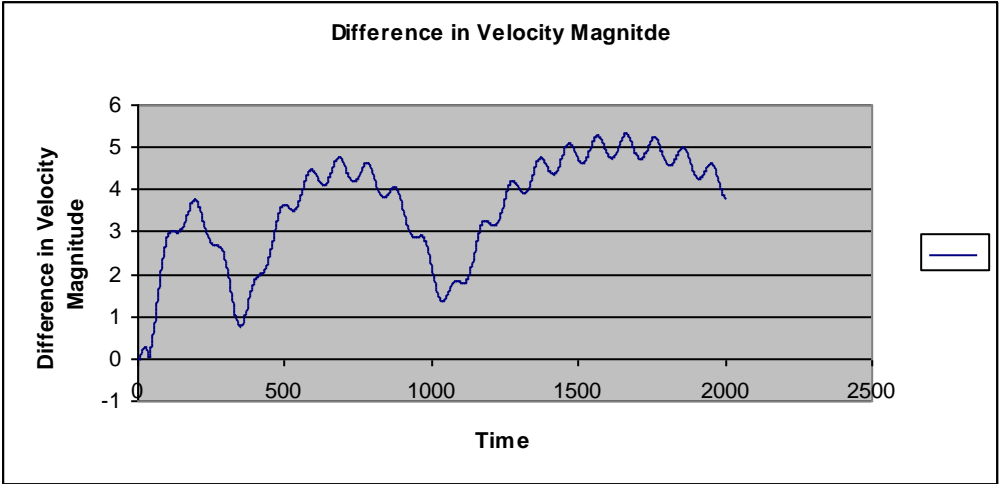


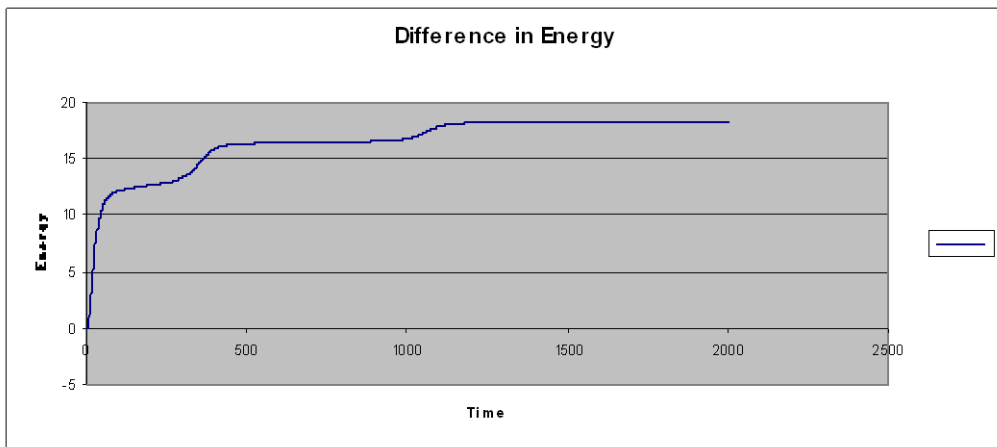
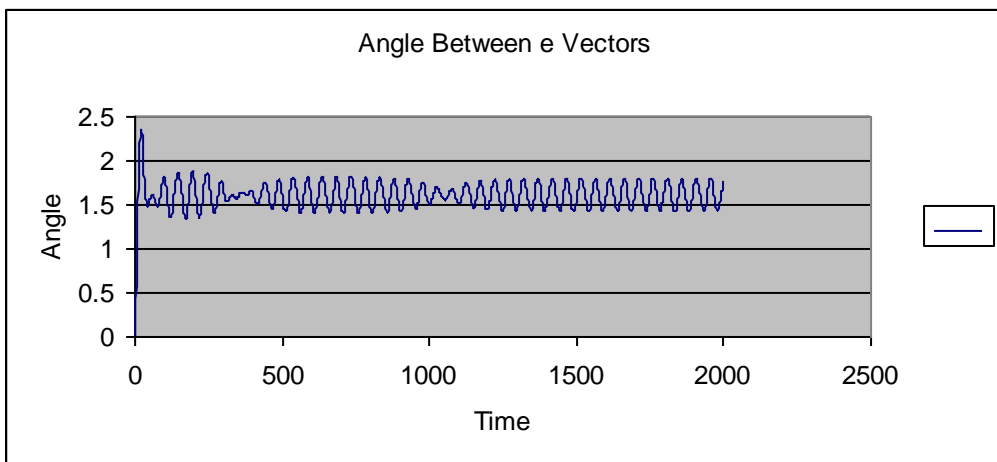
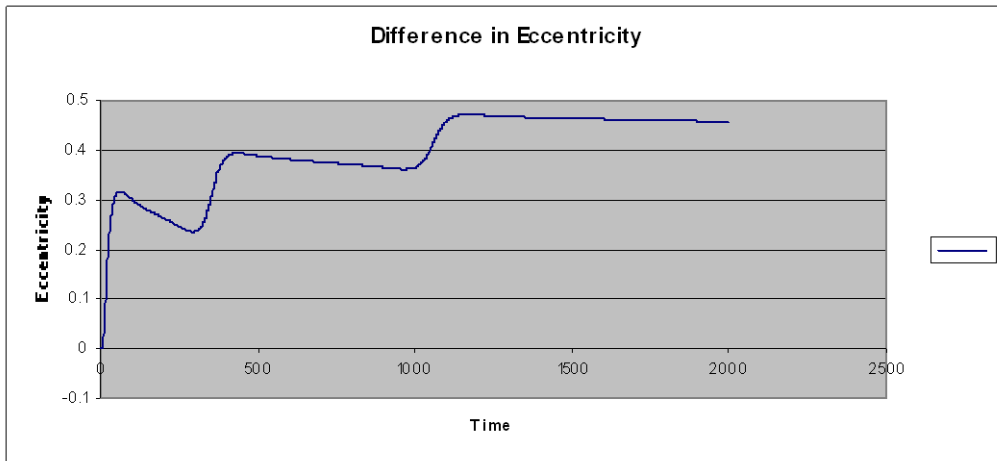




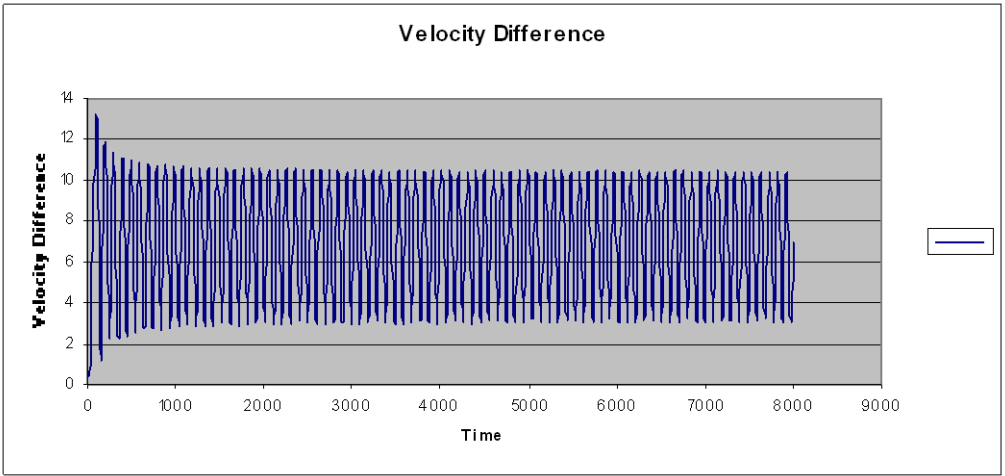
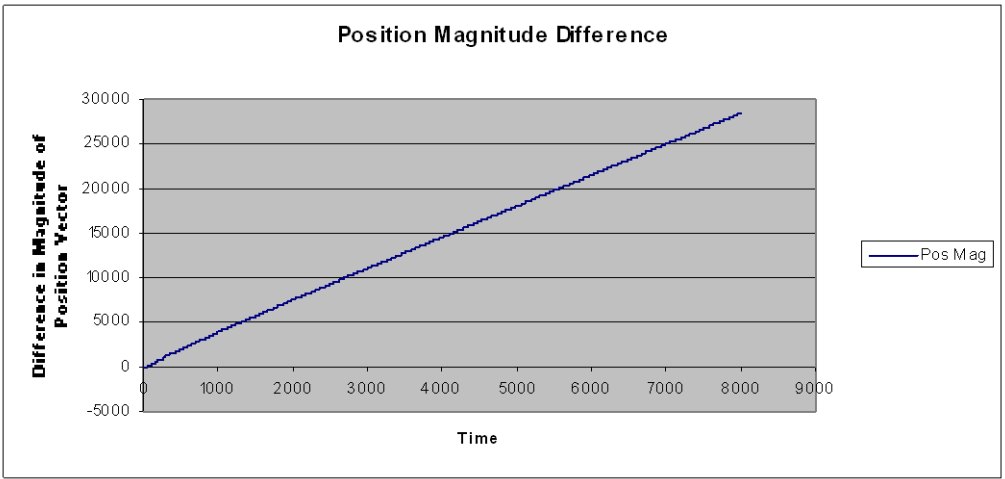
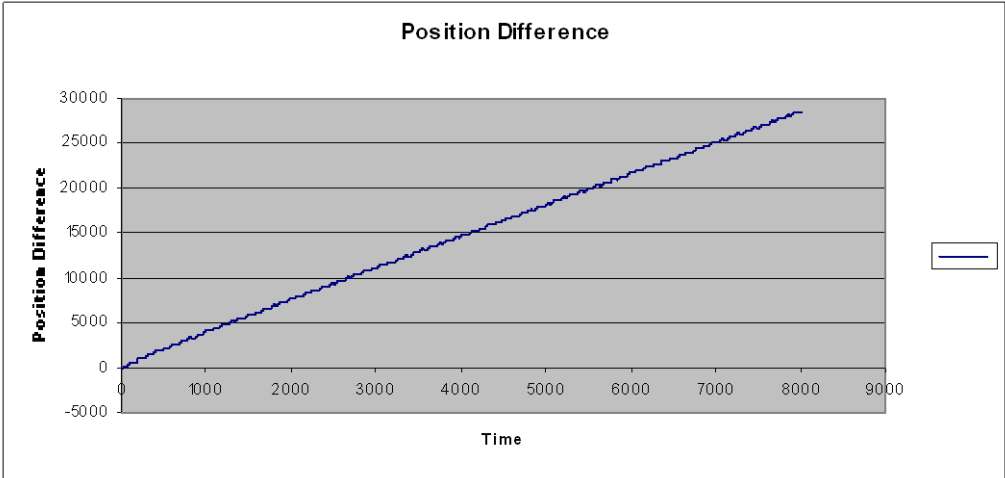
Time Step: 2

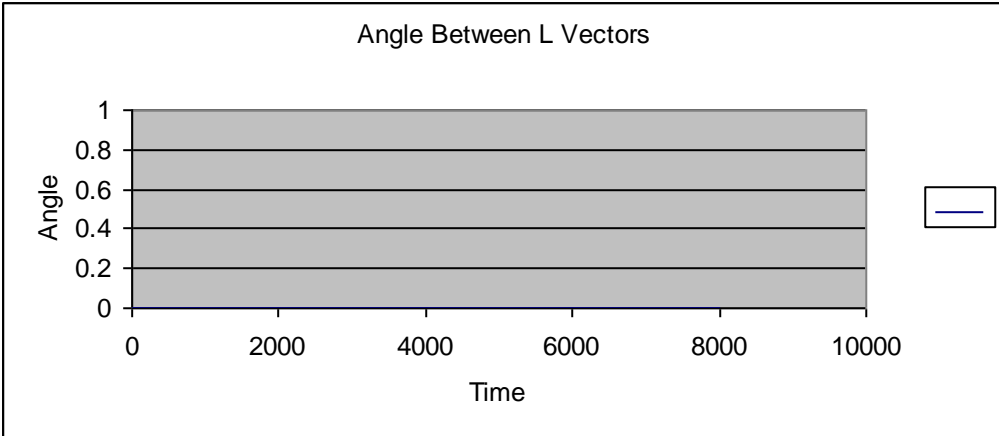
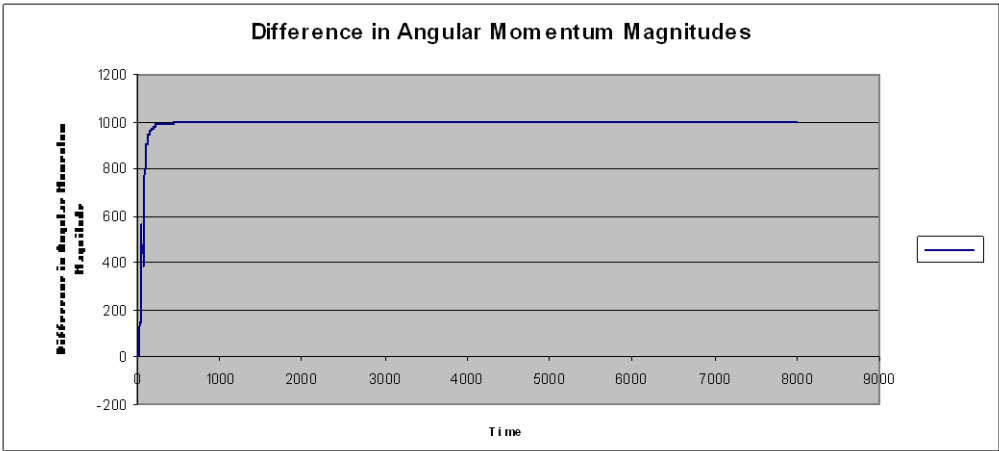
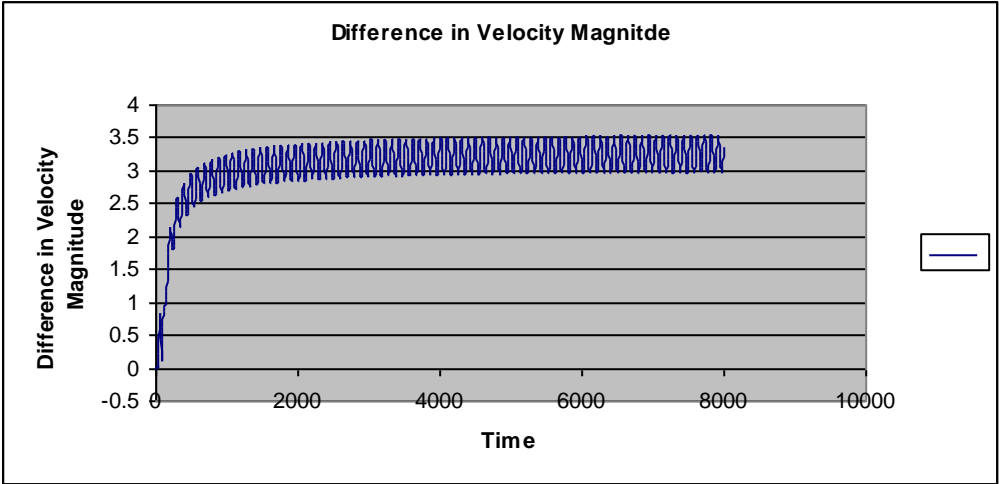


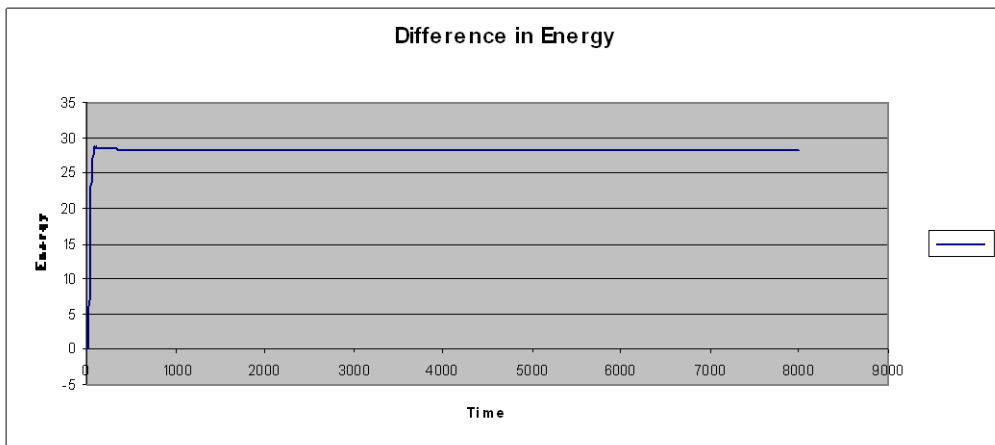
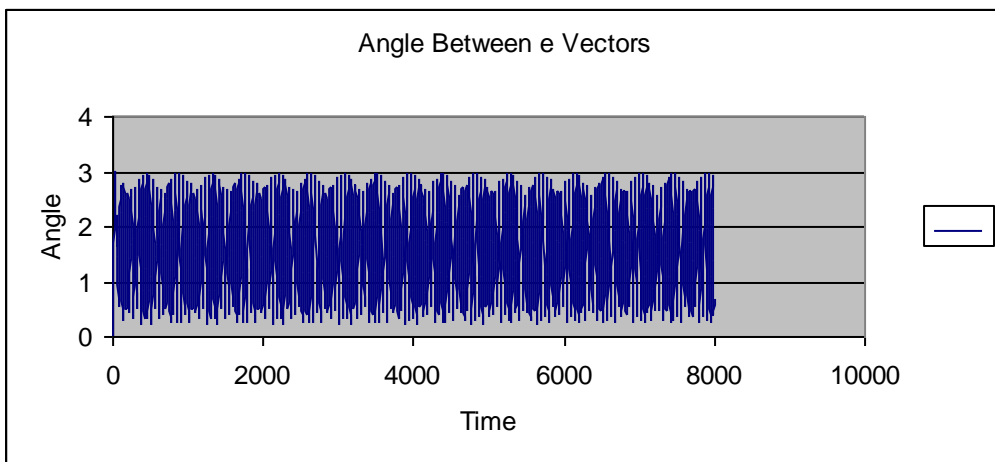
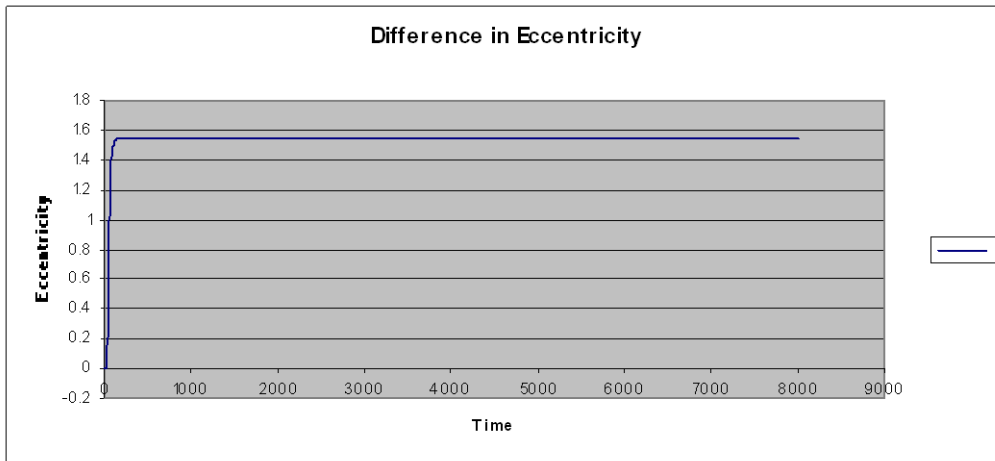




