CPLS 5100 – Major Project

Exploring Orbital Motion with Numerical Methods and Examining Numerical Method Accuracies

John Mick

Abstract

In this experiment we explore orbital motion caused by a single gravitational force in two different simulations. Our first simulation creates a starting condition which causes a satellite to orbit a body in an elliptical fashion while our second simulation will examine a starting condition which causes a circular orbit. We observe differences in the elliptical and circular orbits; such as how the speed of the satellite orbiting in an ellipse increases as it approaches the orbital perigee, while our simulated circular orbit moves at nearly a completely constant speed. Also noted is how the acceleration and jerk of a satellite peaks quickly over a short period of time near the perigee of the elliptical orbit, causing a fast and large increase in speed. Conversely the circular orbit exhibits acceleration and jerk values close to zero.

Due to observed positional error we explore nine different ODE solvers in order to explore which method may produce the least amount of numerical error. The nine methods consist of, the seven ODE solvers provided by Matlab as well as a manually implemented first order Euler method and a fourth order Runge-Kutta method. In our elliptical simulation we found only the Runge-Kutta and Euler methods to produce visually accurate orbits, where the end point of an orbital period meets the initial point it began at. When simulating a circular orbit we saw that all the solvers were able to produce circular orbits, though the error introduced in many Matlab ODE methods create a jagged "sawtooth" circle. Based on the stability that Euler and Runge-Kutta Methods demonstrated in the two simulations we elected to explore those two methods in further detail.

Our next experiment involves simulating our elliptical orbit for twenty orbital periods with both Runge-Kutta and Euler using a static timestep of .01 seconds. By visually examining the orbital paths we note that Runge-Kutta appears to diverge away from an accurate orbit slower than the Euler method. Additionally we see that both methods appear to be more accurate near the perigee of the orbits as opposed to the apogee where more positional error is present. We also observe that Runge-Kutta appears to diverge towards the gravitational force, and diverges in the X direction faster than the Y direction, while Euler's method diverges away from the gravitational force, and diverges faster in the Y direction than the X direction.

Lastly, we explore changes in our positional results and the number of times our system of equations must be applied to produce these results with the Euler and Runge-Kutta methods across varying constant timesteps and the Matlab ODE functions. This is done with both our elliptical and circular simulation run. In the elliptical case, we see that reductions in the timestep correlate to more accurate solutions by several significant digits. The Runge-Kutta method requires the most calculations, due to it's fourth-order nature, but produces our most accurate results. Several of the Matlab ODE functions diverge from a solution completely and we observe that the ODE 45 and 23 solvers are the most accurate of the Matlab ODE functions; though they do not create visually correct orbits. In the circular case, Runge-Kutta with a .1 second timestep achieves an accuracy better than Euler's method at a .0005 timestep in significantly less calculations. The errors introduced by Runge-Kutta and Euler across the experimented timesteps are all very small. Matlab ODE Solvers also perform better with the circular orbit, with ODE 23t providing the best accuracy of the Matlab ODE functions.

From the experiments we have examined that one can simulate visually accurate orbits with a variety of ODE solvers. The shape of the orbital path will determine the variability of speed, acceleration, and jerk in a satellite's movement. More variance creates a more numerically difficult simulation to accurately calculate. With a short enough time span in which a simulation is going to be executed and a large enough visual coordinate scale, one may create a simulation which visually appears to be quite accurate using Runge-Kutta, Euler, or even perhaps one of Matlab's ODE solvers. On the other hand, if your intention is to create something numerically accurate to more than 10 significant digits, other more effective methods may need to be explored as the number of calculations required to increase the number of accurate significant digits begins to increase dramatically.

1. Introduction

In this set of experiments we intend to use a set of Newtonian equations to simulate orbital motion of a satellite body that is being pulled on gravitationally by a single mass. We simulate two scenarios from the same satellite starting position but with different initial velocities, one causing an elliptical orbit while the other creates a circular orbital path.

We intend to explore how the speed, acceleration, and jerk of the satellite changes in both of our simulations. Our expectation is to see more variability in the motion of the elliptical orbit than in the circular orbit.

We also will examine how nine different ODE solvers produce varying orbital position results. Next we will identify positional error behaviors between a Runge-Kutta solver and an Euler solver.

Lastly we delve more in depth, comparing which methods produce less error as time progresses, as well as taking note of the number of times our system of equations must be evaluated to produce our positional results. The intent is to gain a better appreciation for how the shape of the orbital path and the numerical method you apply can impact the results and the number of computational steps required to achieve different levels of accuracy.

2. Formulation

Equations:

Newton's Universal Law of Gravitational Force:

G: Universal Gravitational Constant
M: Mass of Gravitational Force Source
m: Mass of the Satellite
d: Distance of Satellite from Gravitational Force Source

Force = G * M * m / d^2

Distance Between the Satellite and Gravitational Force Origin (Pythagorean Theorem):

Distance (km) = Square Root ((Satellite X – Force X)^2 + (Satellite Y – Force Y)^2)

Angle Between The Gravitational Force Origin and Satellite:

Theta = ArcTan2(Satellite Y – Force Y, Satellite X – Force X)

Acceleration in the X Direction:

UDOT (km/s^2) = Force * Cosine(Theta) / Mass of Satellite

Acceleration in the Y Direction:

VDOT (km/s^2) = Force * Sine(theta) / Mass of Satellite

Velocity Required to Enter Circular Motion:

M: Mass of Gravitational Force Source
G: Universal Gravitational Constant
d: Distance of Satellite from Gravitational Force Source

Velocity (km/s) = Square Root (M * G / d)

Constants:

Universal Gravitational Constant (G): **-6.67300e-11**
Mass of the Object Creating the Gravitational Force: **1e13 Kilograms**
Mass of the Satellite: **15 Kilograms**
Mass of the Object Creating the Gravitational Force Constant XY Coordinate: **(0, 0) km**
Initial Satellite XY Position: **(75, 0) km**

Elliptical Orbit Simulation Run Initial Conditions:

Satellite X Velocity (U): **0 kilometers per second^2**
Satellite Y Velocity (V): **.5 kilometers per second^2**

Circular Orbit Simulation Run Initial Conditions:

Satellite X Velocity (U): **0 kilometers per second^2**
Satellite Y Velocity (V): **2.9828 kilometers per second^2**

Considering our initial satellite XY position constant for both simulations, we define an orbital period to be completed when the Y position of a satellite at time 't-1' is less than zero and the position at time 't' is greater than or equal to zero.