Time Step: 8

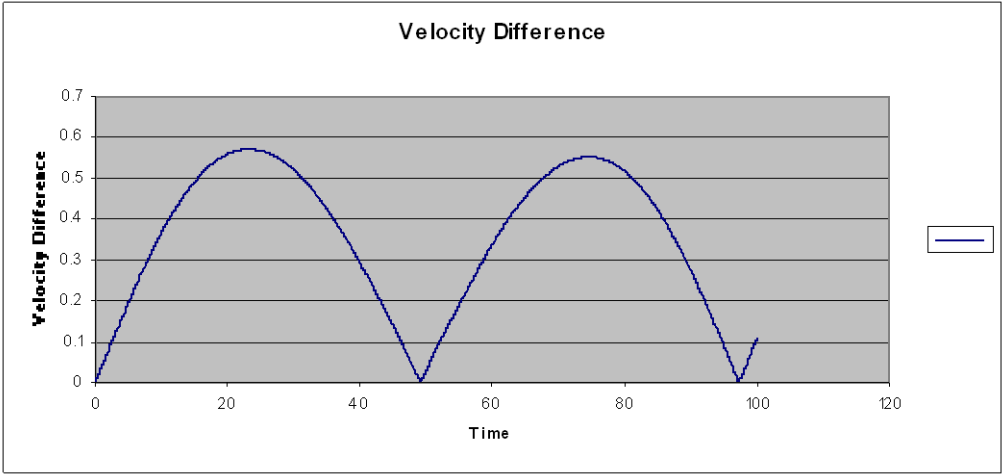
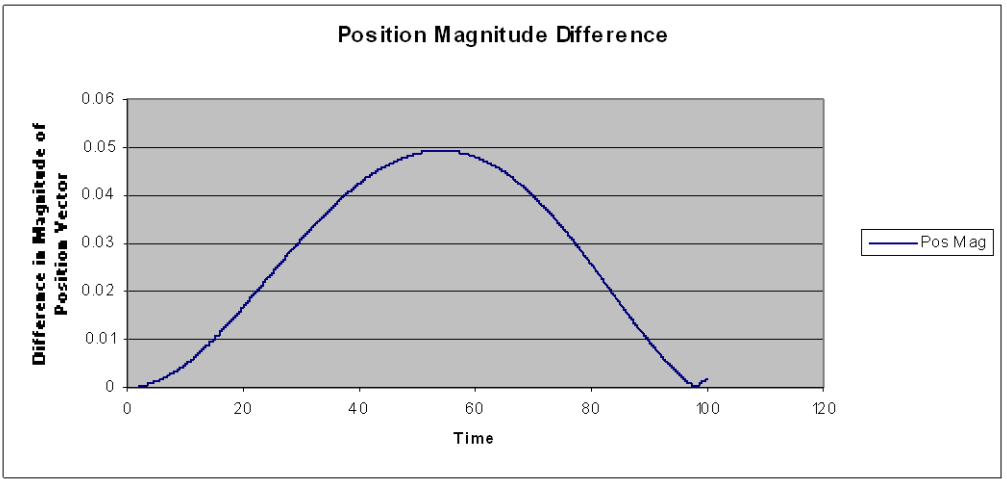
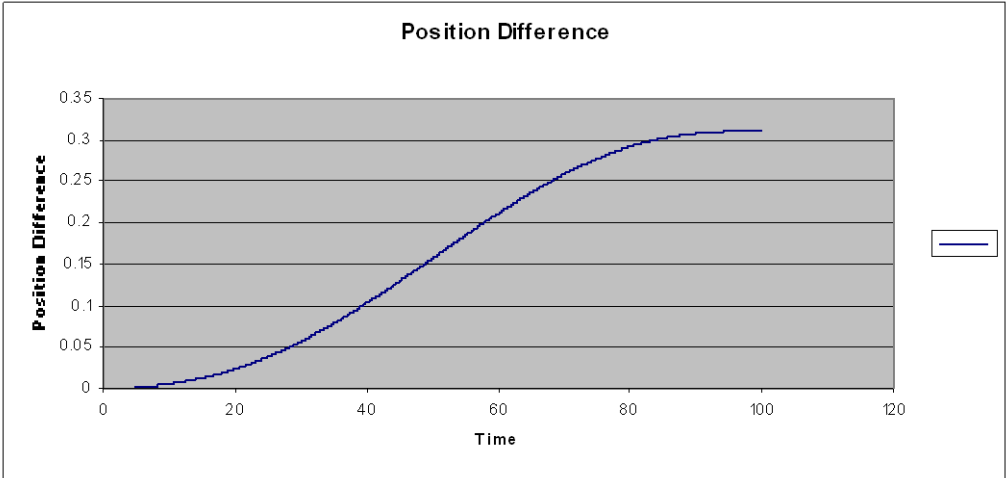


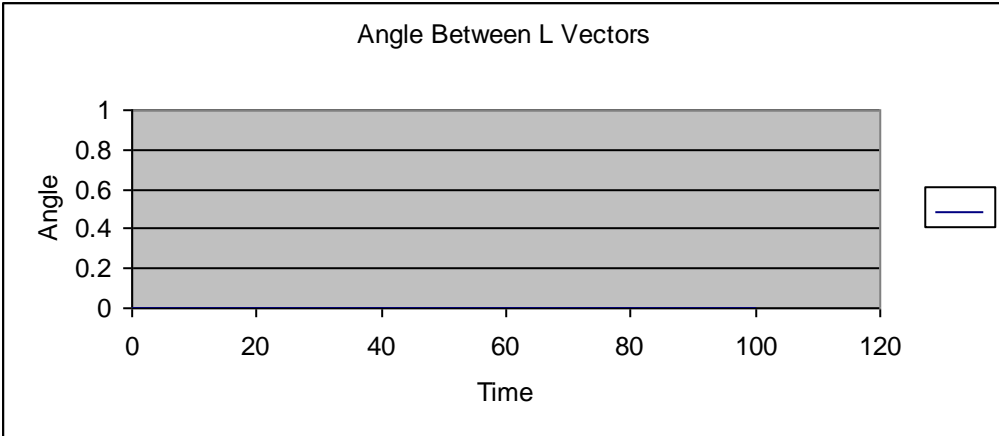
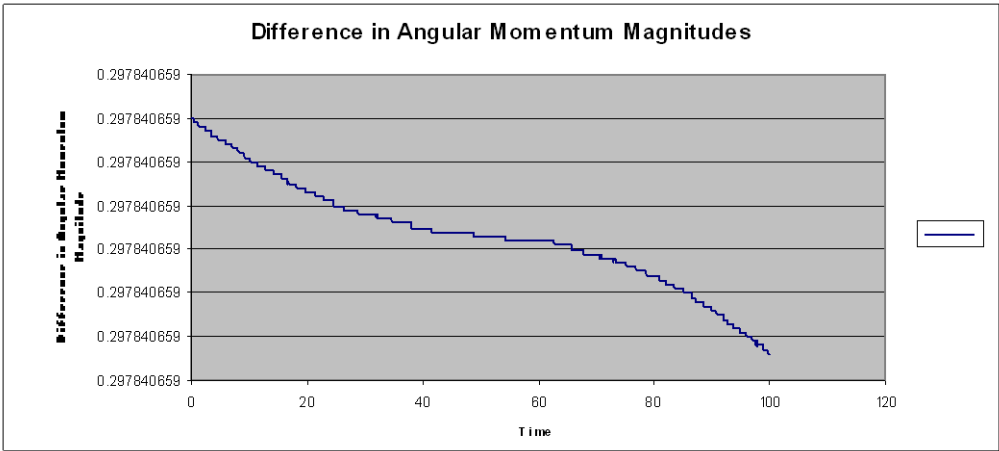
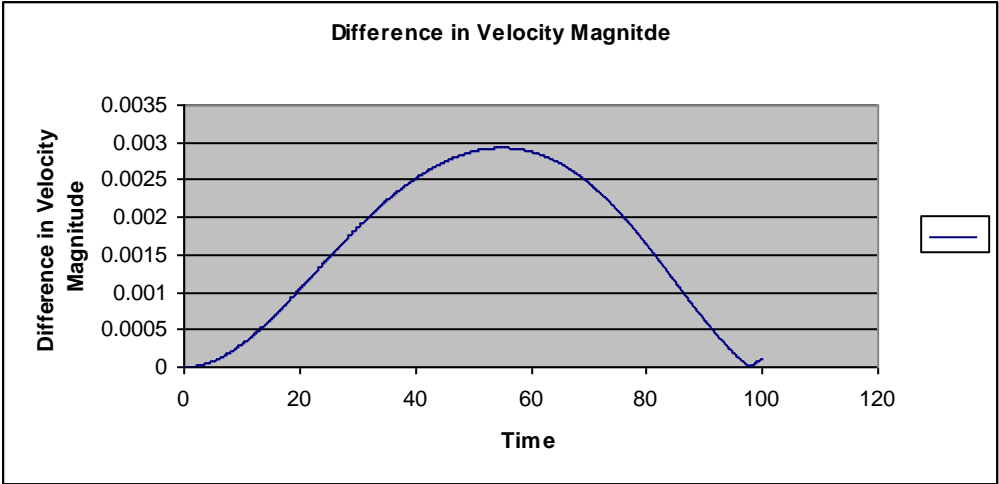


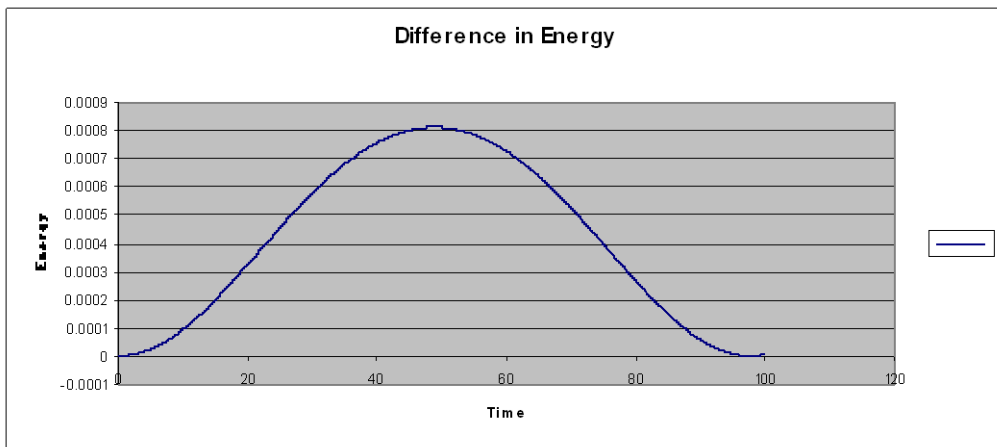
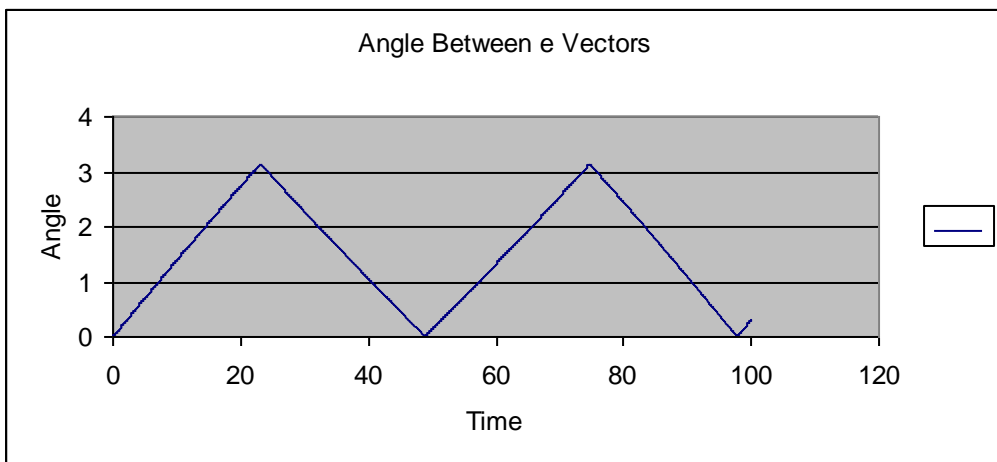
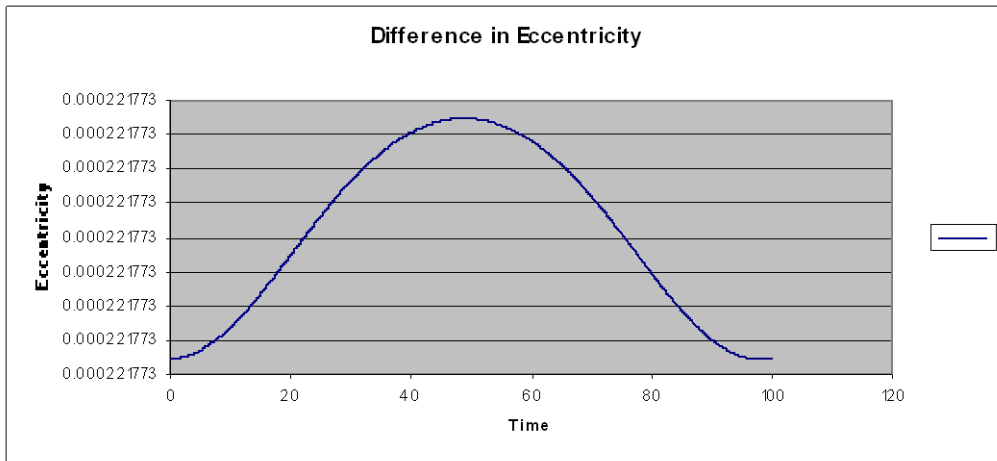


Gill's Method

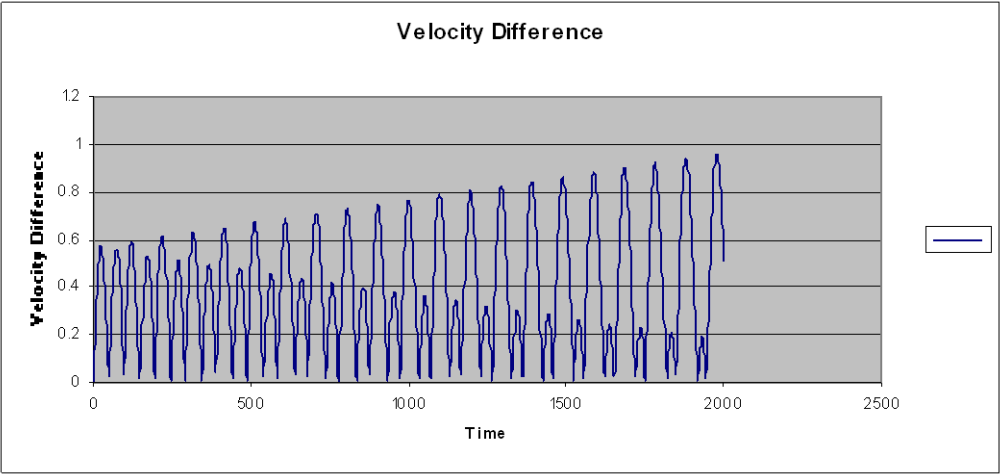
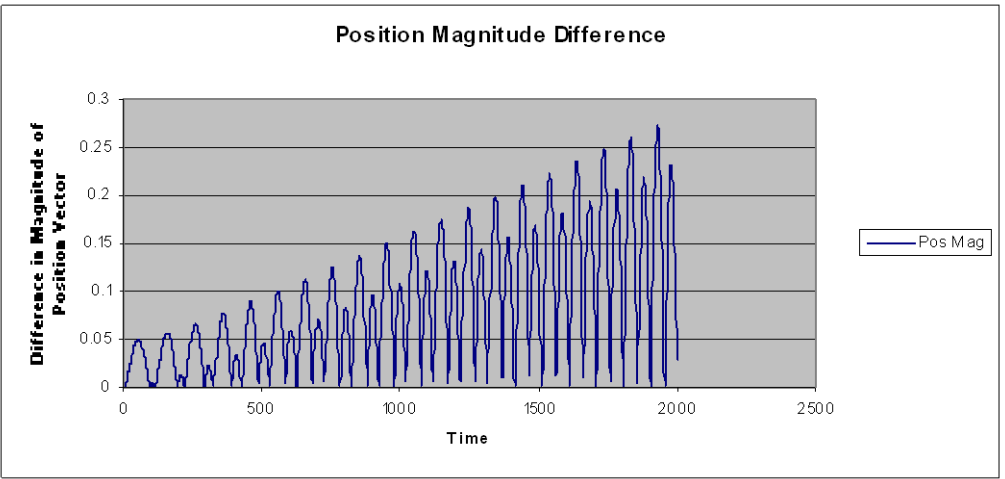
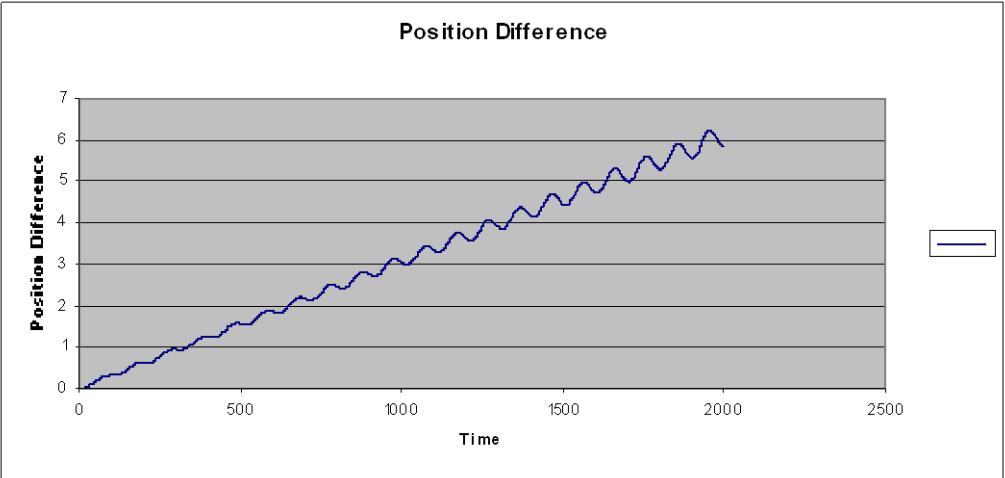
Time Step: .1

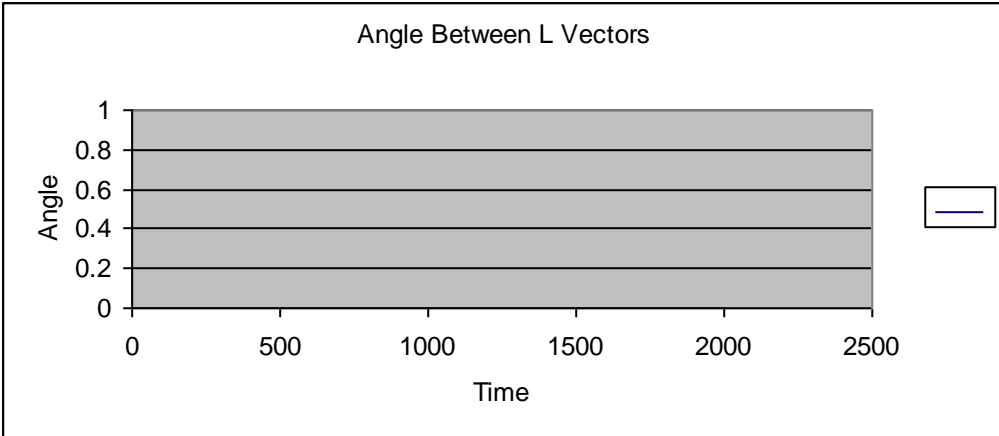
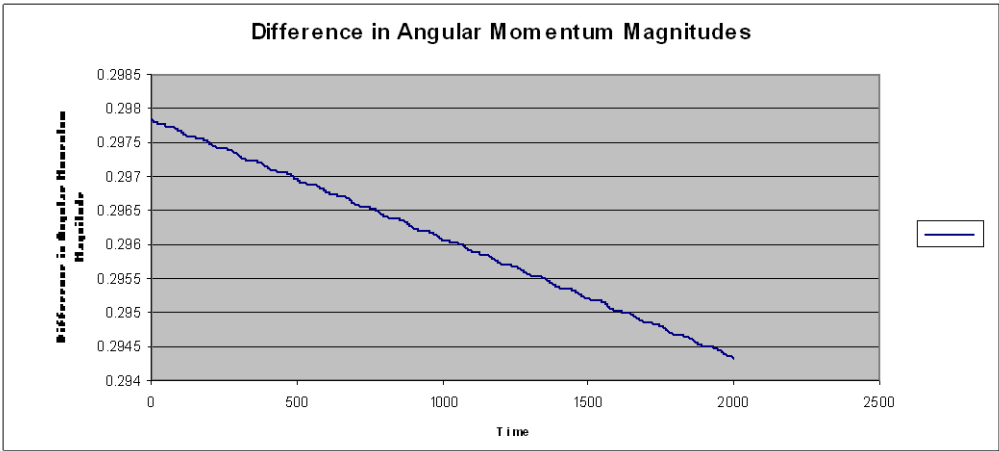
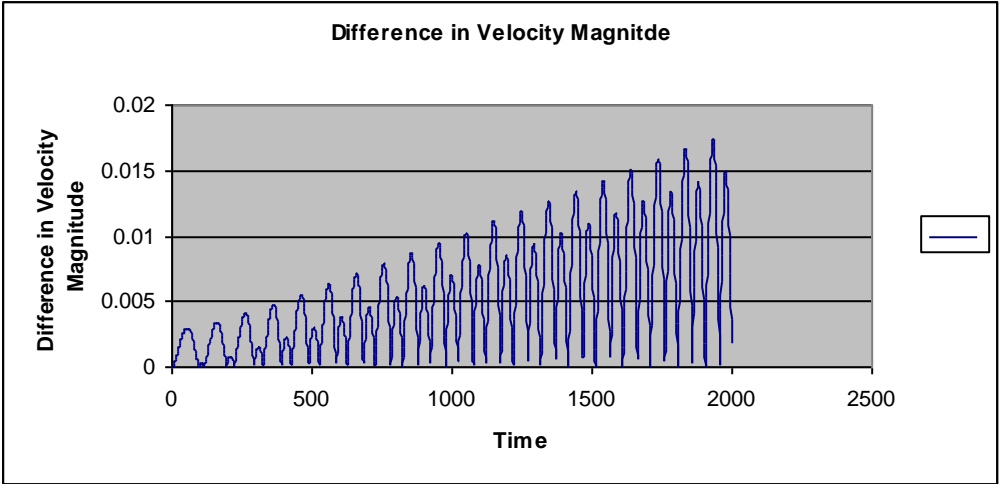


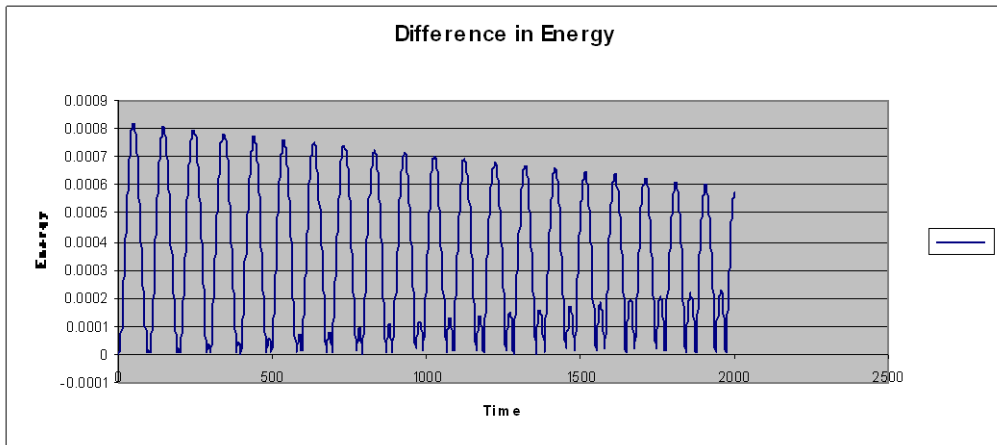
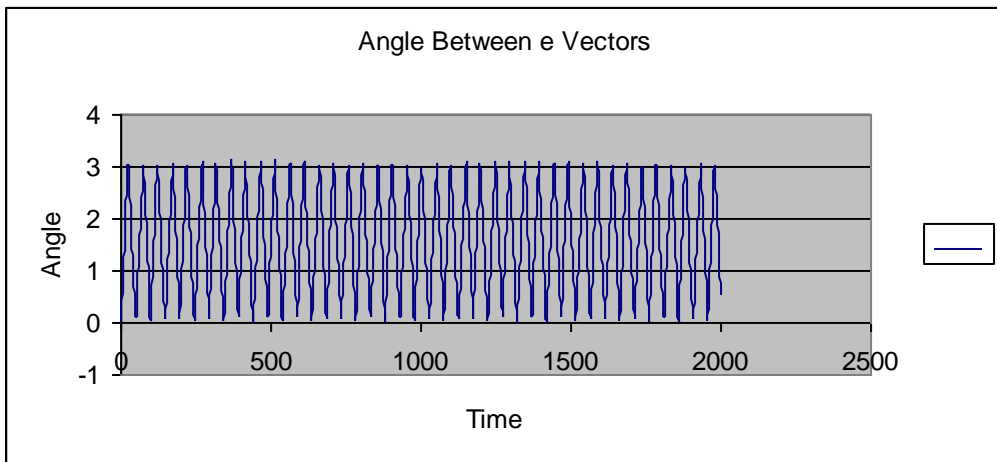
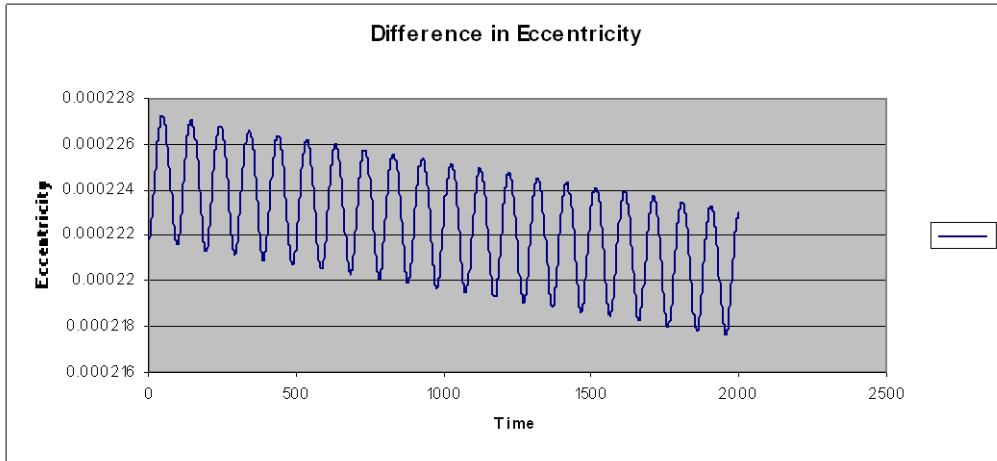




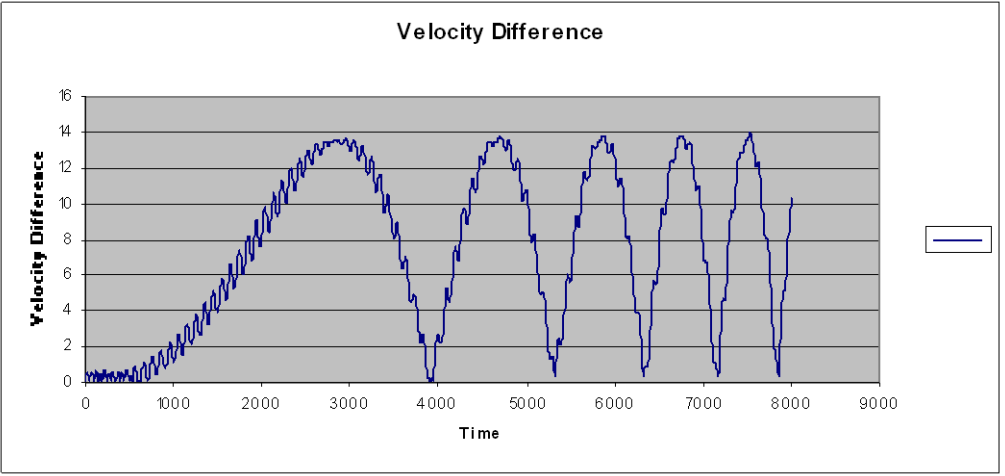
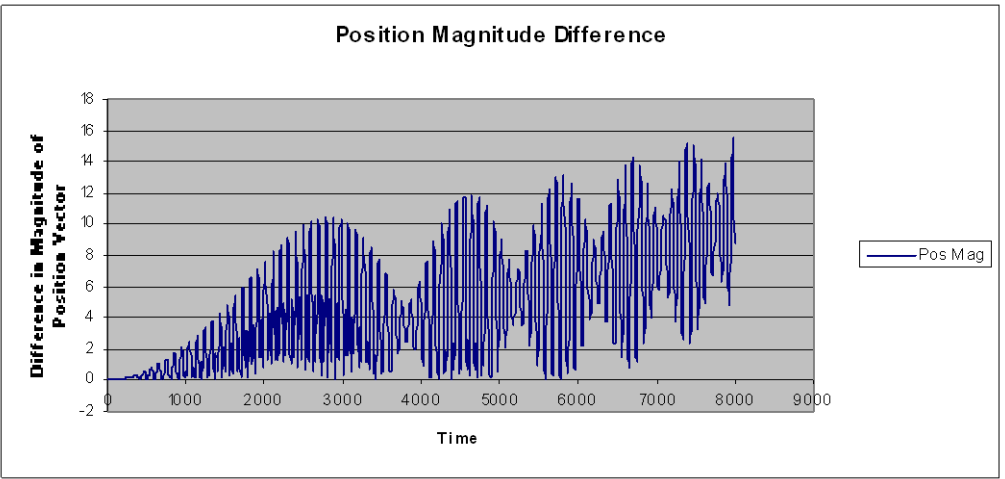
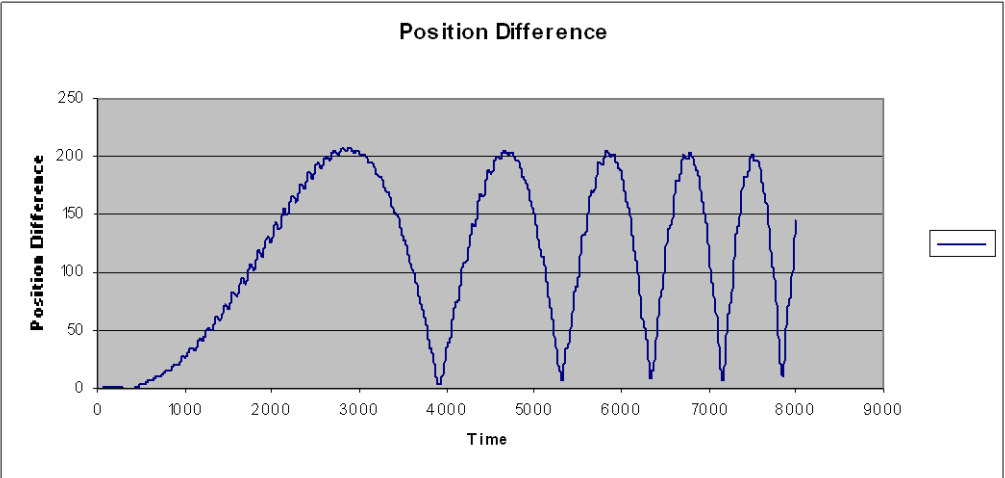
Time Step: 2

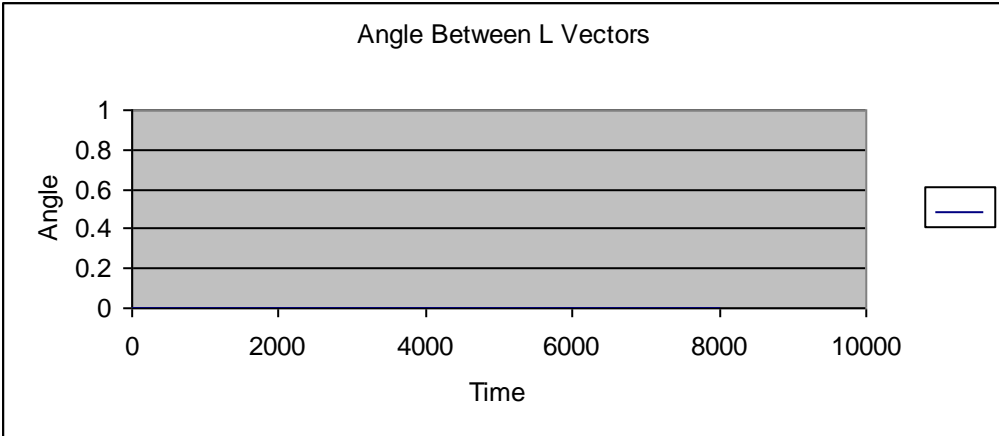
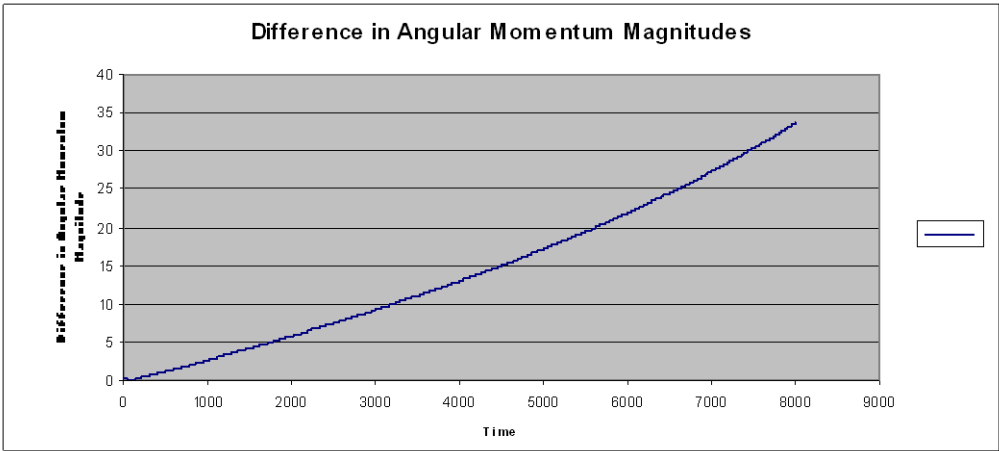
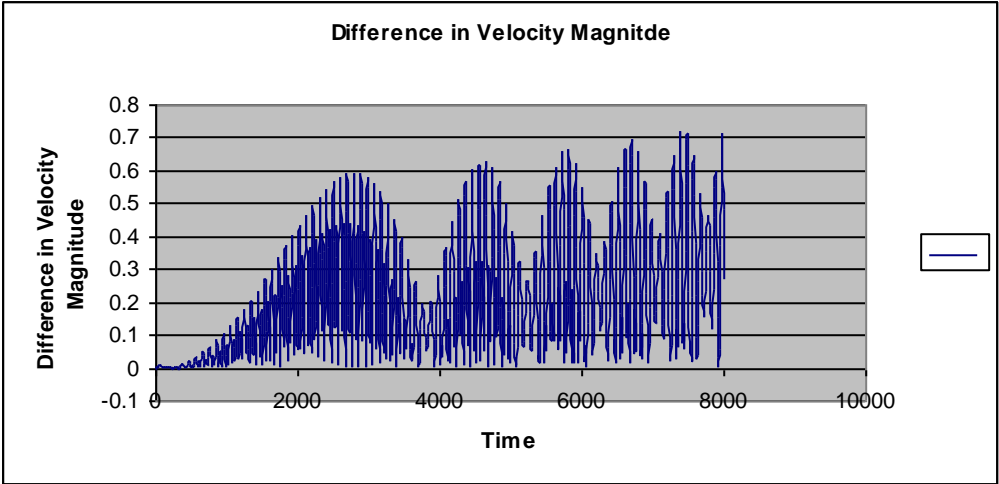


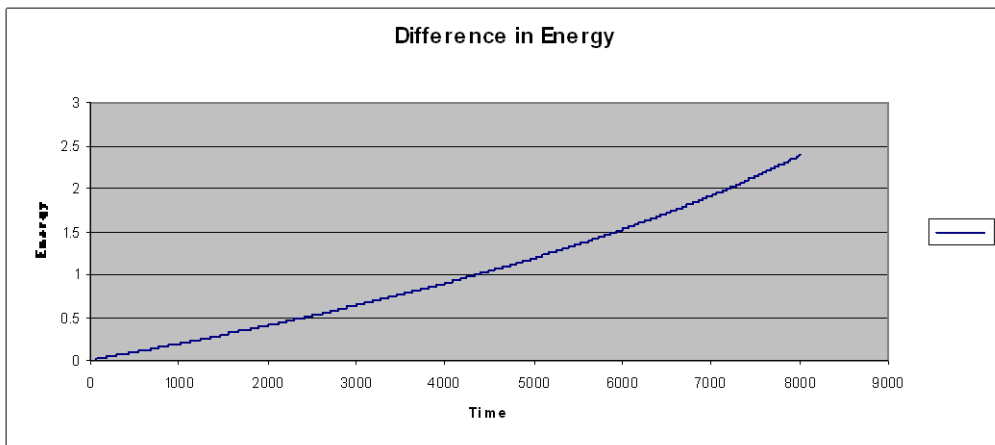
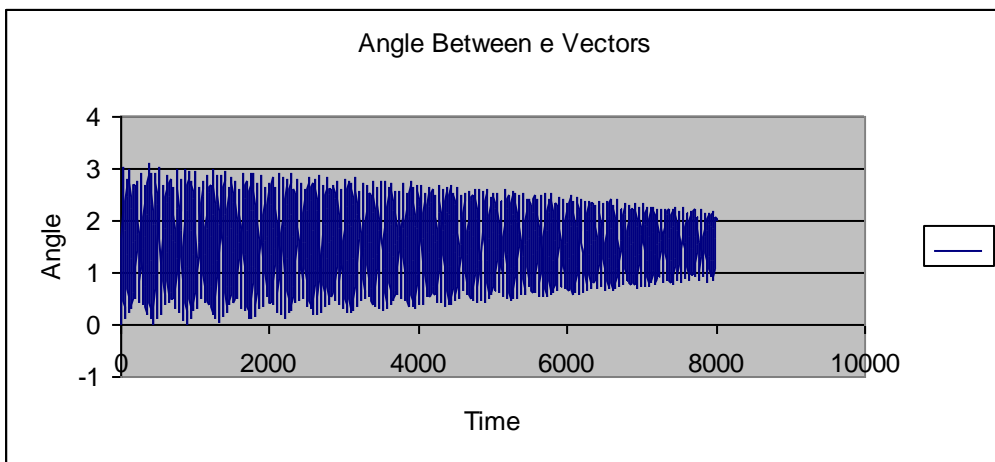
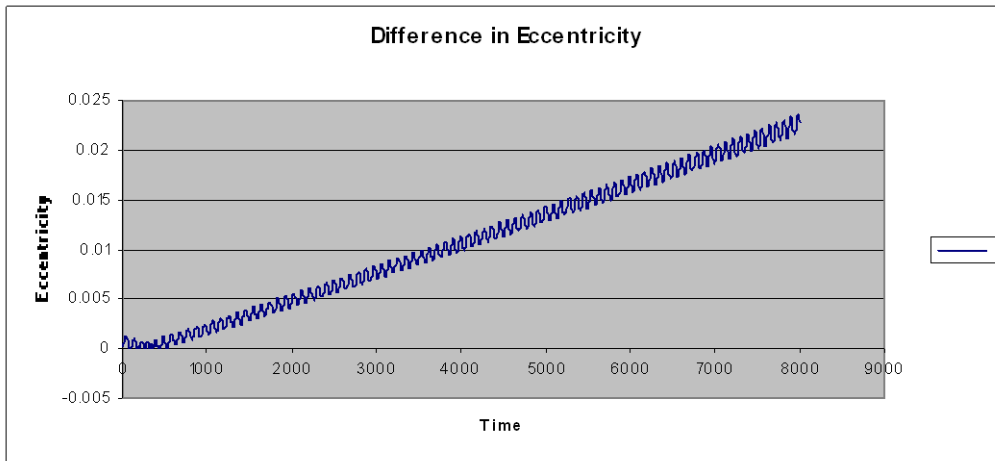




Time Step: 8

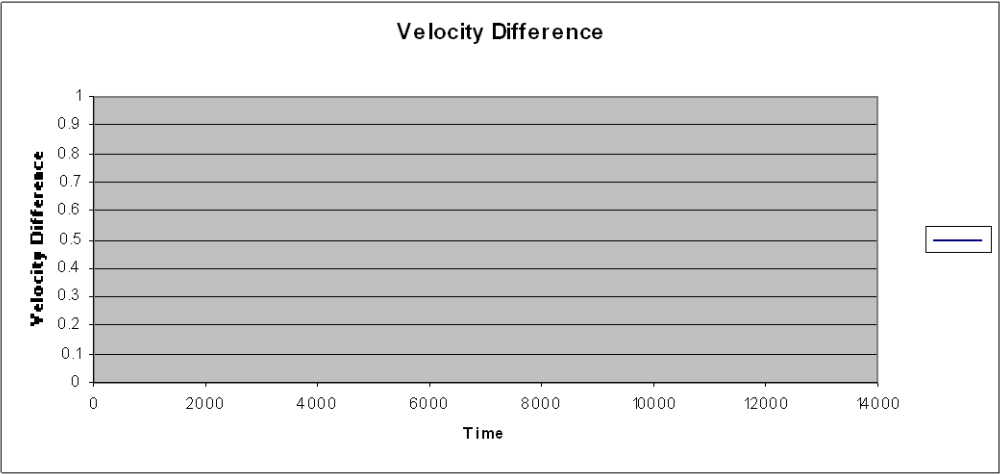
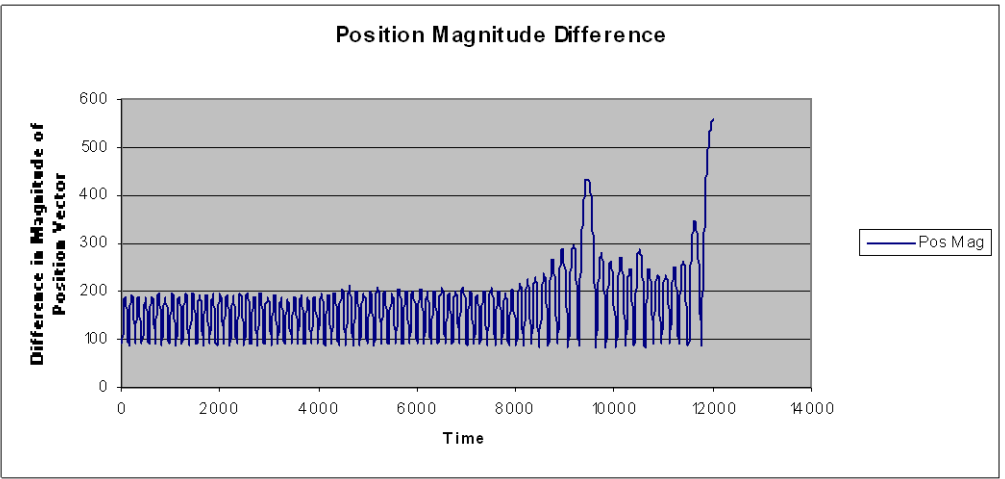
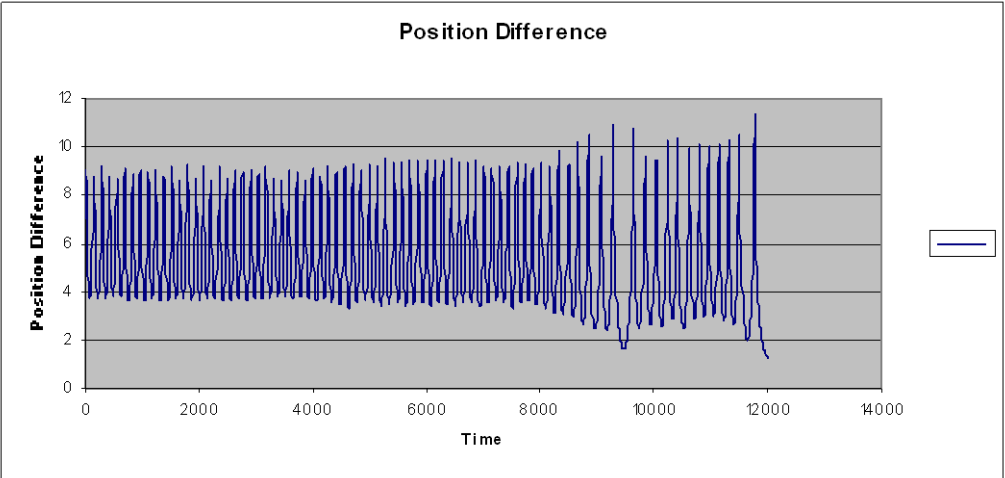