3. Methods

In order to determine our change in position over time we must calculate a series of equations. For any give time step we begin by calculating the distance the satellite is from the mass creating the gravitational force, using the Pythagorean theorem. Knowing this distance value, we calculate force using Newton's Universal Law of Gravitational Force. Next we calculate the angle, theta, between the satellite and the mass exerting the force with the four-quadrant inverse tangent function provided by Matlab, atan2. With theta and force determined, we use our acceleration equations to find our acceleration in the X and Y direction. Jerk is determined by finding the difference between this calculated acceleration and the previously calculated acceleration. Our new velocities are found by incrementing our previous velocity with this new acceleration value multiplied by a defined timestep. Lastly our new position is found by incrementing the previous position by our newly found velocities multiplied by the same defined timestep. This process is also reflective of how our Euler solver is implemented.

We modularized this process into a Matlab function which could be reused by all of Matlab's ODE Solvers, as well as a fourth-order Runge-Kutta solver that we implemented earlier in the semester. The Runge-Kutta method applies this series of equations four times and uses the results from each iteration to derive a final value to be stored for later use.

4. Integrity

All conclusions drawn in these sets of experiments are limited in that they have only two different simulations to draw from. Changes in the satellite's initial position and velocity may present new observations which can not be made in only these two simulations. Also changes in the mass of the body creating the force, which would cause a satellite in the same starting position to experience higher speeds, may also create new conclusions onto the numerical method accuracies.

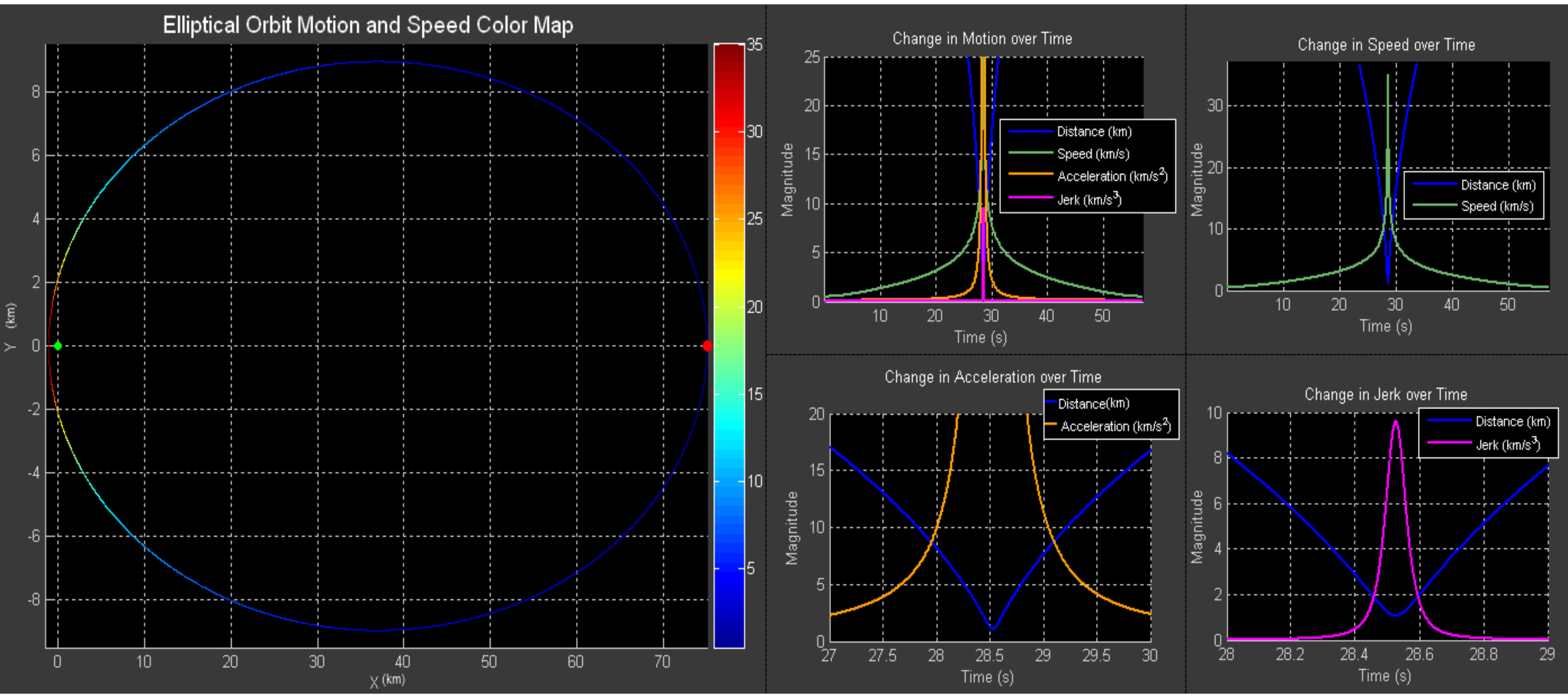
Additionally, this simulation is unrealistic due to the fact that it only models force acting on the satellite. In reality the body creating the force would also be pulled by the satellite, though our current constants would mean this unsimulated force would be extremely small. Implementing a solution which takes this into account would help pave the way towards a N-body simulation.

If our intention is to create a realistic simulation, depending on the environment in which the orbital simulation is meant to represent, we would anticipate many other forces acting on the satellite in addition to gravity. In a near Earth simulation, we would need to include forces for solar radiation pressure, magnetic forces, atmospheric drag, account for variability in the Earth's gravitational field, and gravitational pull from the Moon, Sun, third body forces, and the non-central gravitational pull from the surface of the Earth.