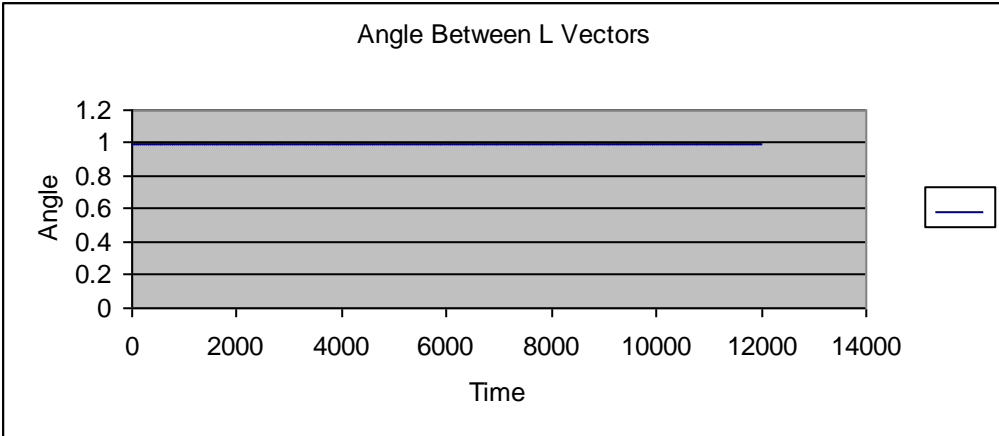
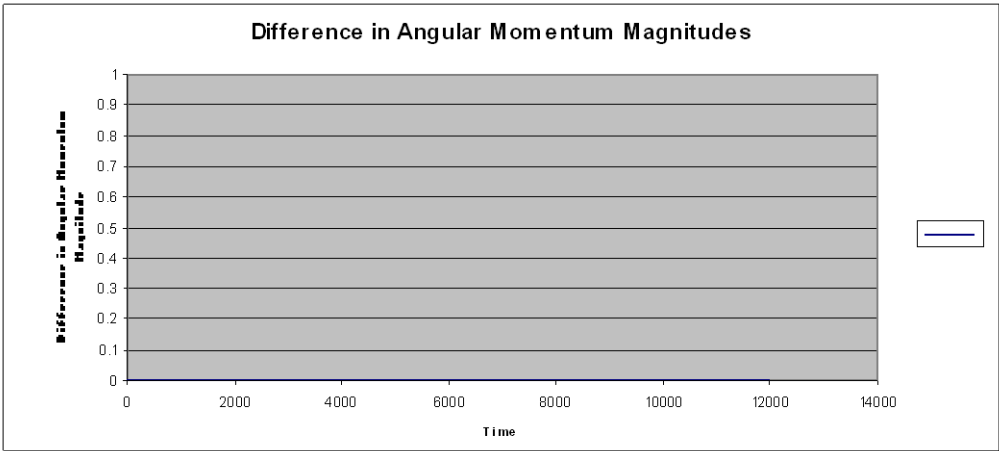
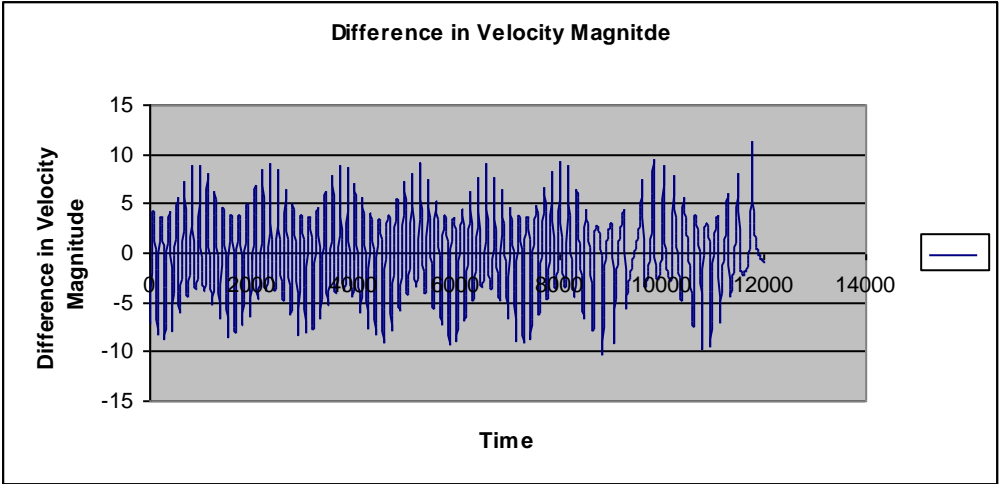


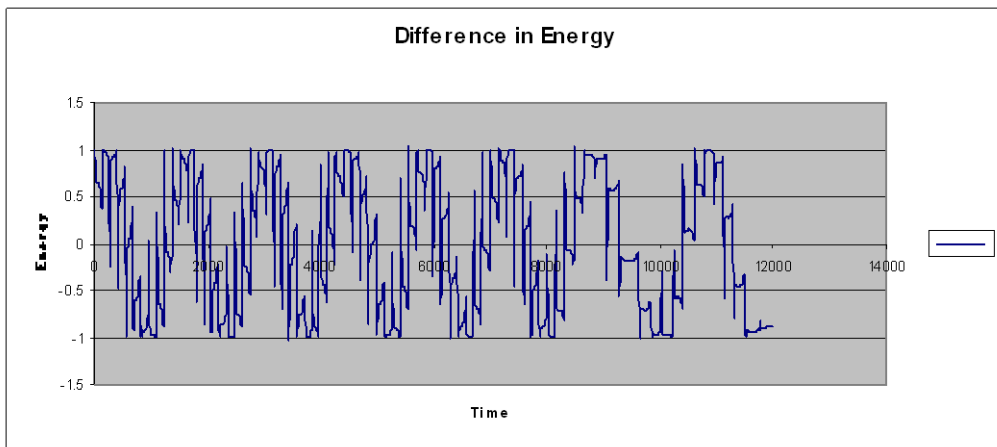
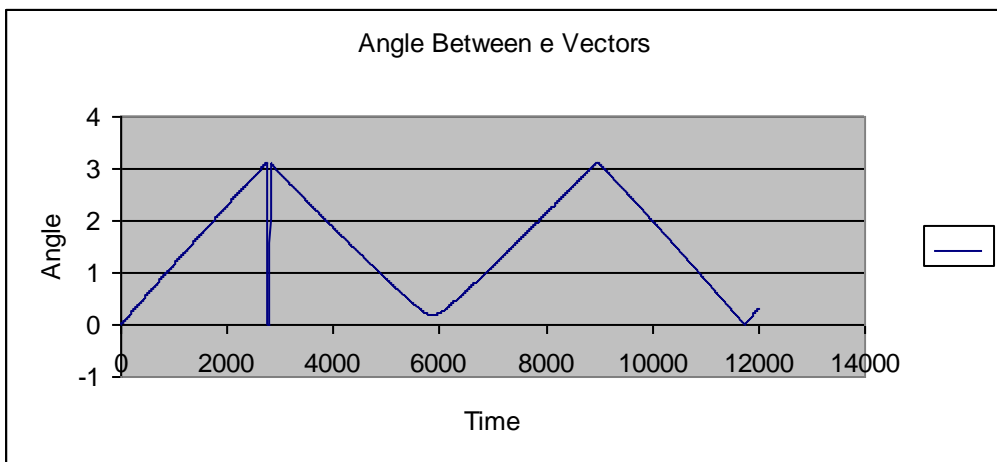
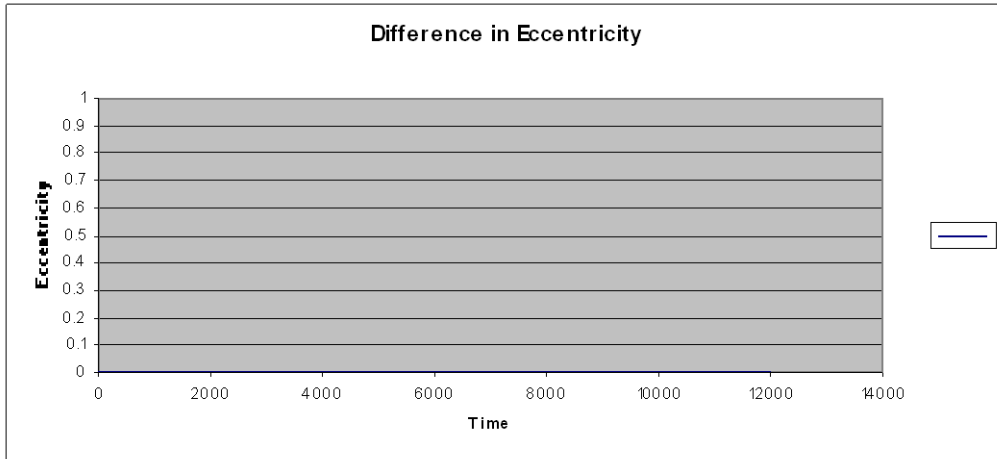


Euler-Cromer

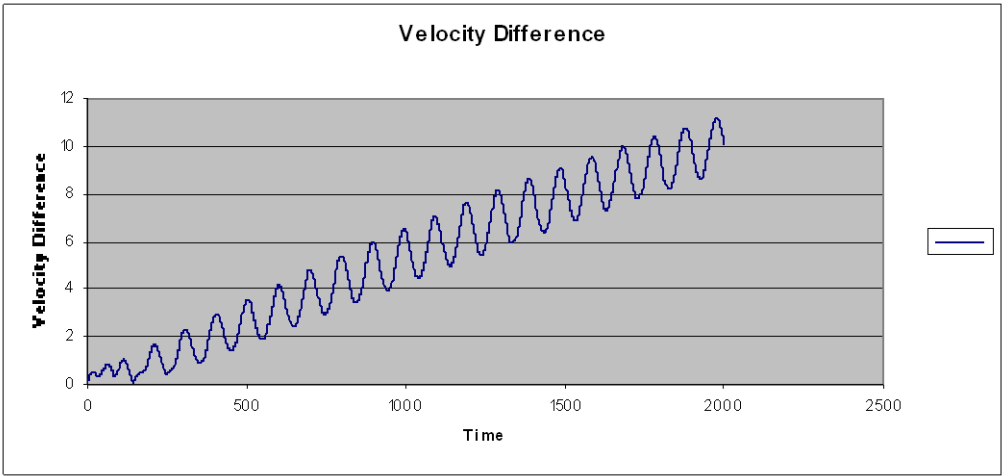
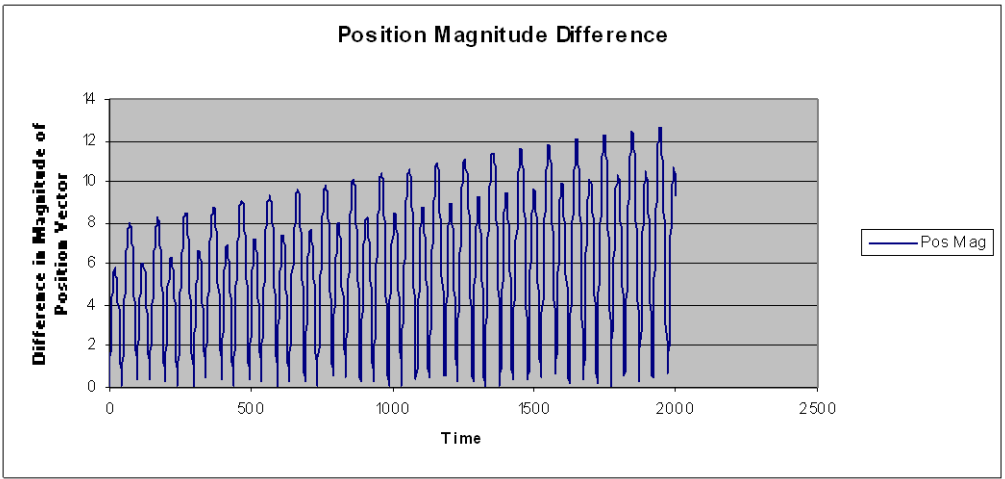
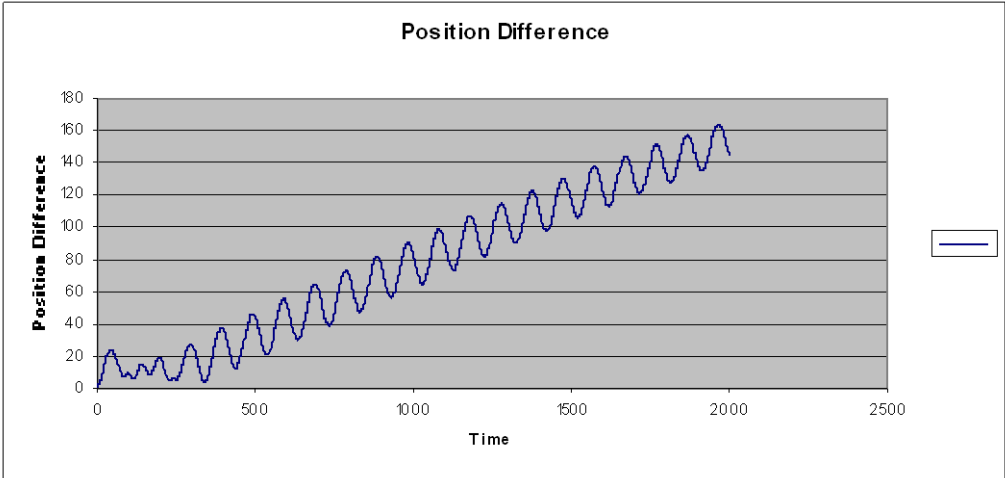
Time Step: .1

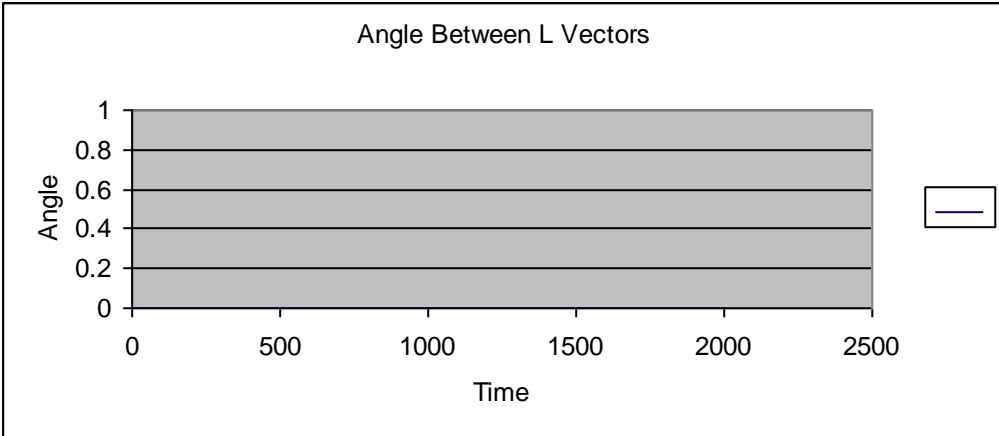
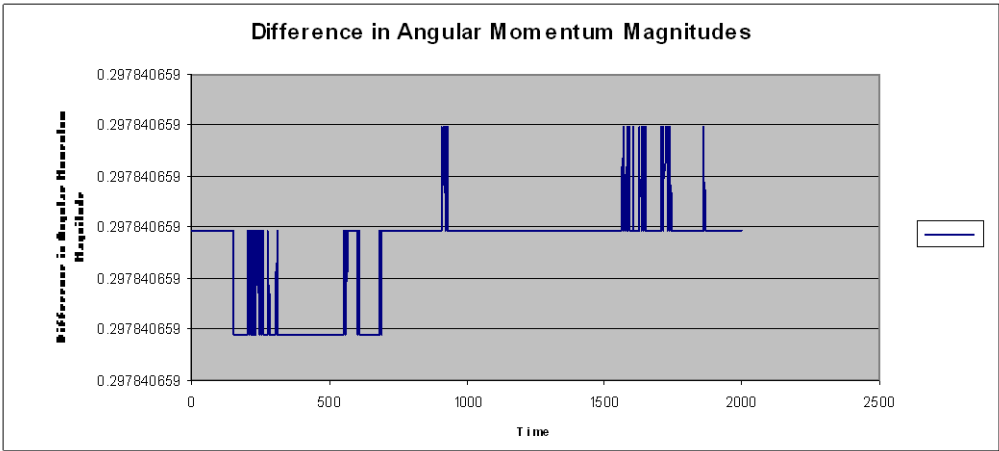
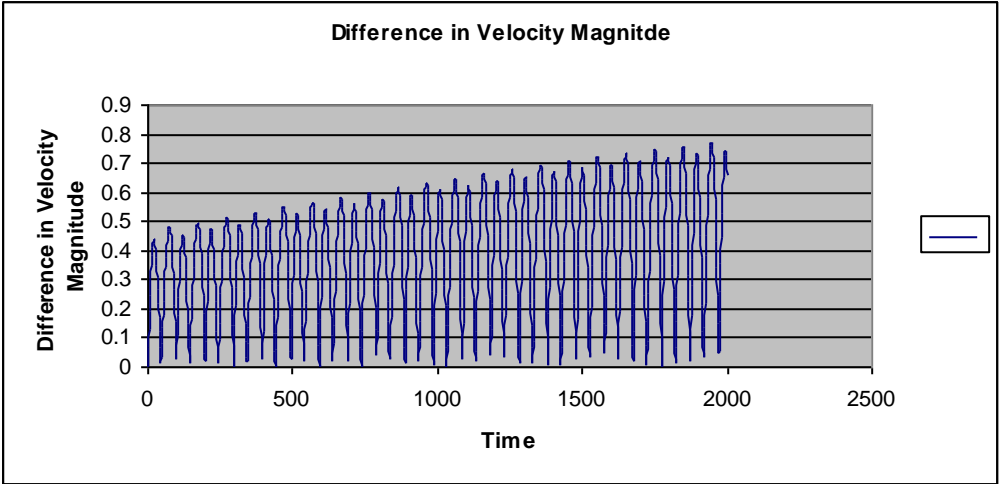


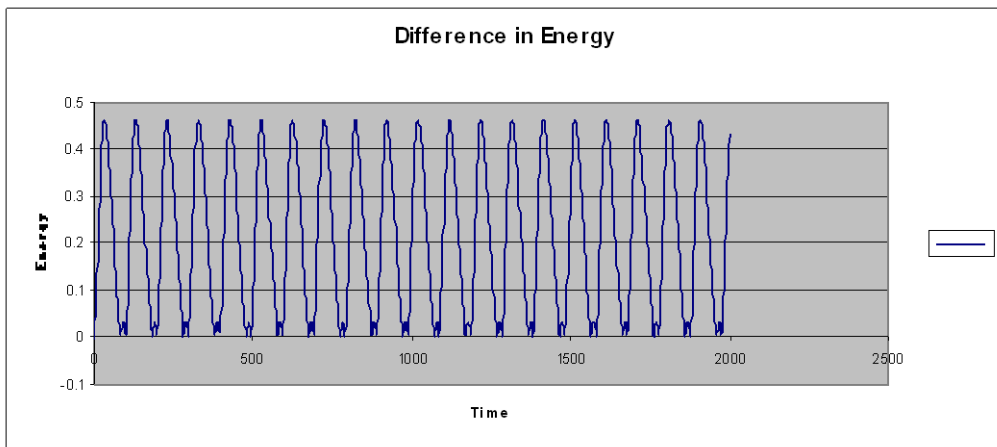
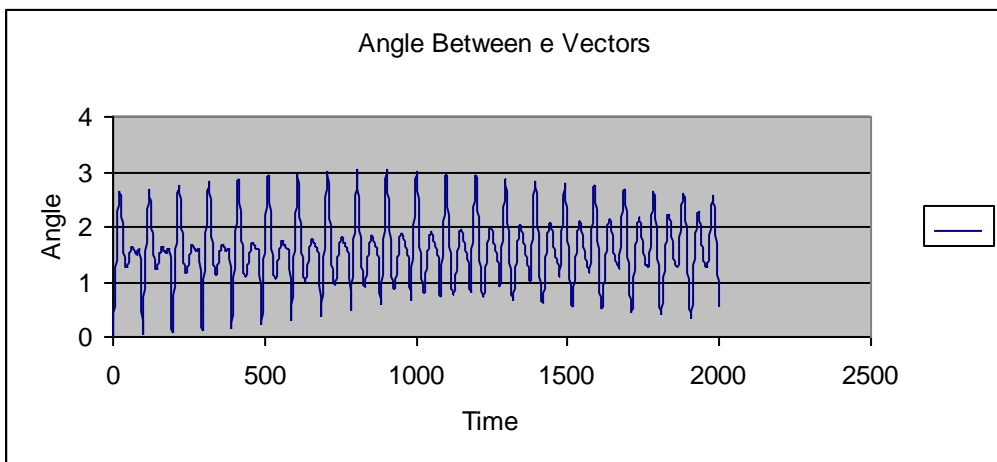
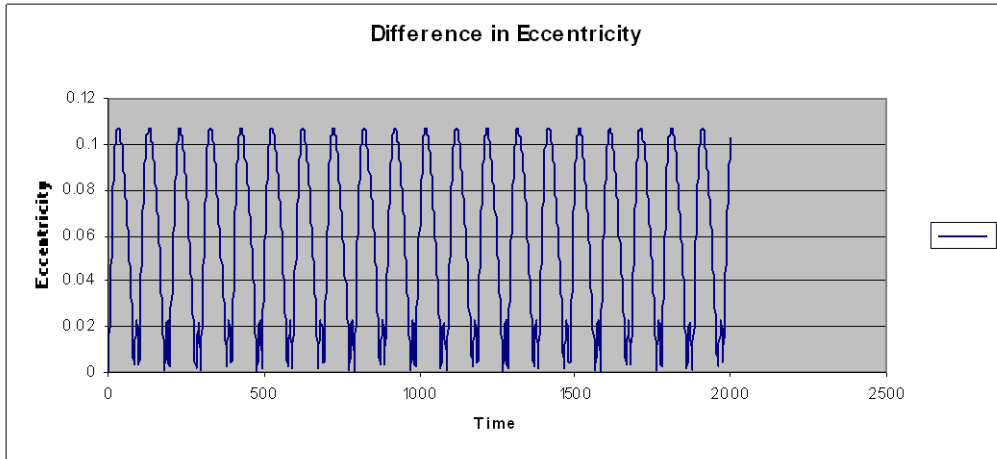




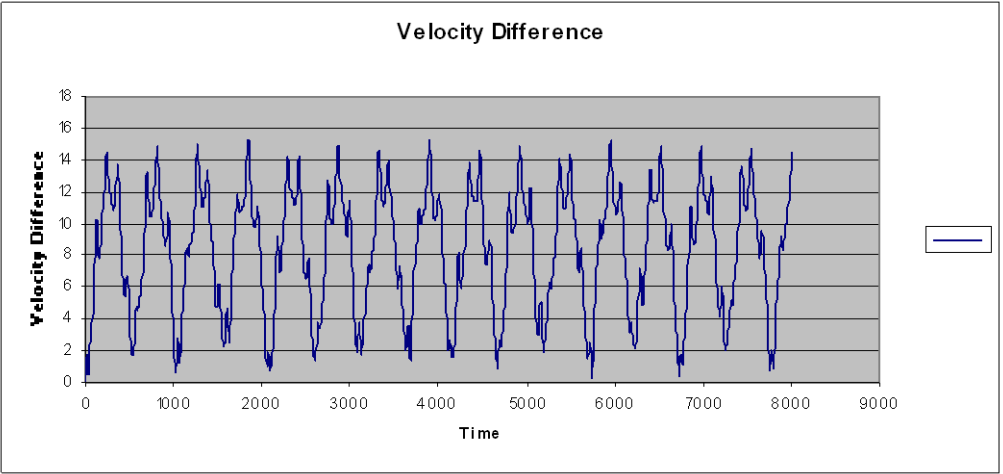
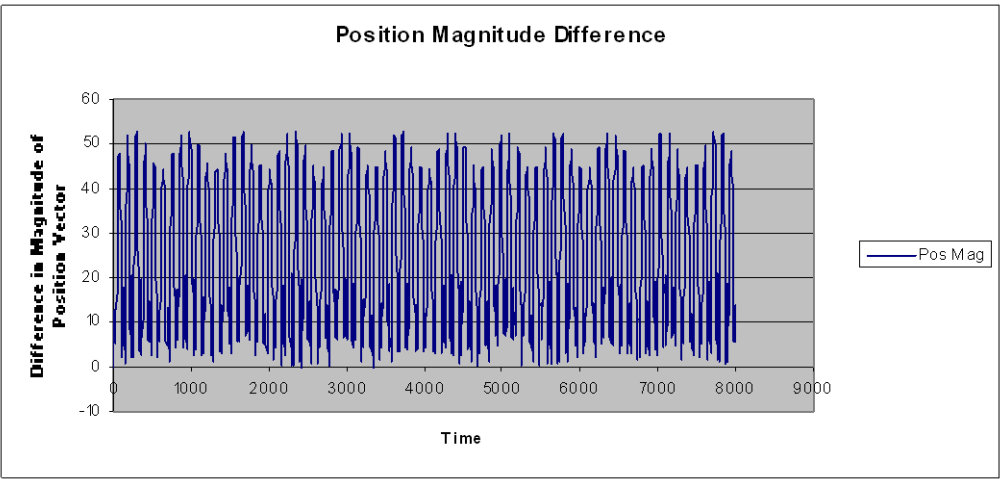
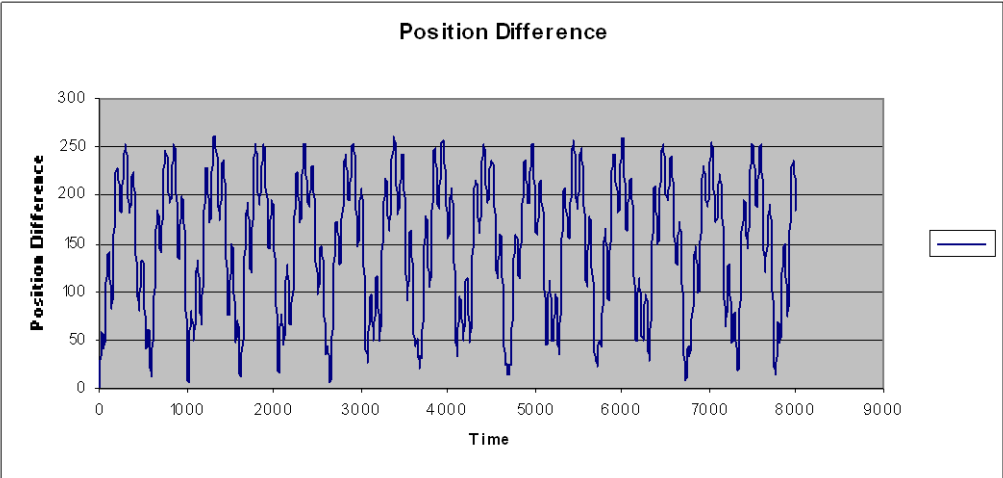
Time Step: 2

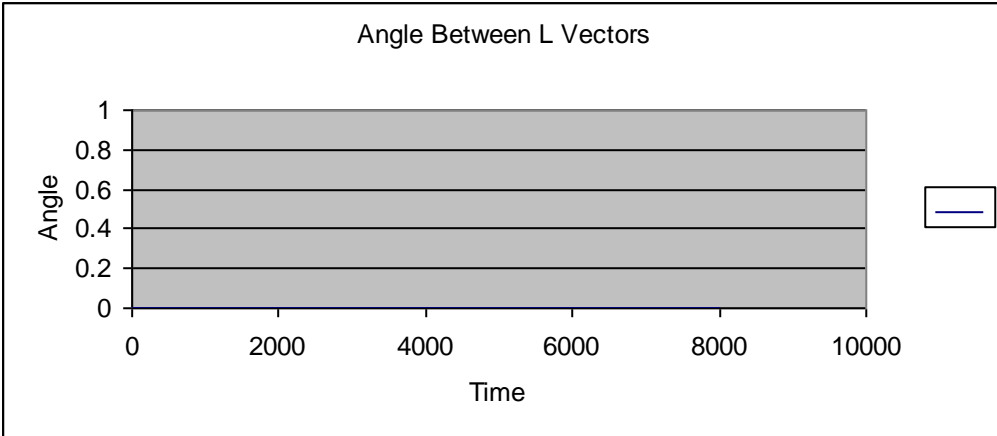
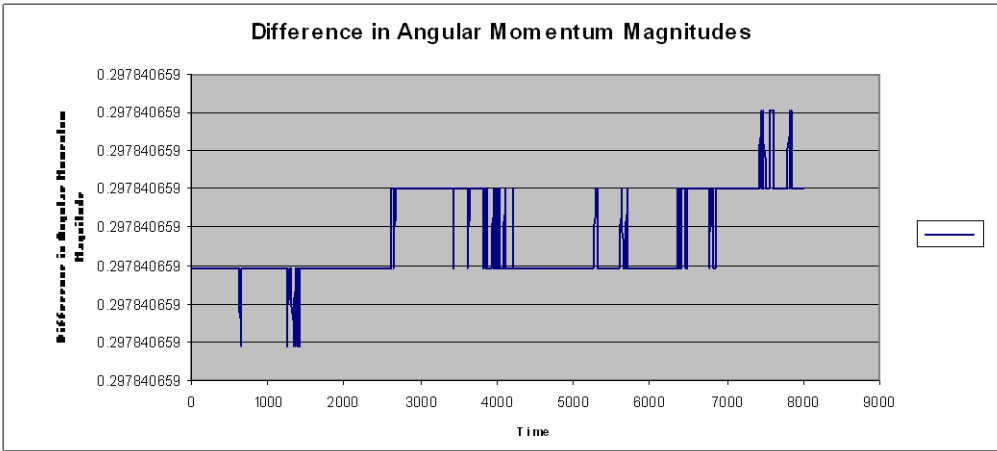
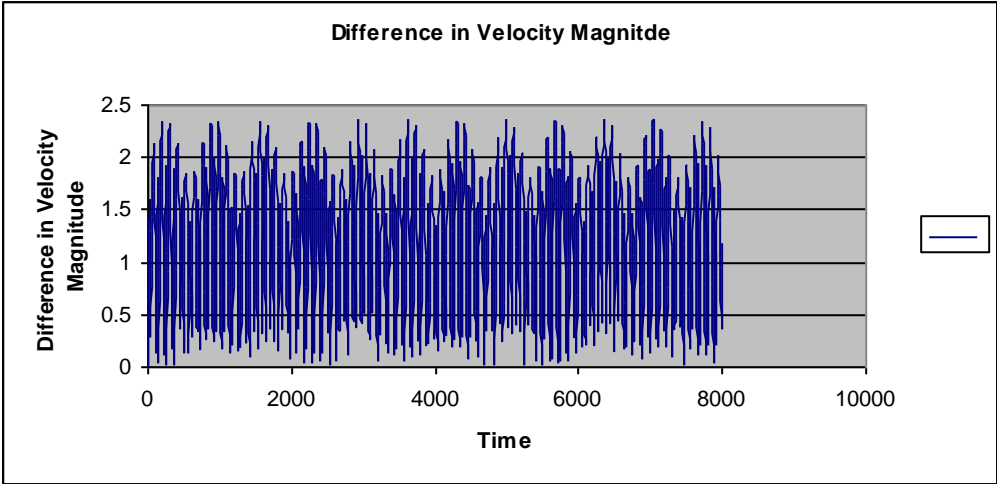


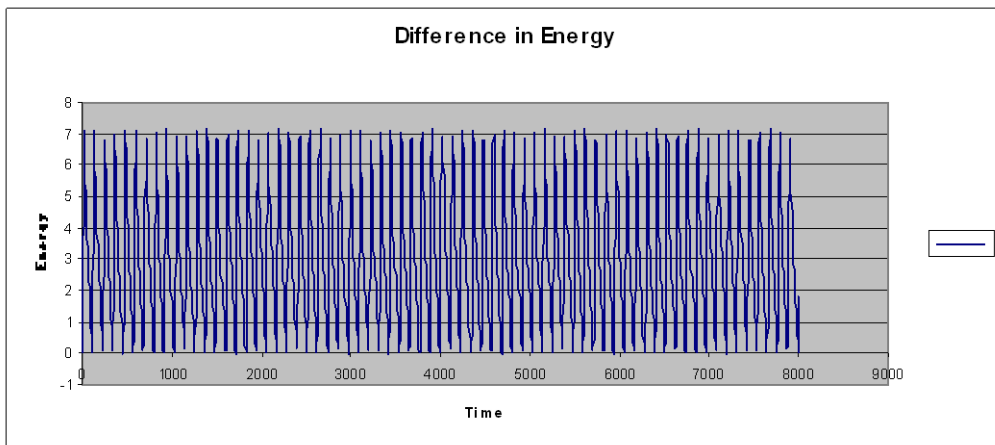
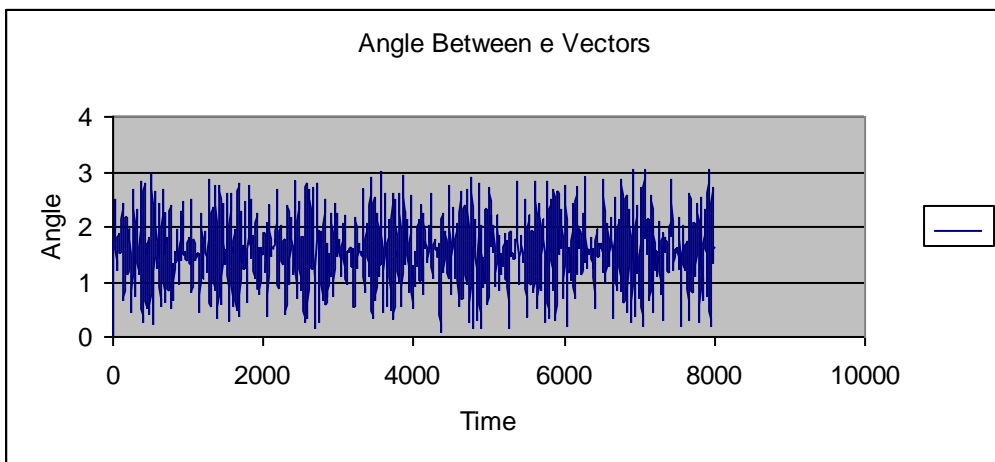
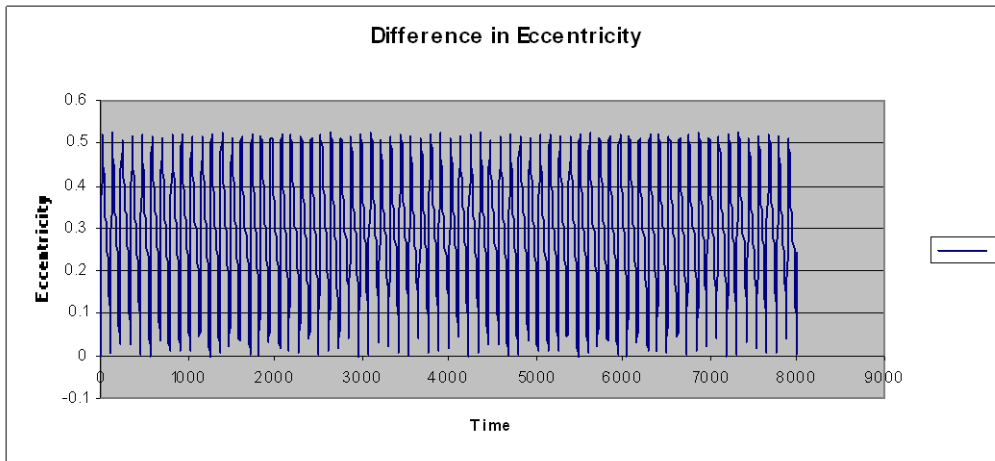




Time Step: 8

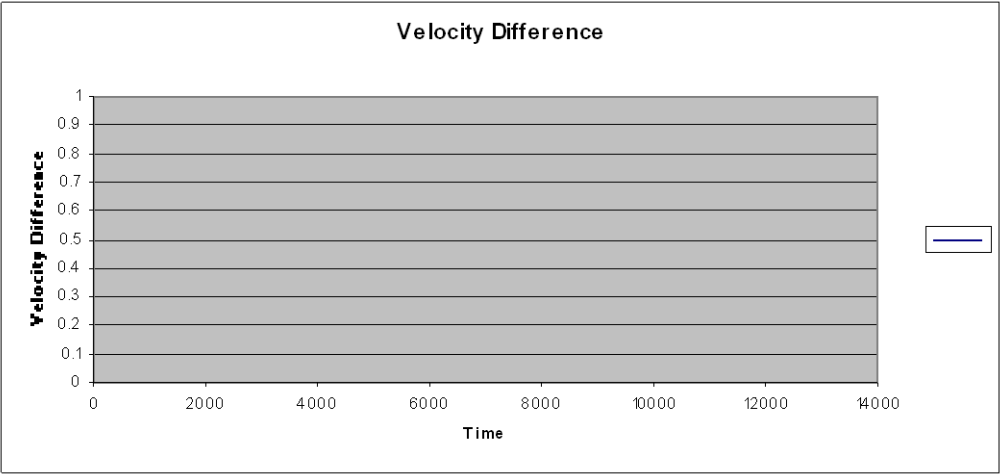
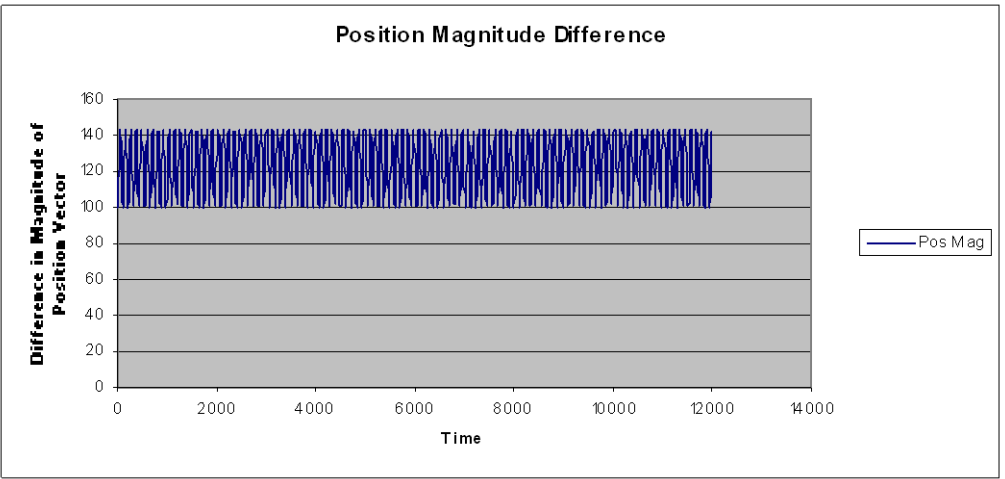
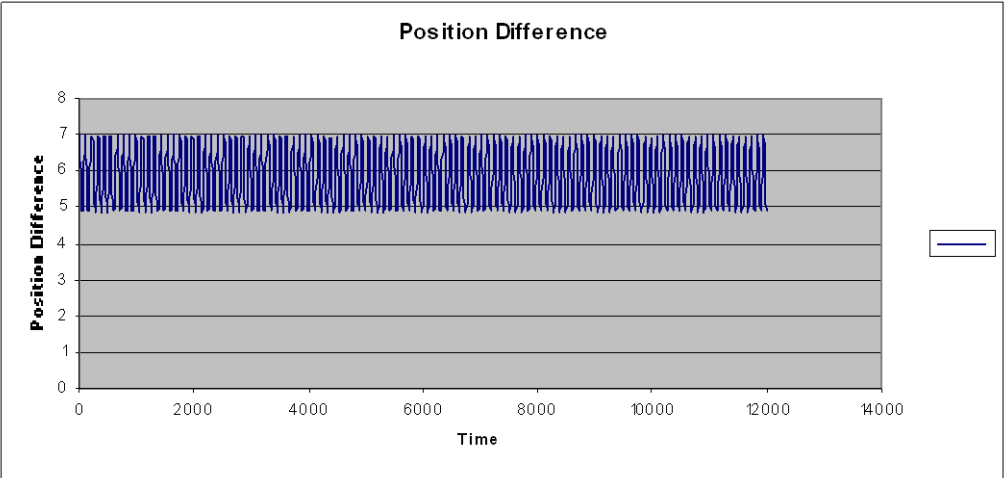