5. Computational Experiments (Model runs)

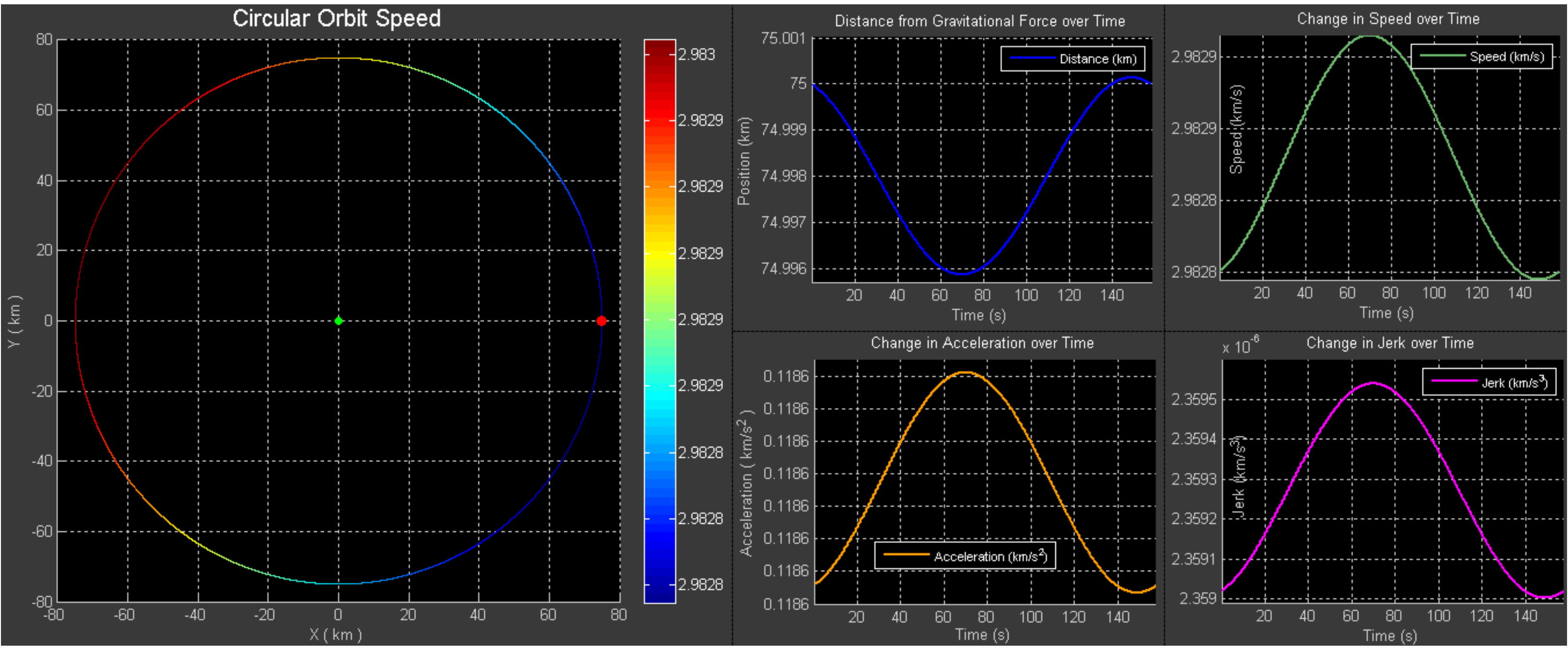
5.1) Orbital Motion Models:

We begin by examining the motion of a elliptical single orbit. The orbit motion and speed color map figure shows a green circle, representing the source of the gravitational pull, a red circle signifying the initial starting point of the satellite, and the satellite orbital path color coded based on the magnitude of the velocities at that position. The four smaller figures plot the change in the satellite's distance from the gravitational force, speed, acceleration, and jerk over time.



We observe that the satellite's change in speed, acceleration, and jerk peak when the satellite is at the smallest distance away from the gravitational force during the single orbit. The peak in the jerk lasts for the shortest period of time, while the peak in the acceleration lasts for slightly longer, and the peak in the speed spreading across the largest span of time.

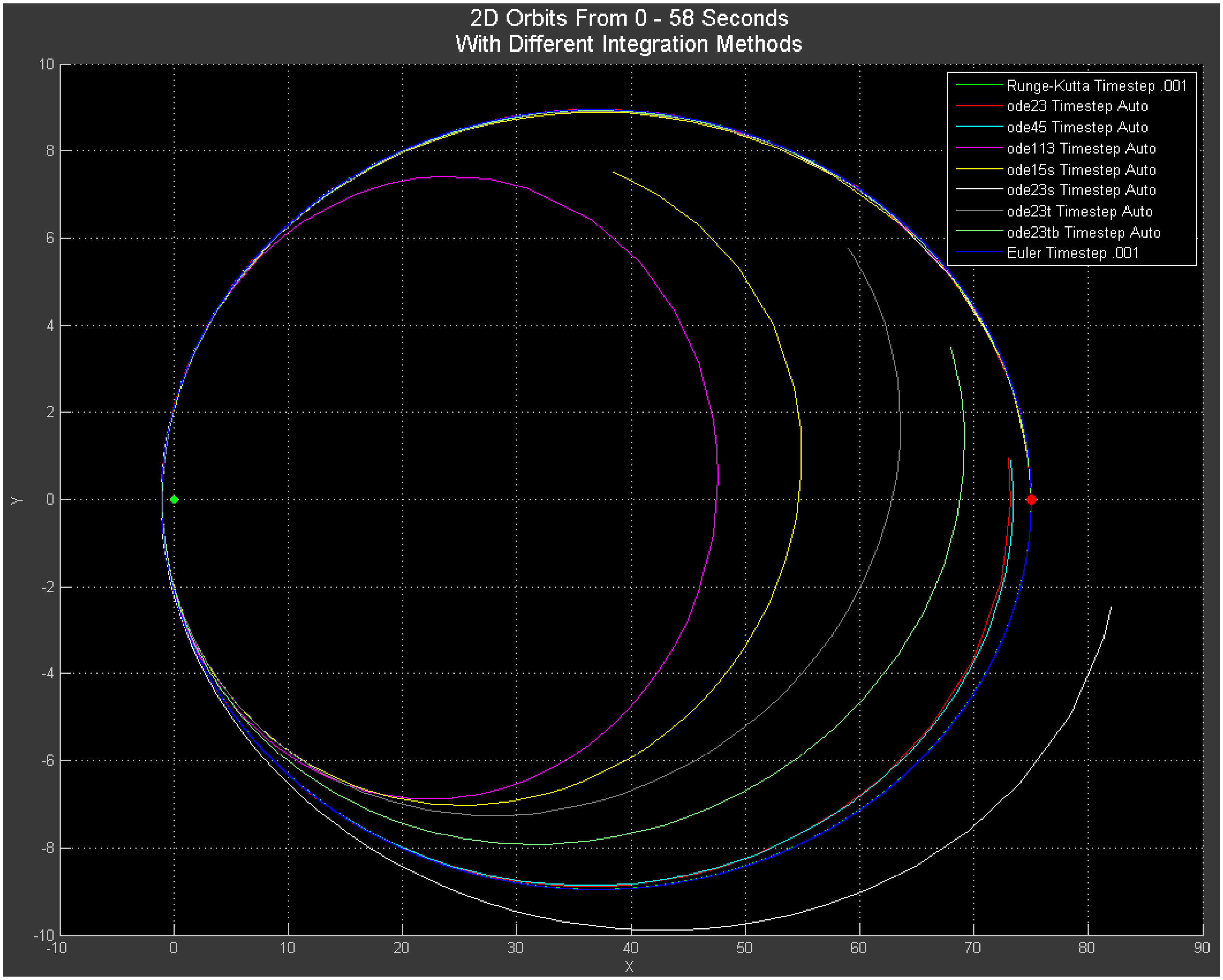
Next we examine the change in motion of a satellite in a circular orbit.



The left plot's color mapping and the smaller plot's Y axis scale have all been exaggerated to help highlight where errors are introduced. We would expect a circular orbit to maintain a constant distance away from the gravitational force and a constant speed, with no change in the magnitude of the acceleration or jerk vectors. However we observe a small error that seems to increase as the satellite approaches the perigee of its orbit and subsequently reduces as it approaches the orbit's apogee. All of the smaller plots on the right would appear to be linear functions if we were to expand the Y scale limitations.

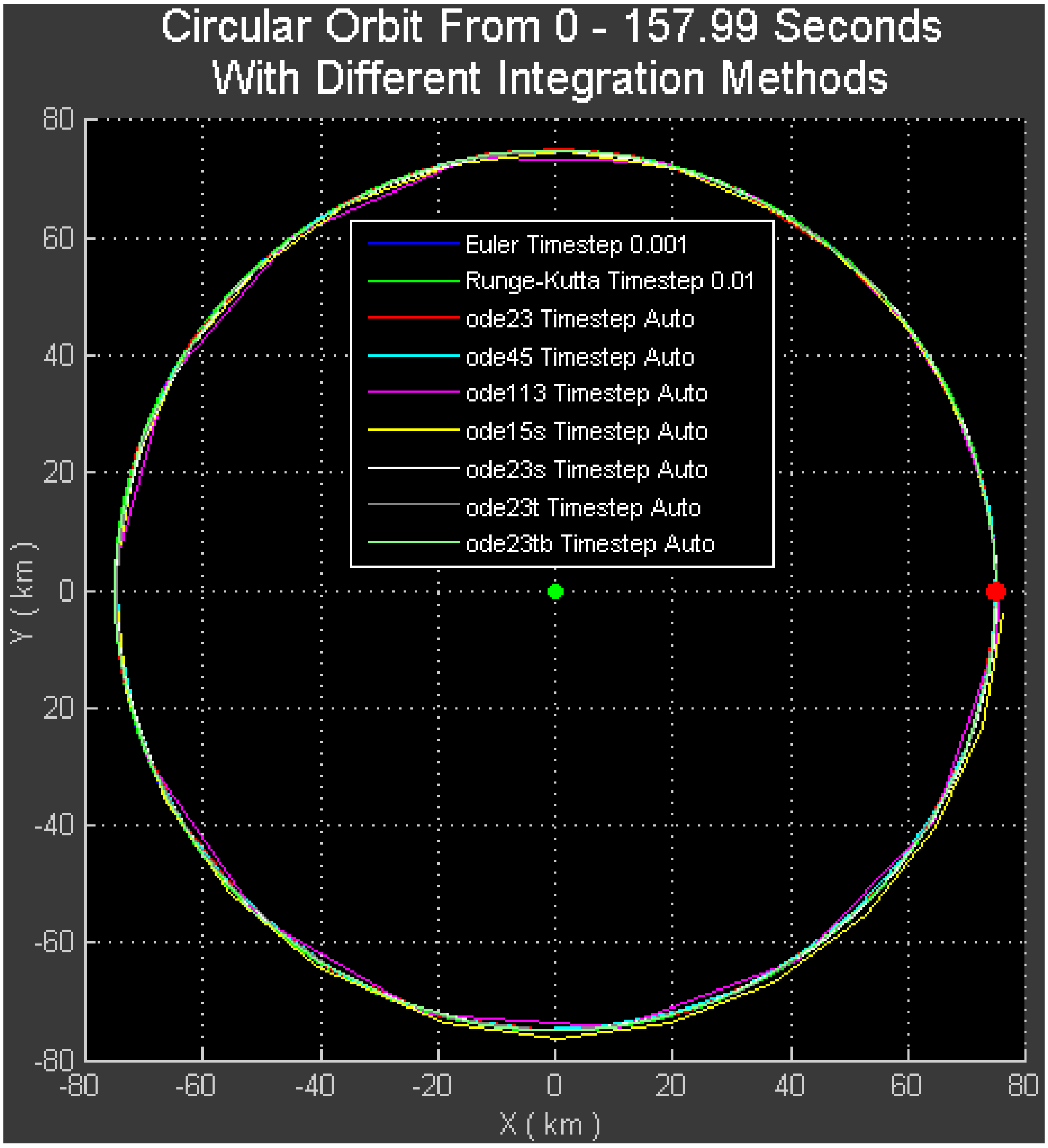
5.2) Numerical Method Orbit Modeling Experiments:

We begin by intending to explore visually how different ODE solvers calculate change in position over time. We do this by plotting a single orbit using our elliptical constants with nine different ode solvers. The green circle represents the object creating the gravitational force and the red circle represents the satellite's initial starting point.



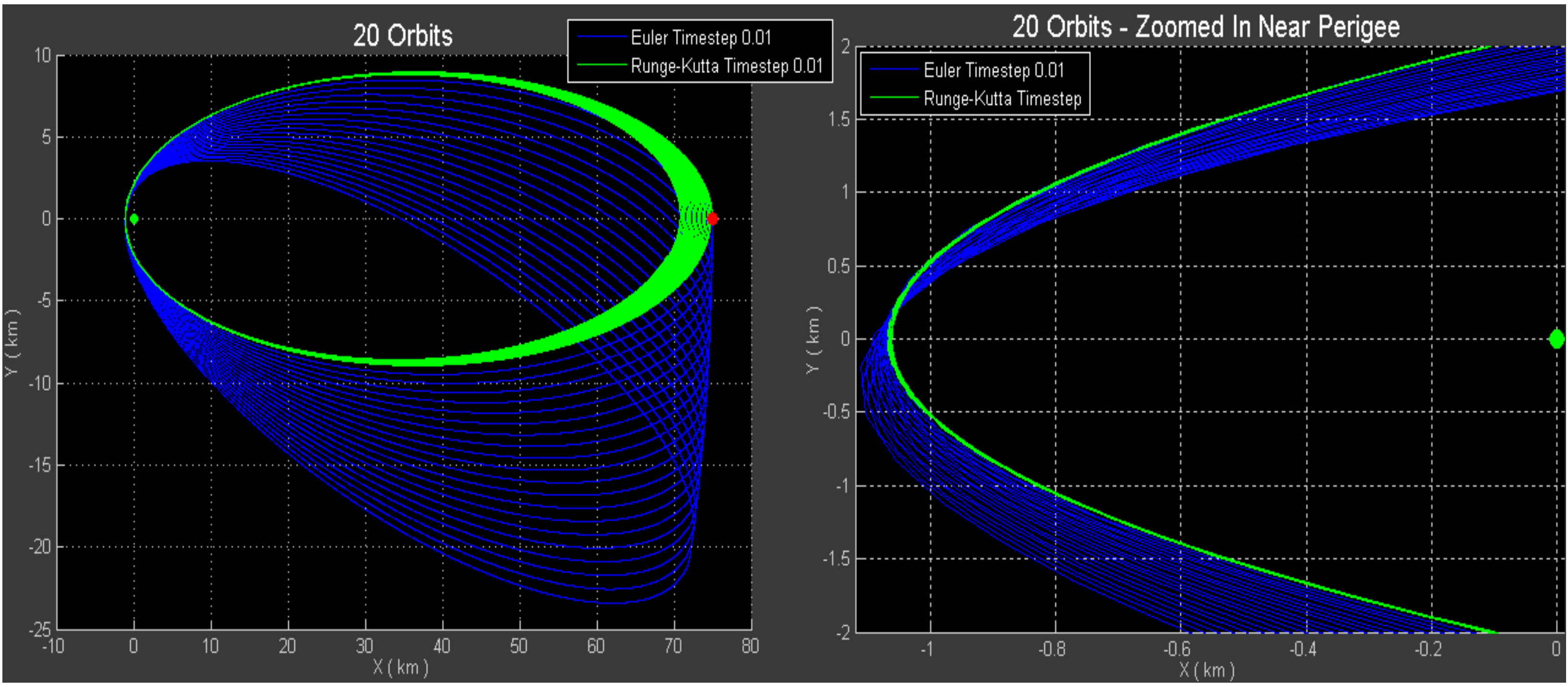
We observe that only the Euler and Runge-Kutta methods appear to calculate an orbit which returns to the initial starting point. The Matlab ode45 and ode23 functions are the closest of the Matlab solvers to match the behavior of our custom solvers with static time steps.