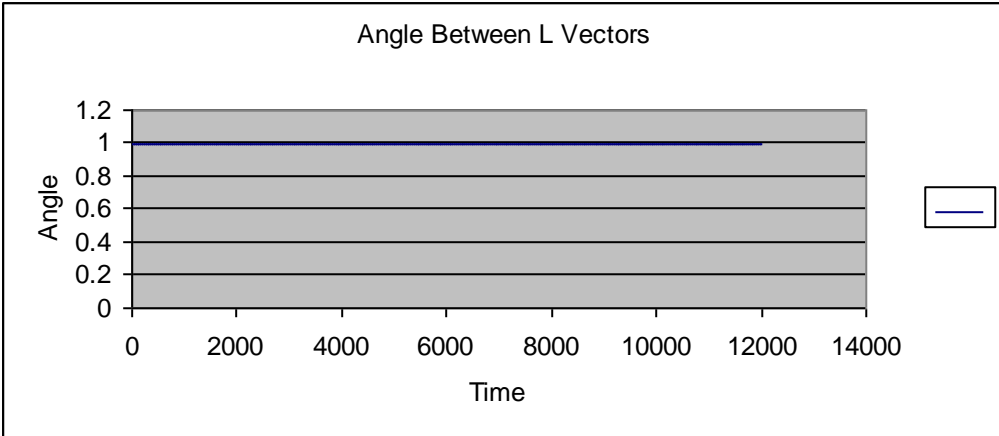
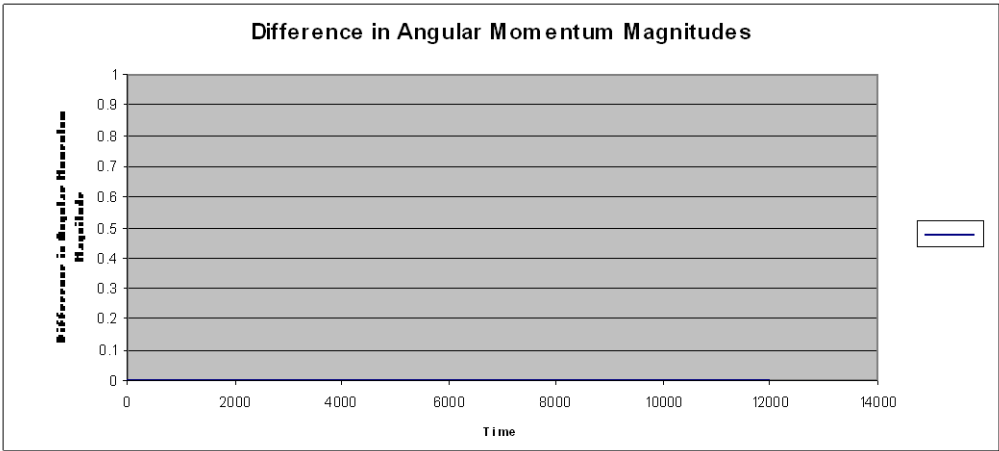
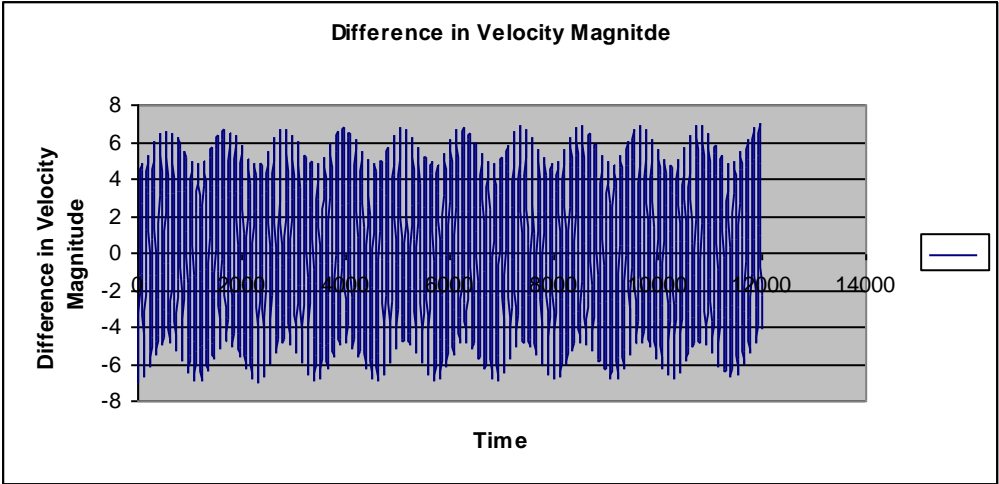


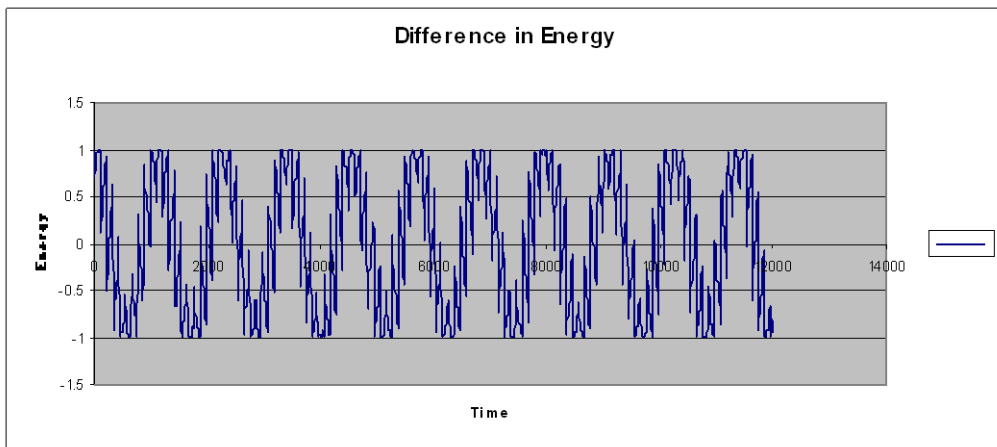
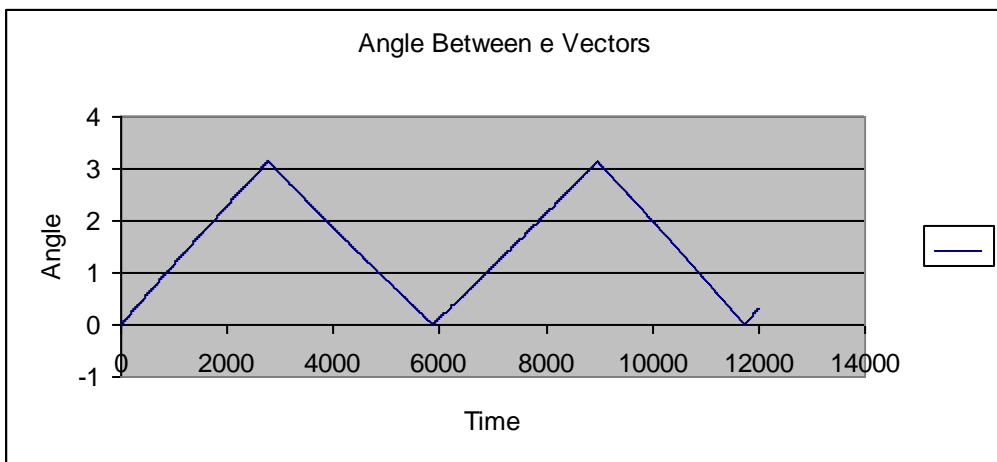
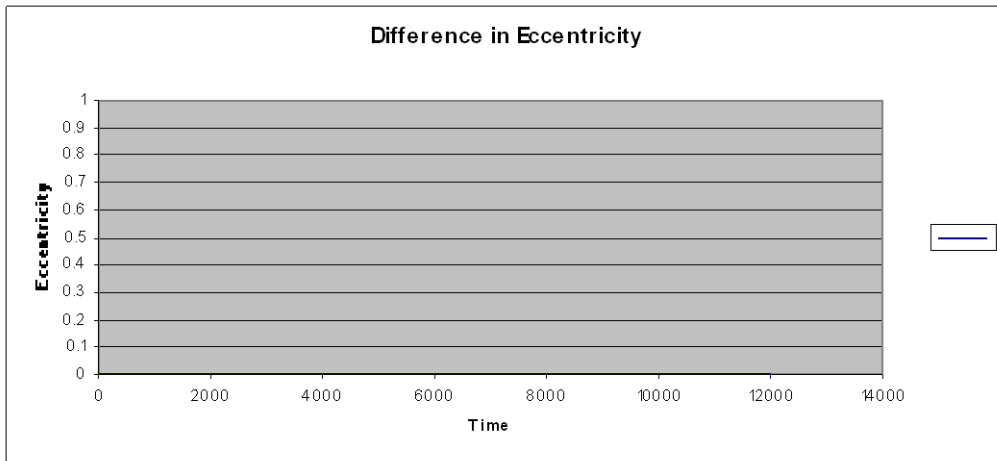


Leapfrog Method

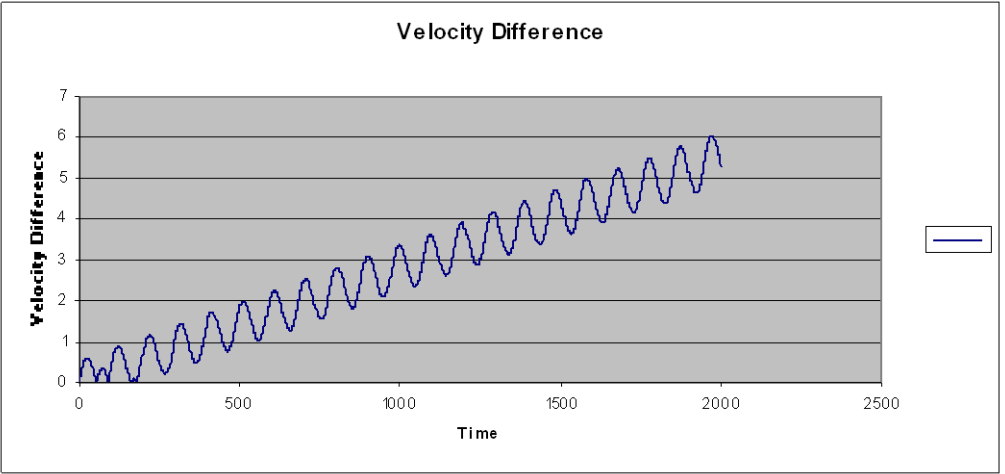
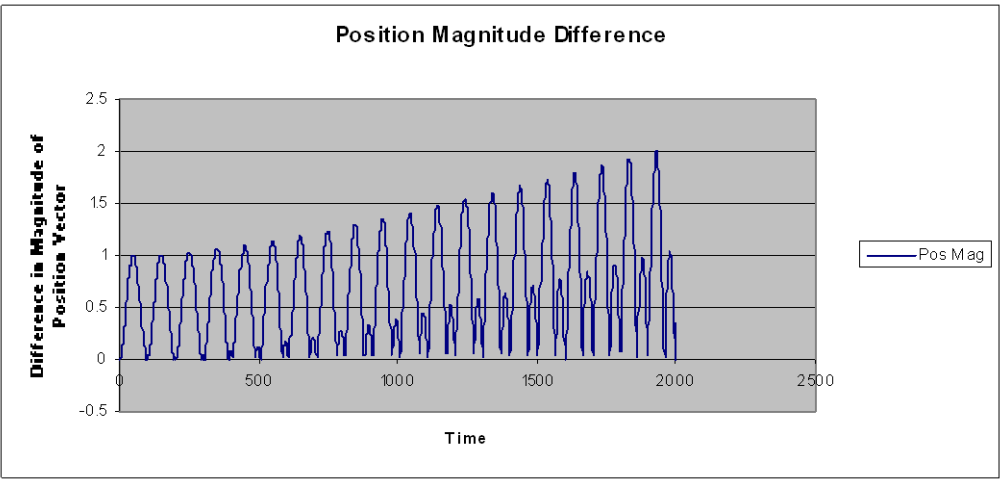
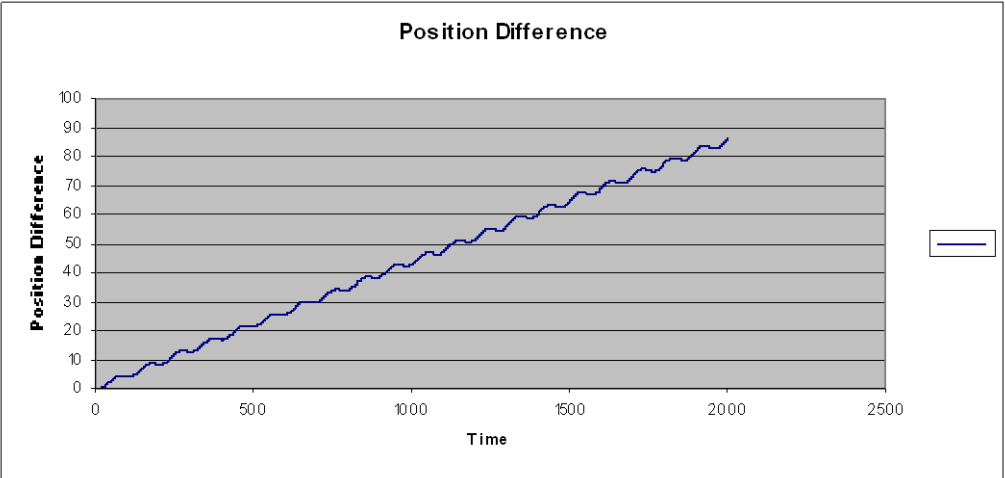
Time Step: .1

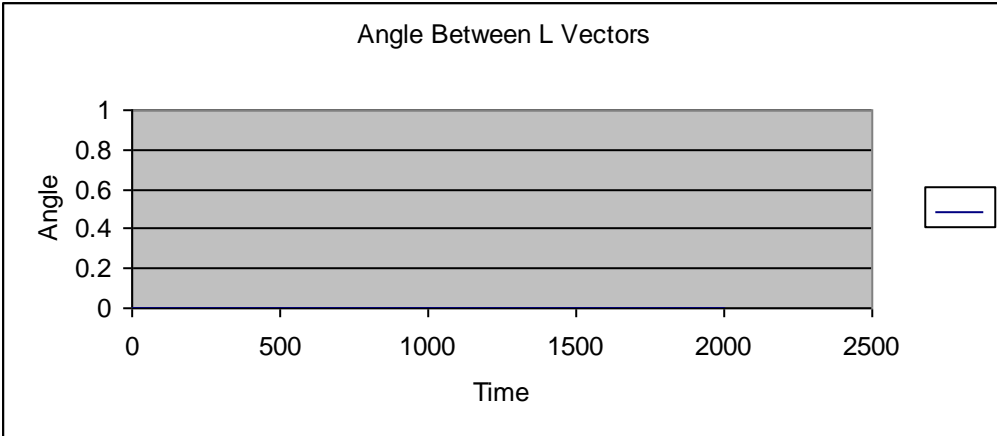
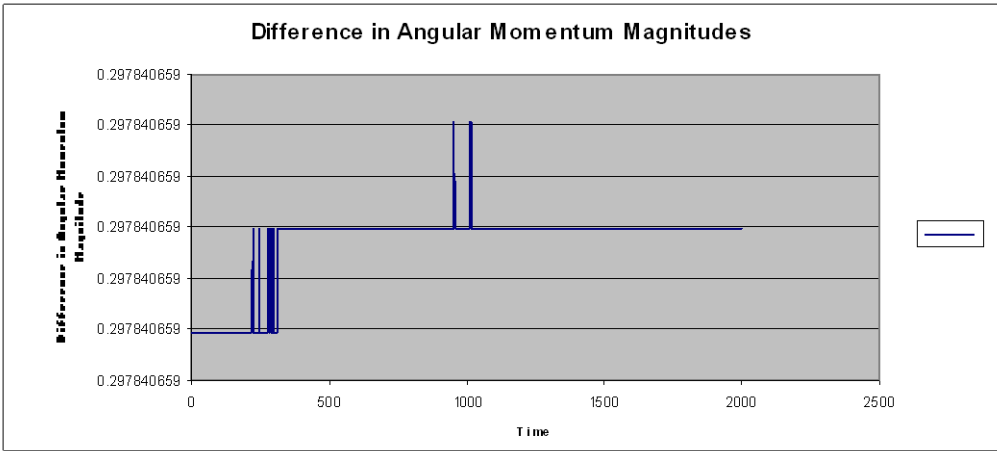
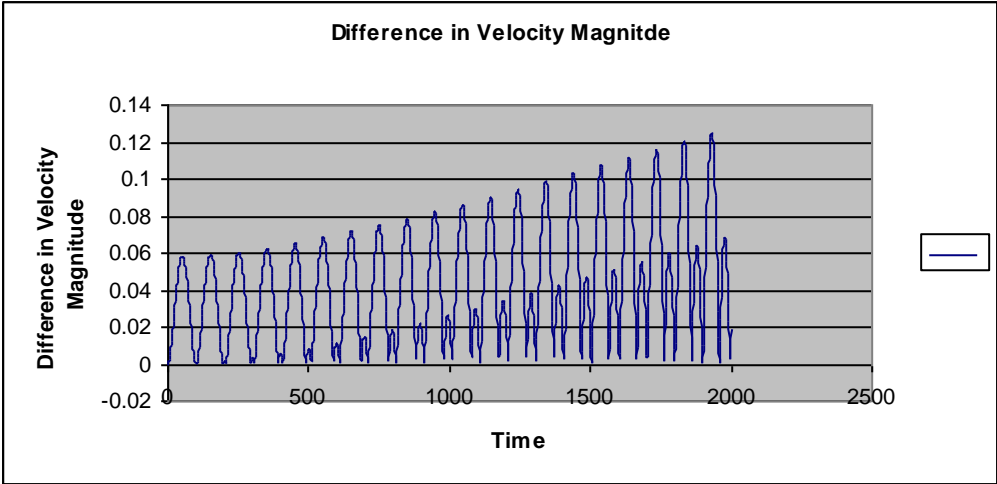


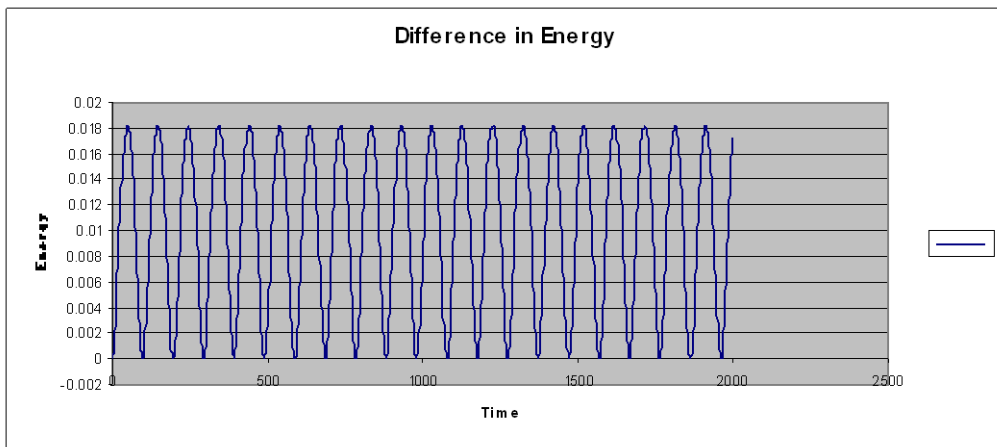
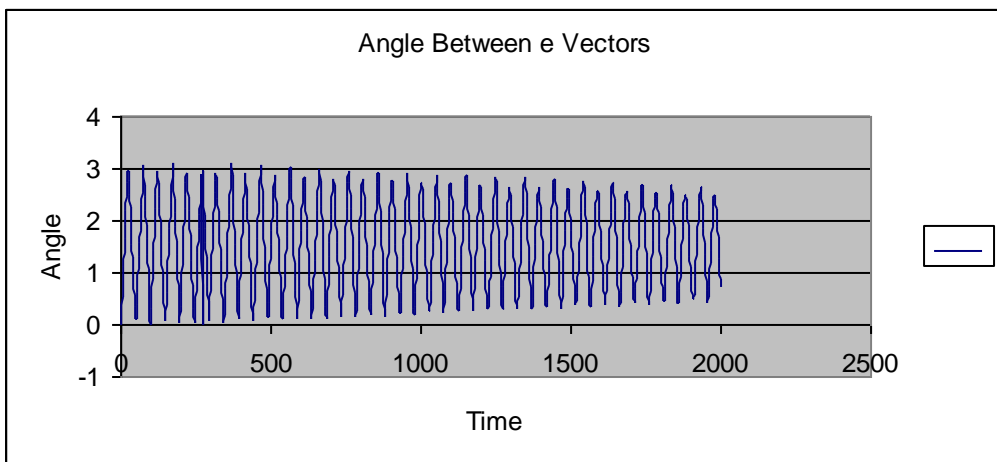
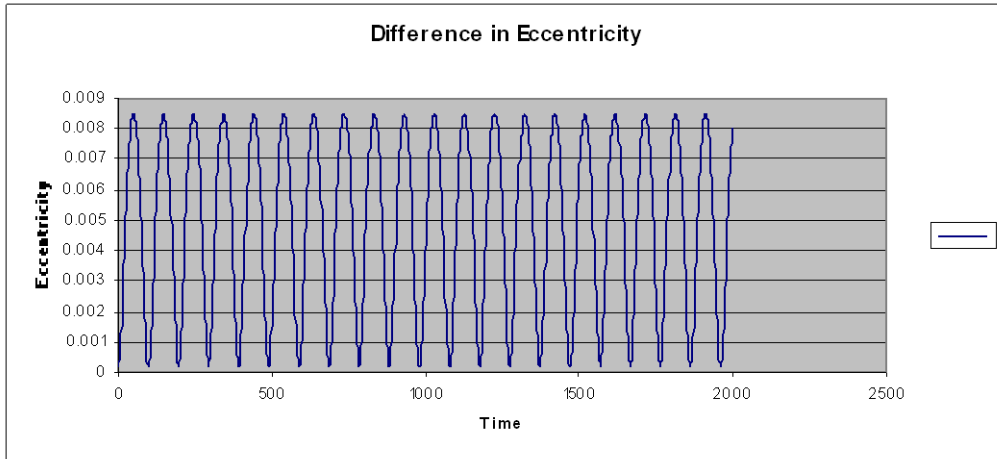