Now we change our initial satellite velocity to our circular orbit constants and repeat the plotting procedure.



It seems that by using a circular orbit all of the Matlab ODE solvers have shown an improvement in their accuracy. We observe that many Matlab ODE solvers create jagged sawtooth-like circles.

We now would like to visualize where the largest errors in satellite position occur over time in the elliptical orbit simulation using the Euler and Runge-Kutta methods. To accomplish this we generate two plots to explore where differences in the Euler and Runge-Kutta method positions tend to occur. In order to observe how these differences manifest over time we will plot for 20 orbit periods.



We observe that both methods seem to converge on points near the orbit's perigee and exhibit more errors as points approach the orbit's apogee. The Euler method seems to be diverging away from the gravitational force while the Runge-Kutta method diverges inward. The Euler method seems to create error in the Y plane faster than the X plane. Conversely, the Runge-Kutta Method appears to create error in the X plane faster than the Y plane. It also appears that the Euler method is exhibiting larger errors overall than the Runge-Kutta method.

5.3) Euler Method and Runge-Kutta Varying Time Step Experiments:

Here we will examine how varying static timestep intervals for Runge-Kutta and Euler's numerical methods impacts the distance the satellite is positioned from it's initial starting point over time. Also included is the number of times we are required to apply our system of equations for each method to produce a complete set of results.

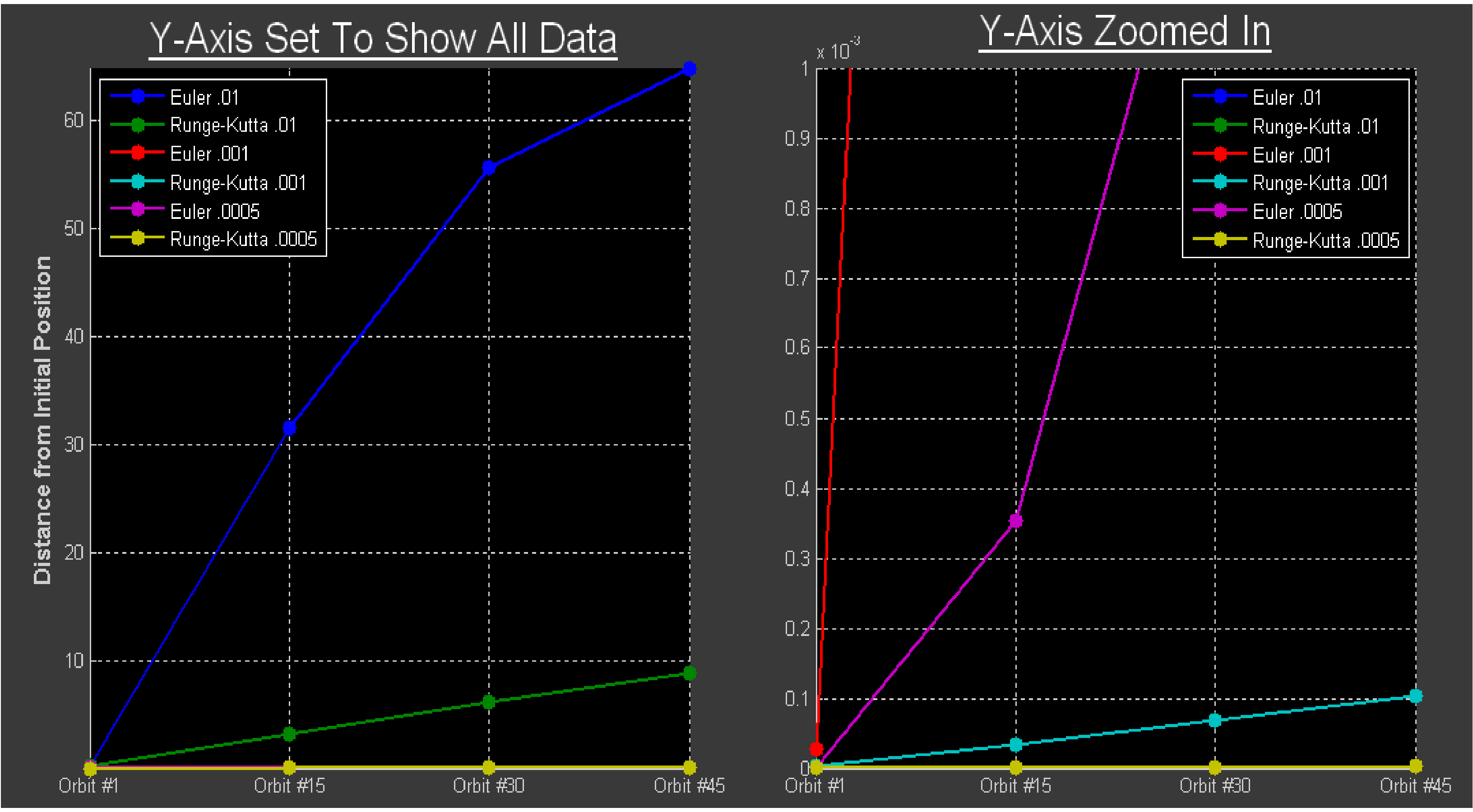
Distance Away From Initial Starting Position Over Time For An Elliptical Orbit – With Varying Timesteps Data Tables

	Euler .01	RK .01	Euler .001	RK .001	Euler .0005	RK .0005
Orbit #1	.2453	.2214	2.759 e-5	2.3059 e-6	1.8194 e-6	7.9576 e-8
Orbit #15	31.5698	3.1934	.0056	3.4506 e-5	3.5305 e-4	1.0775 e-6
Orbit #30	55.6681	6.1345	.0224	6.8944 e-5	.0014	2.1550 e-6
Orbit #45	64.8890	8.8508	.0504	1.0341 e-4	.0037	3.2327 e-6
Total Number of Calcs	285,267	1,141,068	2,852,681	11,410,724	5,705,364	22,821,456

By examining the data we observe that the Runge-Kutta method provides us with the most stable and accurate positional calculation results. The Euler method generally seems to diverges faster than Runge-Kutta. There is a case where the Euler .0005 timestep demonstrates a capability to produce slightly better accuracy in less computational steps than the Runge-Kutta method with a timestep of .001 during the first orbital period only. Due to the fourth order nature of the Runge-Kutta method, the number of applied calculation cycles is much higher than the first order Euler method. Both methods improve in accuracy greatly when decreasing the timestep to .001 from .01.

Here we plot the distance the satellite is from it's initial position at the end of the four orbital period times explored. As we expect each orbital period to end in the same place it began, any distance between the two points indicates an error. The first plot on the left show all of the data while the second plot shows only four of the methods with a smaller y scale.

Distance Away From Initial Starting Position Over Time For An Elliptical Orbit – With Varying Timesteps Figures



Visually we see that the three Runge-Kutta methods appear to introduce error linearly over the examined period of time. By modifying the Runge-Kutta method' timestep from .001 to .0005 we only observe an improvement of .1e-3 kilometers, but we double the number of calculations required to find these positional values.

Next we will examine the Matlab ODE Solvers ability to calculate accurate elliptical orbits. Here the number of calculations is a direct counter of the number of times Matlab evaluated our ODE function which calculates our motion with respect to time derivatives.

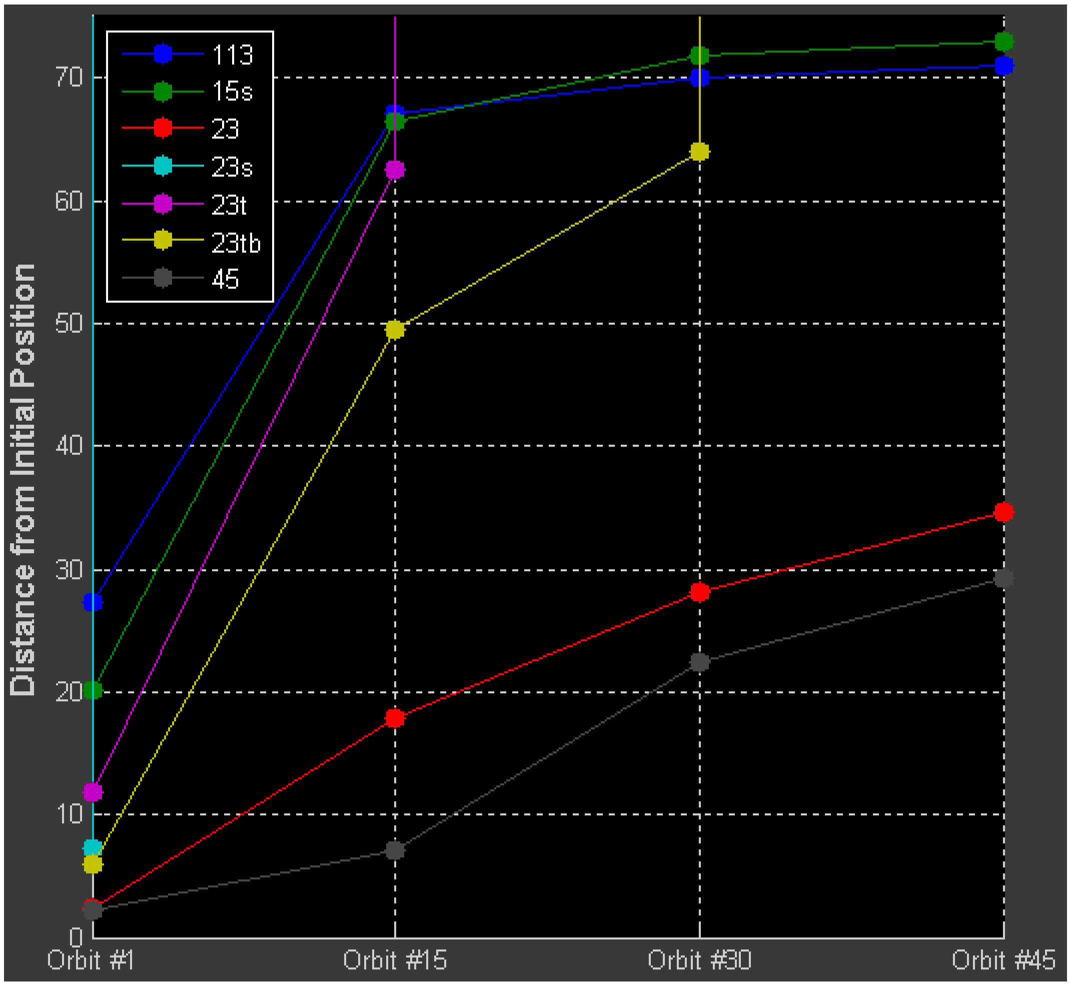
Distance Away From Initial Starting Position Over Time For An Elliptical Orbit – With Matlab ODE Solvers

	113	**15s**	**23**	**23s**	**23t**	**23tb**	**45**
Orbit #1	27.3481	20.0871	2.3411	7.2792	11.8064	5.8712	2.1427
Orbit #15	67.1660	66.4162	17.8103	Diverged Greatly	62.5238	49.5308	7.0170
Orbit #30	69.9605	71.7412	28.1006	Diverged Greatly	Diverged Greatly	63.9745	22.4052
Orbit #45	70.9371	72.9510	34.6032	Diverged Greatly	Diverged Greatly	Diverged Greatly	29.2373
Total Number of Calcs	458,214	10,648	33,562	7,062	7,004	14,431	18,415

The Matlab ODE Solvers appear to be unable to match the accuracy or stability of the Euler or Runge-Kutta methods. The 23s, 23t, and 23tb solvers diverge in a manner which causes our orbital period definition to not occur within our evaluated time span. We also note that the first order 113 solver requires the most calculations yet exhibits less accuracy compared to the other solvers.