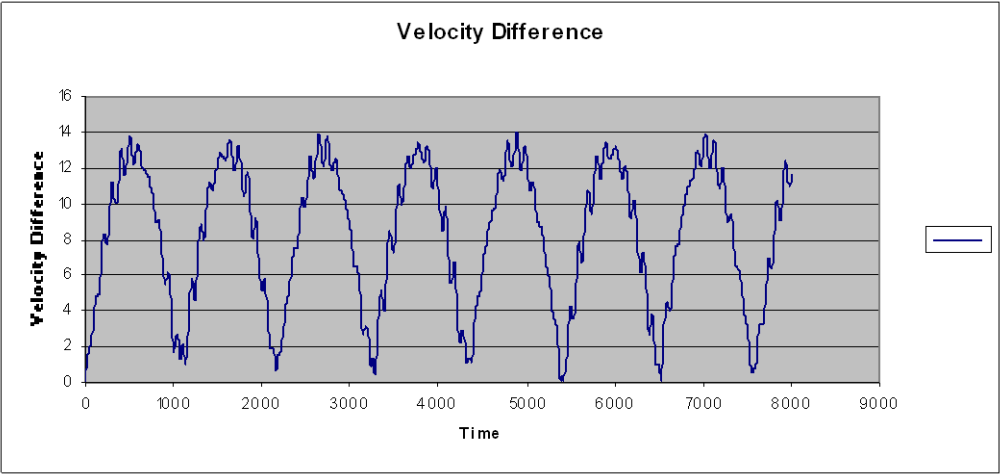
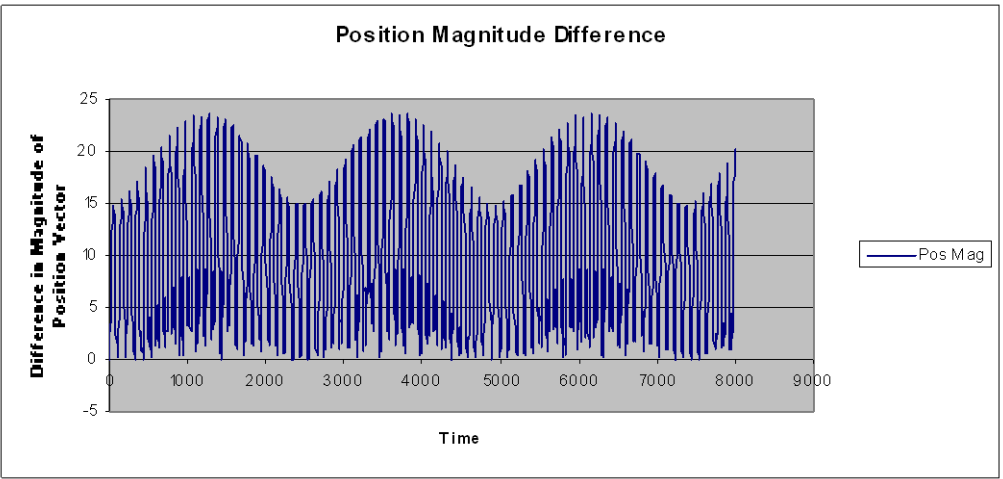
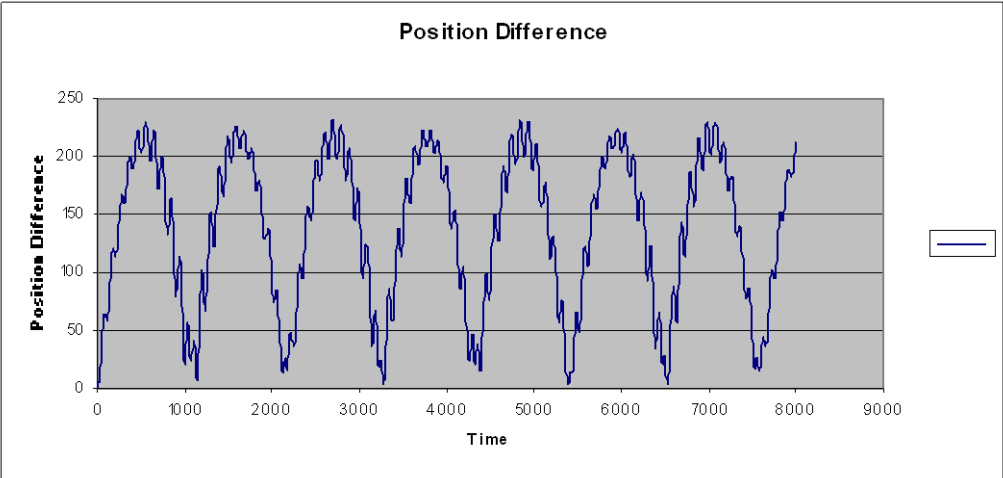
Time Step: 2

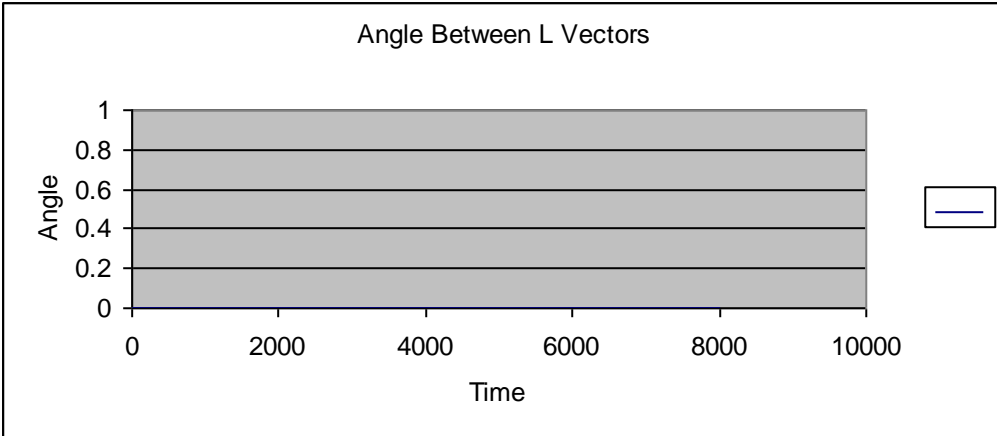
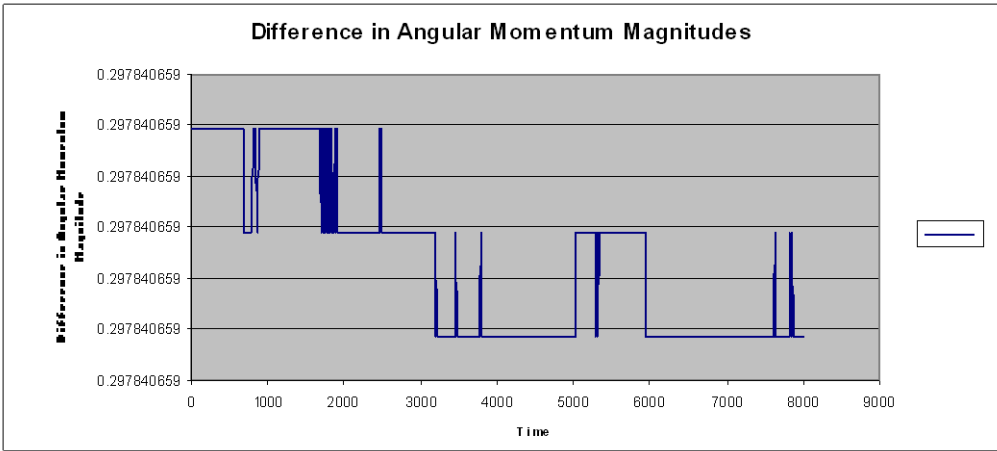
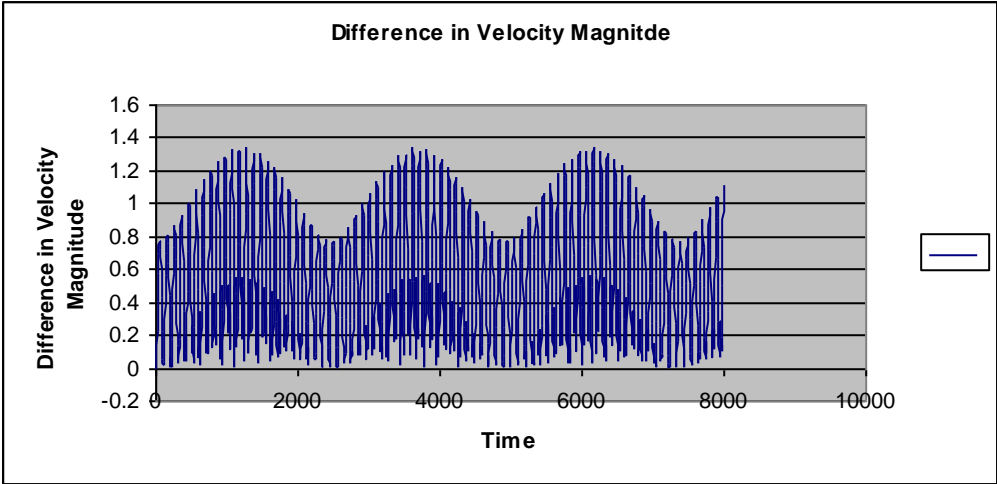


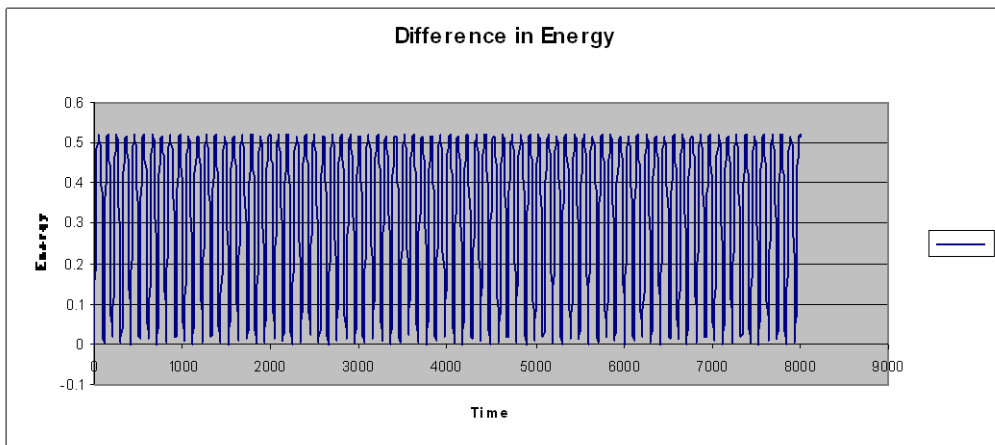
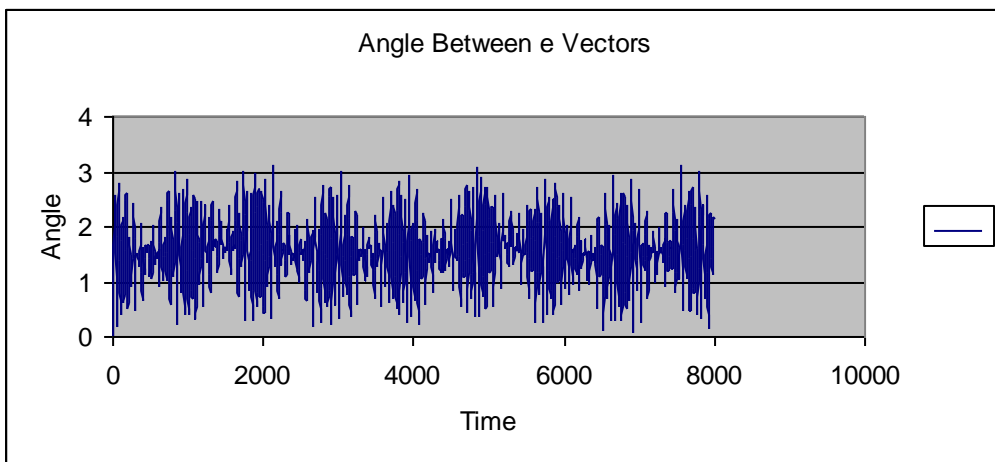
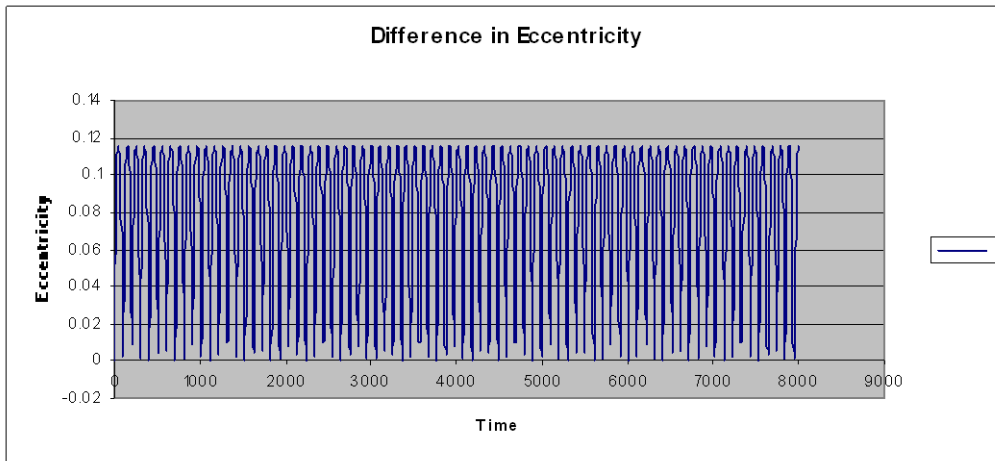




Time Step: 8

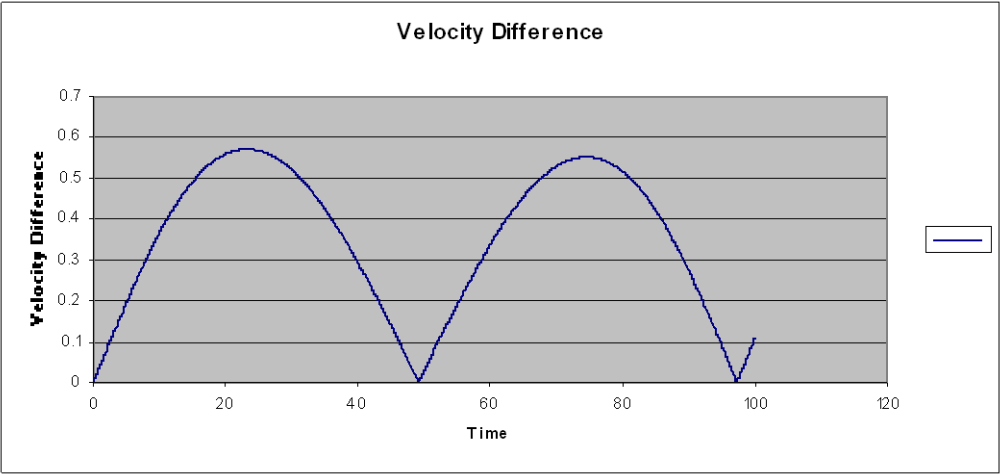
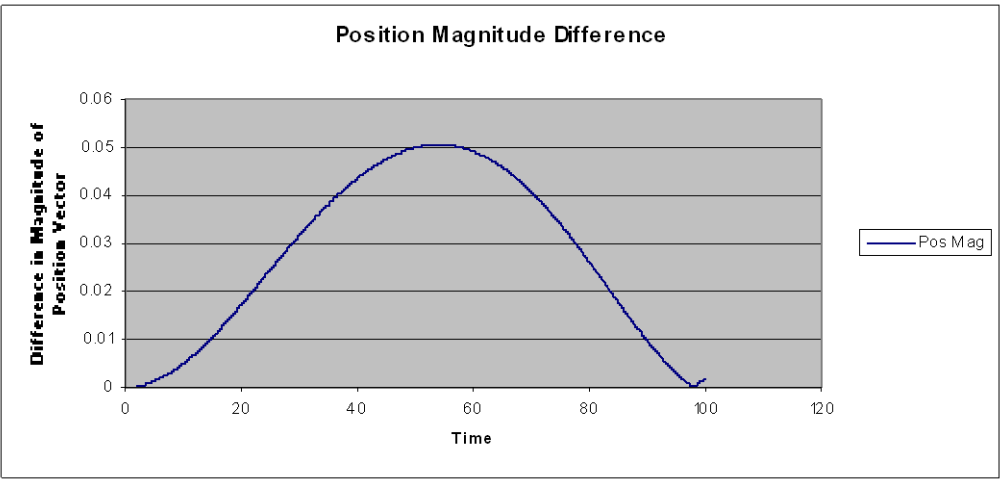
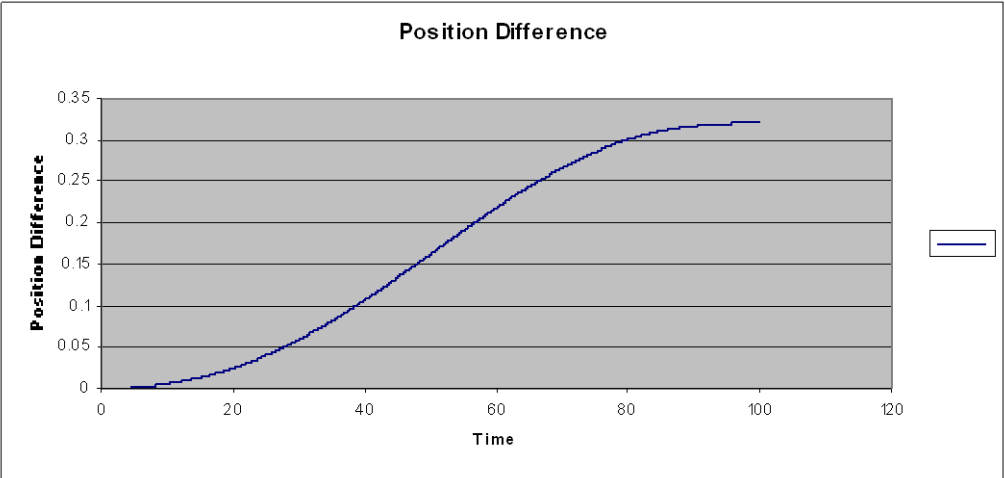