Here we plot the distance the satellite is from it's initial starting position at the end of the four orbital period times explored.

Distance Away From Initial Starting Position Over Time For An Elliptical Orbit – With Matlab ODE Solvers Figures



The ODE 23 and 45 solvers appear to be increasing in error at a slower rate than the 15s and 113 methods. The 23s, 23t, and 23tb solvers remain comparably accurate to the other solvers until they rapidly diverge.

Now we will examine how varying static timestep intervals for Runge-Kutta and Euler's numerical methods impact the distance the satellite is positioned from it's initial starting point after each orbital period for a circular orbit. In the same manner as before, we also include the number of calculations required for each method to calculate these results.

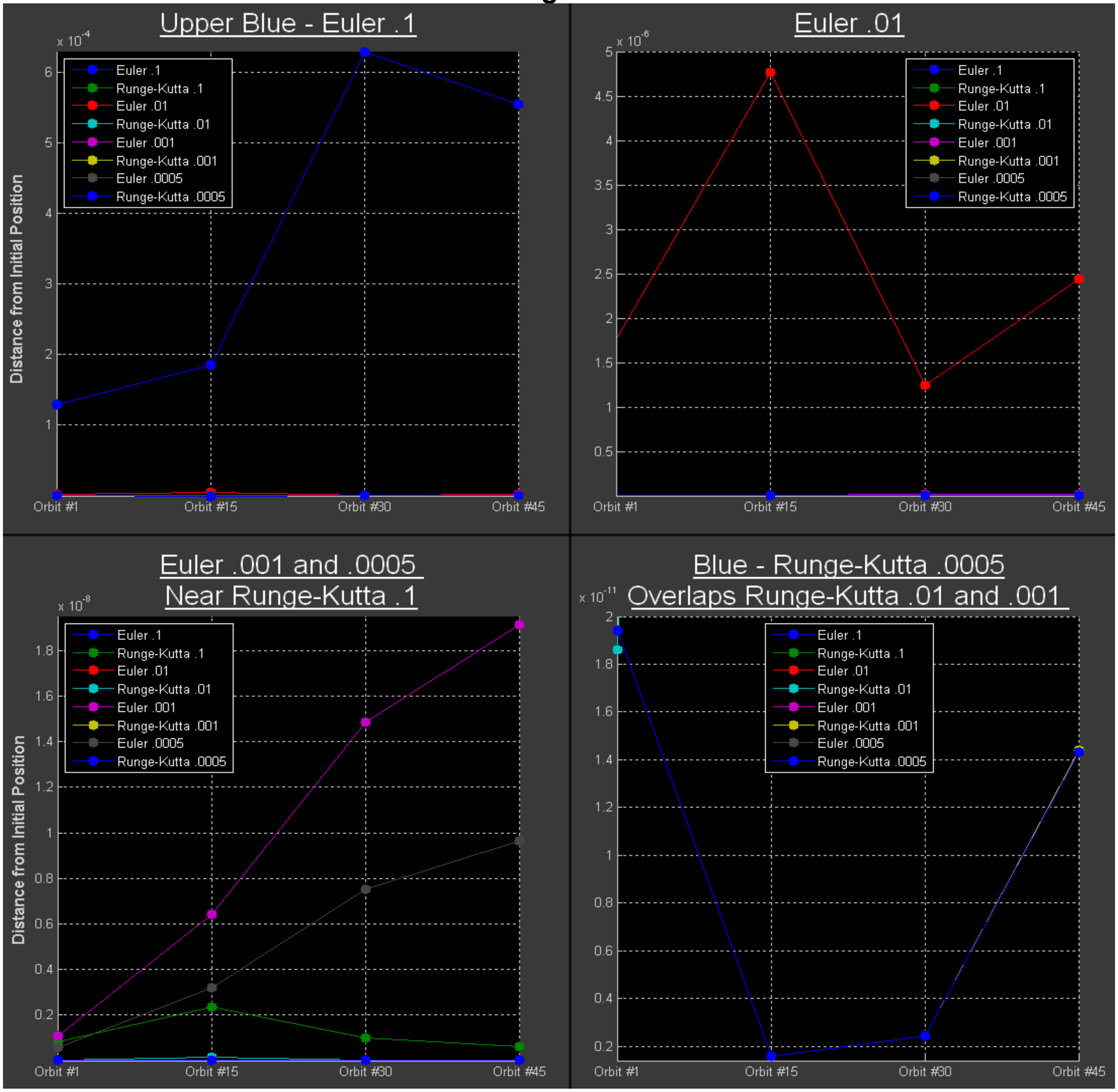
Distance Away From Initial Starting Position Over Time For A Circular Orbit – With Varying Timesteps Data Tables

	Euler .1	**RK .1**	**Euler .01**	**RK .01**	**Euler .001**	**RK .001**	**Euler .0005**	**RK .0005**
Orbit #1	1.2974 e-4	8.4451 e-10	1.7834 e-6	1.8616 e-11	1.081 e-9	1.9398 e-11	5.8068 e-10	1.9398 e-11
Orbit #6	1.8528 e-4	2.3604 e-9	4.7689 e-6	1.2764 e-10	6.4152 e-9	1.5916 e-12	3.205 e-9	1.5916 e-12
Orbit #14	6.3006 e-4	1.0062 e-9	1.2414 e-6	2.9814 e-11	1.4872 e-8	2.4158 e-12	7.5161 e-9	2.4443 e-12
Orbit #18	5.5586 e-4	5.9349 e-10	2.4435 e-6	2.1231 e-11	1.9137 e-8	1.4381 e-11	9.6451 e-9	1.4339 e-11
Total Number of Calcs	28,525	114,100	285,267	1,141,068	2,852,681	11,410,724	5,705,364	22,821,456

By examining the data we observe that the Runge-Kutta method provides us with the most stable and accurate positional calculation results. The error both of these methods create is extremely small. With large increases in the number of calculations required, Runge-Kutta is only able to bring a few more significant digits of accuracy to our solution. Another notable comparison is that the Runge-Kutta timestep of .1 is able to produce smaller errors in less computational steps than all of the Euler timesteps explored.

We now plot our data to visualize our numerical method errors.

Distance Away From Initial Starting Position Over Time
For A Circular Orbit – With Varying Timesteps
Figures



The larger two Euler timesteps explored are several orders of magnitude away from the largest Runge-Kutta .1 timestep. Smaller Runge-Kutta timesteps begin to exhibit less reduction in error, though the number of computational calculations continues to increase.

Next we will examine the Matlab ODE Solvers ability to calculate accurate orbits with circular orbits. As before we also note the number of times Matlab utilizes our function to calculate our motion derivatives with respect to time.