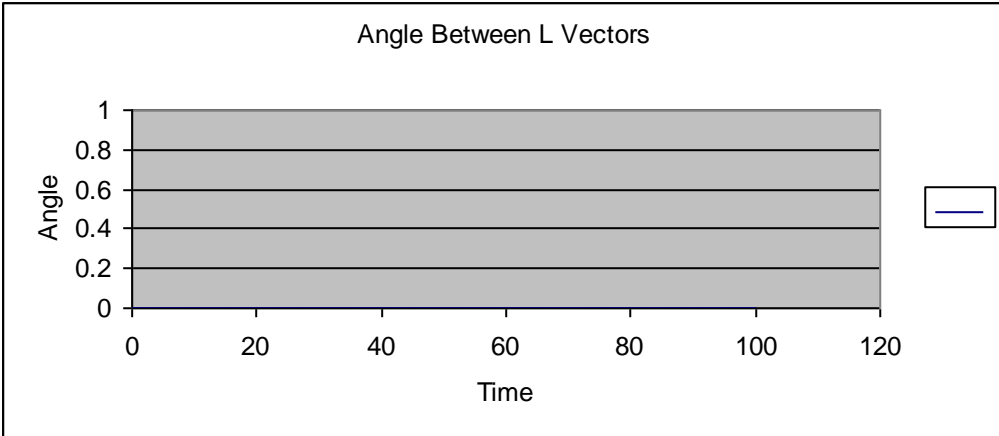
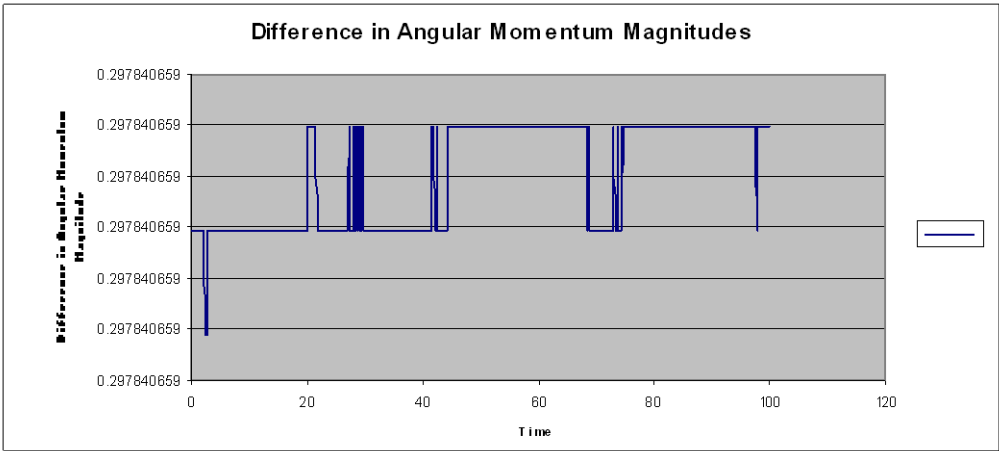
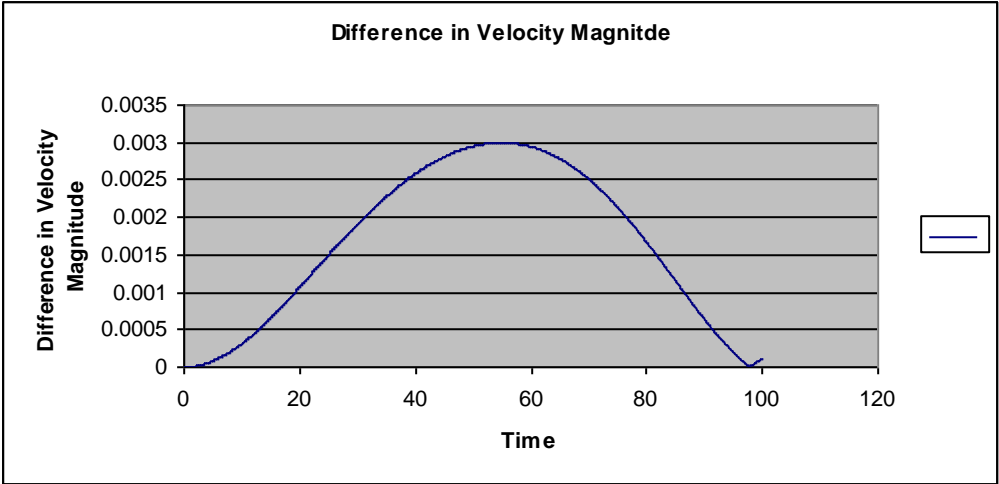


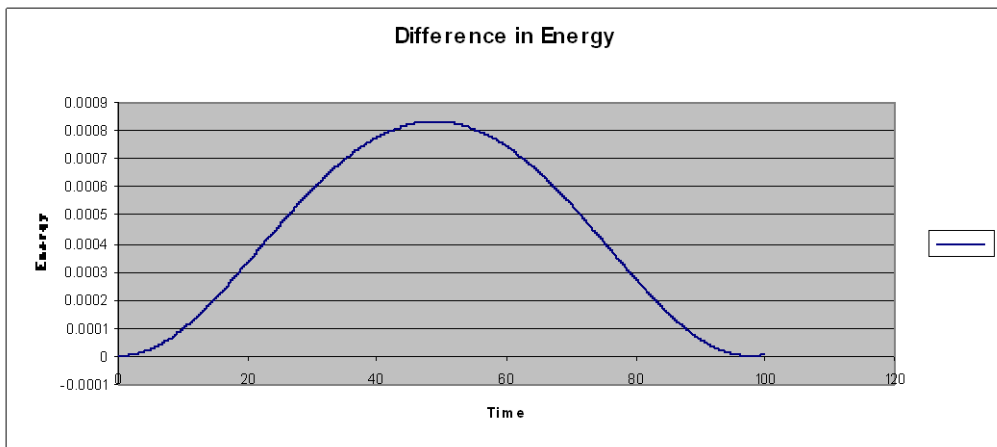
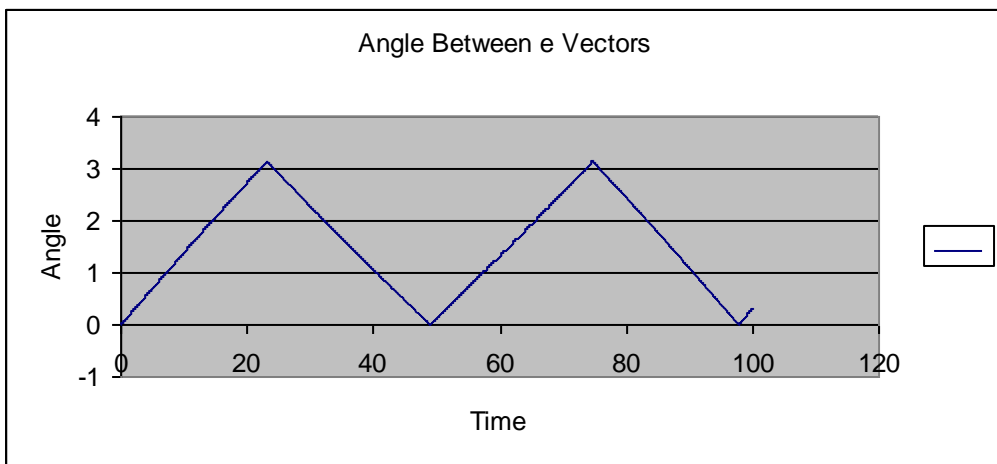
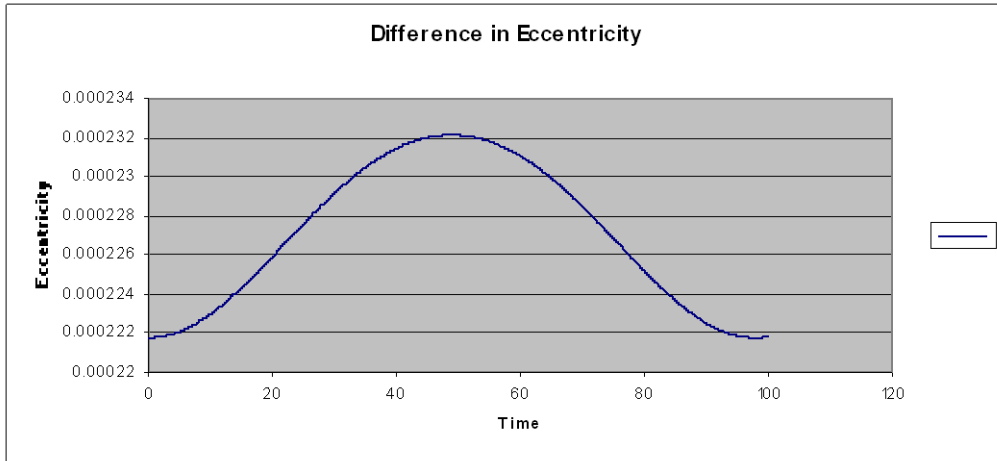


Modified Runge-Kutta 2

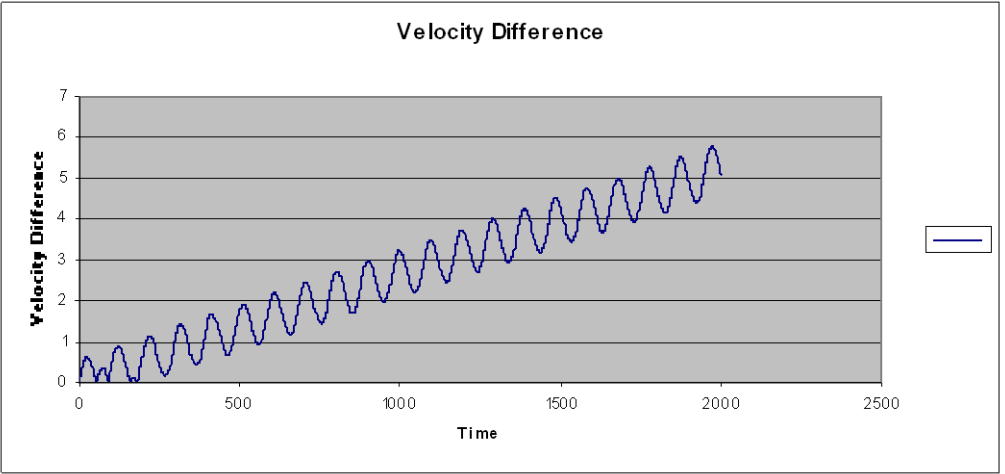
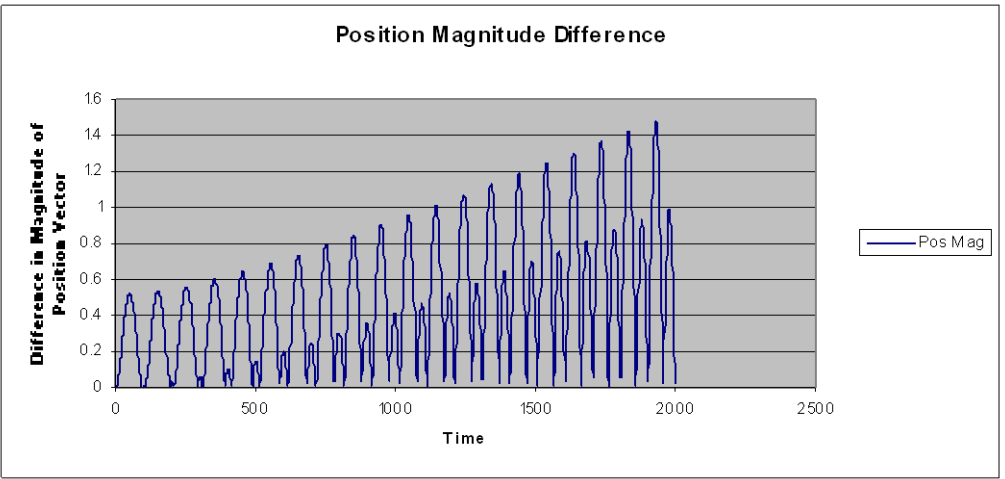
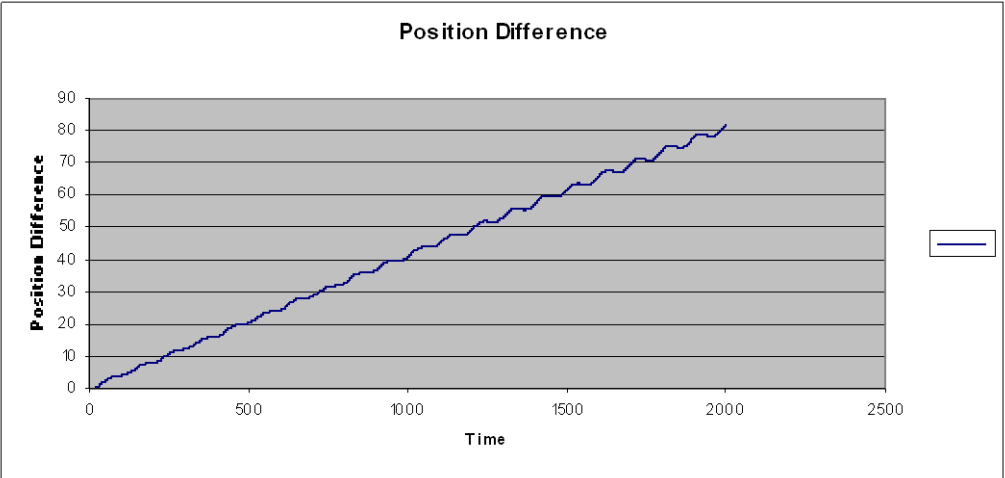
Time Step: .1

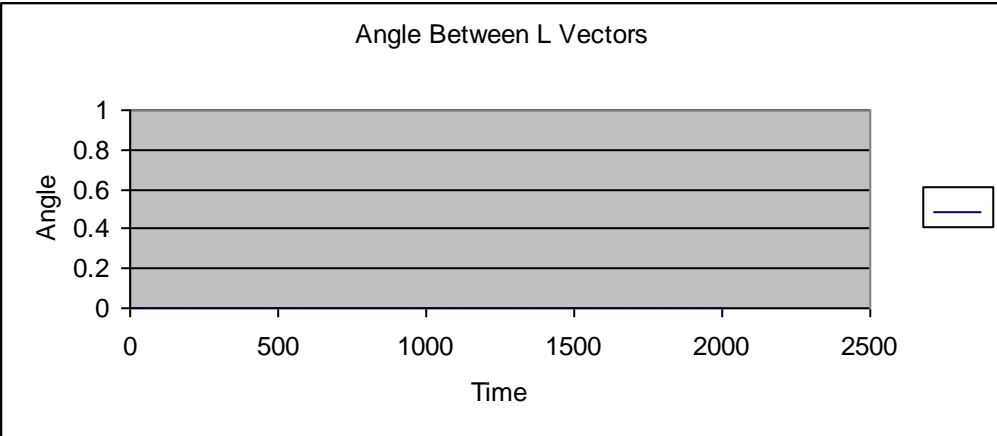
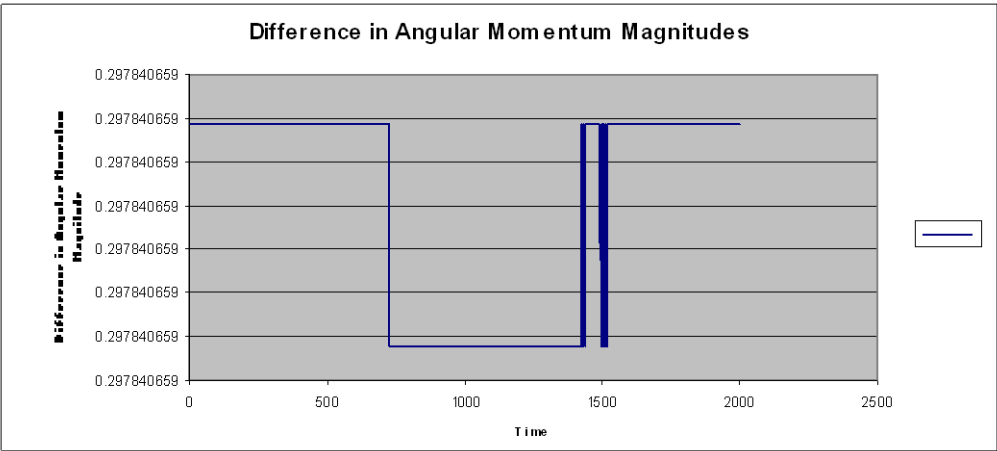
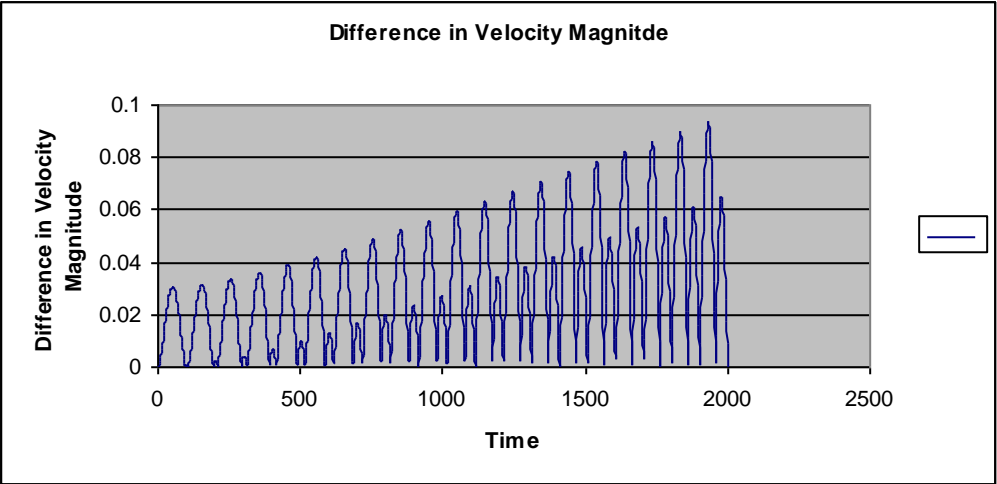


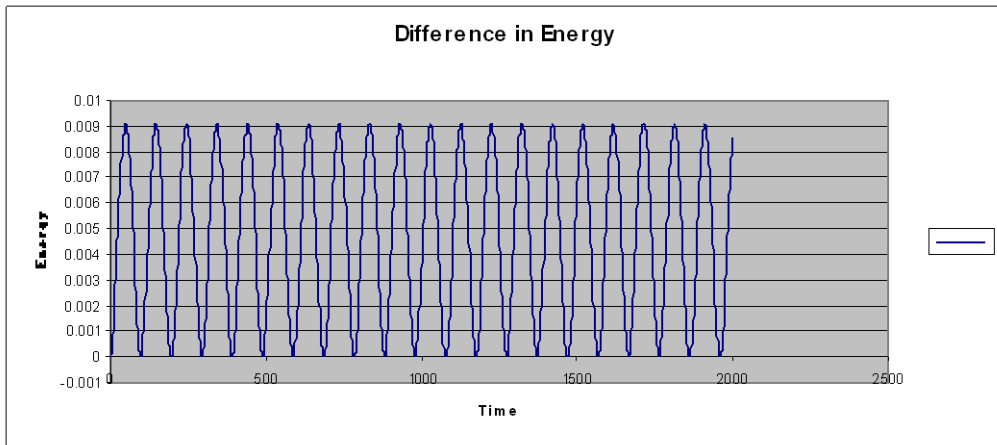
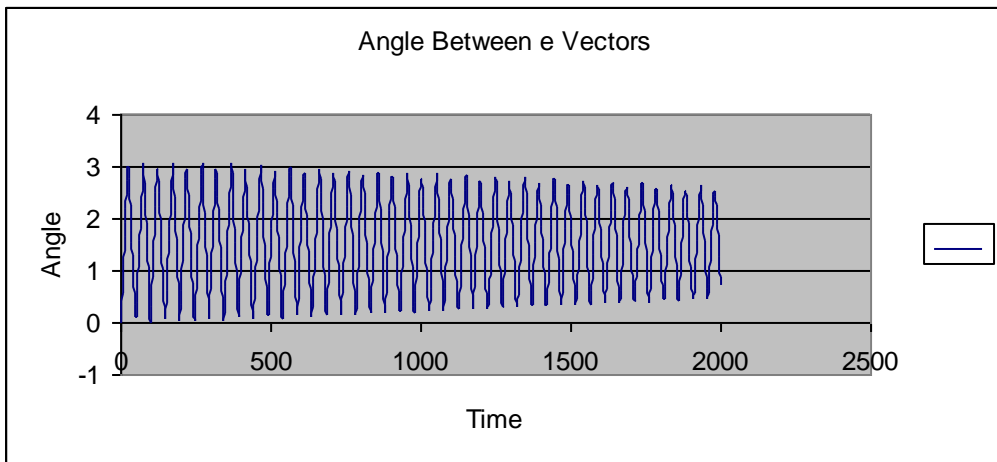
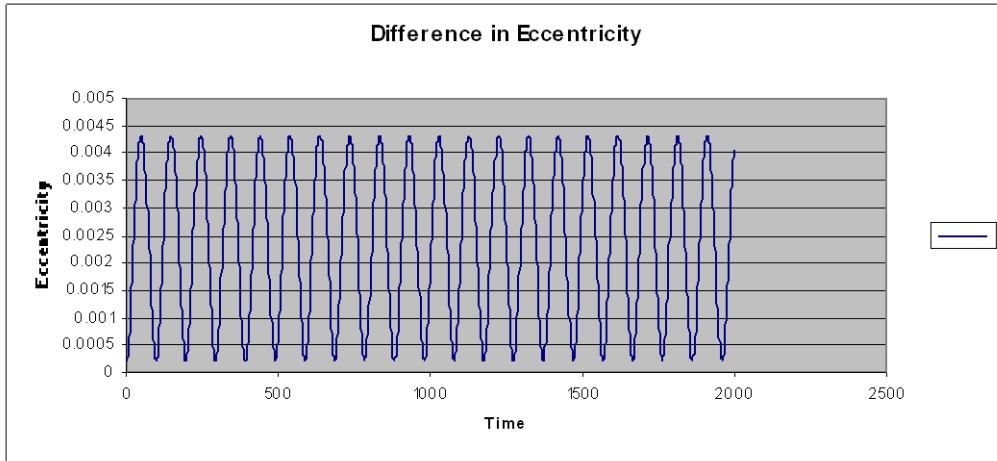




Time Step: 2







Time Step: 8