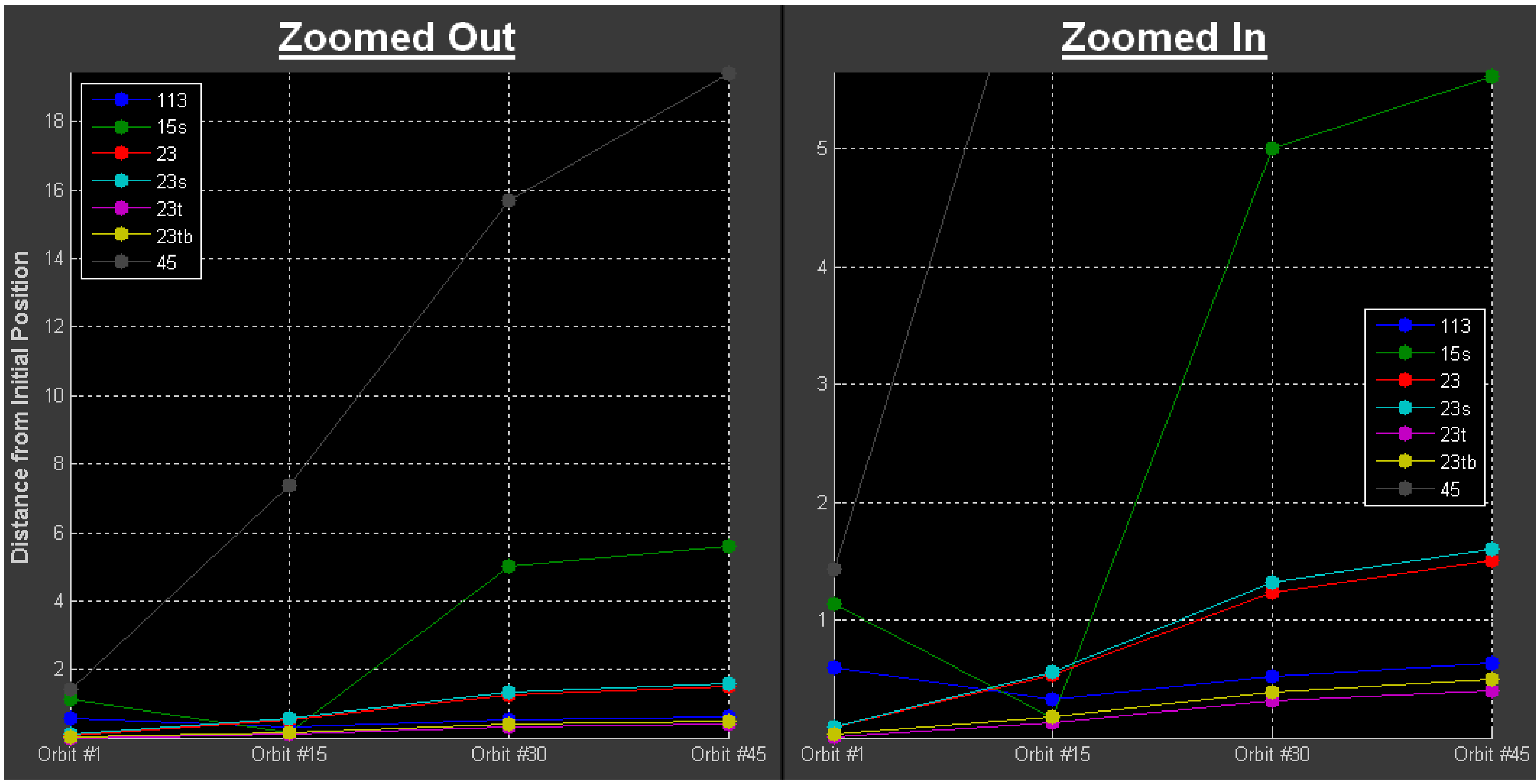
Distance Away From Initial Starting Position Over Time
For A Circular Orbit – With Matlab ODE Solvers
Data Tables

	113	**15s**	**23**	**23s**	**23t**	**23tb**	**45**
Orbit #1	.5989	1.1417	.0878	.0932	.0084	.0283	1.4326
Orbit #6	.3278	.1684	.5306	.5586	.13	.1734	7.3704
Orbit #14	.5275	5.0091	1.2340	1.3213	.3184	.3891	15.6843
Orbit #18	.6276	5.6162	1.5084	1.6043	.4007	.4962	19.4229
Total Number of Calcs	644	975	1,846	8,689	2,161	4,131	1,207

The Matlab ODE Solvers appear to be significantly more stable when presented with an orbit with less variance in speed. Ode 23t shows the lowest error in position of the Matlab ODE Solvers. As noted in the elliptical orbit, these solvers require less calculation cycles than the Euler or Runge-Kutta implementations; however the error in the Matlab ODE solvers are larger by many orders of magnitude compared to the Euler and Runge-Kutta methods.

We now plot our ODE Solver distance errors over the time period of the four orbits explored.

Distance Away From Initial Starting Position Over Time For A Circular Orbit – With Matlab ODE Solvers Figures



Ode45 diverges faster than any of the solvers, followed by the 15s solver. Ode23t, 23tb, and 113 all stay underneath an error of 1 kilometer; with 113 requiring only 644 calculations and 23t requiring 2,161.

6. Conclusions

Our elliptical orbit simulation presented more variance in the motion of an orbiting satellite as opposed to our circular simulation. Numerical error was present in both of our simulations, but it seems to be easier to reduce error in more circular orbits. This may be related to reduced amount of change in motion of a circular orbit compared to an elliptical one. We have observed that the orbital apogee has been where our positional errors appear to be the most visually obvious. These errors also become more obvious if we allow the simulation to run for a longer time span. Reducing the time step in our Runge-Kutta and Euler numerical methods has shown itself to be a reasonable way to achieve more significant digits of positional accuracy; but it can come at a large computational cost. When creating visual simulations over short time spans we observe that one has several numerical methods to possibly experiment with. Allowing simulations to run for longer periods of time may highlight the error present in the numerical method calculation. None of the examined numerical methods were able to present simulations that ran over time with perfect numeric accuracy. Achieving perfect numeric accuracy with numerical methods may not be feasible, one may simply need to define the application in which one intends to use the system of equations and set a tolerance that is acceptable for that application. Additional research could be done by exploring predictor-corrector methods, or perhaps finding a new approach to the problem which does not involve numerical methods. At this point in time the experimenter has yet to research any other viable avenues to achieve improved accuracy – aside from simply reducing timesteps at the cost of significantly more computational cycles and beginning to experiment with a Euler Trapezoidal Predictor Corrector Method.