AM 205 Final Project
The N-Body Problem

Leah Birch
Elizabeth Finn
Karen Yu

December 14, 2012

Abstract

The N-Body Problem can be solved using a variety of numeric integrators. Newton's Law of Universal Gravitation was transformed into a second order ODE, and its most efficient numeric integrator was determined by calculating the conservation of energy of the system. The Leapfrog Integrators were determined to be the most efficient, so they were then applied to two interesting aspects of the N-Body Problem: the Kozai Mechanism and the Tree Code for large N.

1 Physical Principles

We must first determine our equations of motion. From Newton's Law of Universal Gravitation, we know that the force F between two objects is

$$F = G\frac{m_1 m_2}{r^2}$$

From Newton's Second Law, we also know that

$$F = ma$$

From the perspective of object 1, the force acting on it is

$$F_1 = G\frac{m_1 m_2}{r^2} = m_1 a_1$$

Its own mass cancels out, and its acceleration is

$$a_1 = G\frac{m_2}{r^2}$$

Relating acceleration to position

$$\frac{d^2\vec{x}}{dt^2} = \vec{a}$$

1.1 Generalizing to N bodies

With these forces in mind, Newton's Law of Universal Gravitation can be extended to account for more than two bodies, where every body causes a force on every other body. Thus, the force on the i-th body is:

$$\vec{F_i} = \sum_{i \neq k} \frac{G m_i m_k}{||\vec{r_k} - \vec{r_i}||^3}(\vec{r_k} - \vec{r_i}),$$

where r is the vector of the positions.
Again, mass will cancel out, leaving acceleration, so a second order ODE is formed.

$$\vec{a_i} = \sum_{i \neq k} \frac{G m_k}{||\vec{r_k} - \vec{r_i}||^3}(\vec{r_k} - \vec{r_i}) \tag{1}$$

2 Numerical integration methods

Acceleration is a function of position and time.

$$\vec{a} = f(\vec{x}(t))$$

The relationship between position and acceleration is a second order ordinary differential equation. In order to solve this numerically, we can decompose this into two first order ODEs by introducing a new variable, velocity.

$$\frac{d\vec{v}}{dt} = f(\vec{x}(t)) \tag{2}$$

$$\frac{d\vec{x}}{dt} = \vec{v} \tag{3}$$

There are several methods for numerical integration that are examined in this problem.

2.1 Forward Euler

The Forward Euler method is an explicit method meaning that y_{k+1} is explicitly defined in terms of y_k .

$$y_{k+1} = y_k + hf(t_k, y_k) \tag{4}$$

In the implementation of this method, the acceleration was calculated using the positions from the previous time step. That acceleration was then multiplied by the time step, dt, which corresponds to the h in Equation 4 and added to the velocity from the previous time step. This produced the velocities for the current time step. The calculation for current position was implemented in a similar way, however the velocities, not the accelerations, were used in combination with the previous position to update.

2.2 Backward Euler

Backward Euler is an implicit method, meaning it requires use of the next time step in order to calculate the next time step.

$$y_{k+1} = y_k + hf(t_{k+1}, y_{k+1}) \tag{5}$$

Implicit methods are more complicated to implement than explicit methods like Forward Euler. They are also more computationally expensive because they require additional calculations at each time step. In order to implement this method, first the position at the next time step was estimated using the current position and velocity. Using that estimated position the acceleration was calculated and then added to the current velocity to calculate the velocity for the next time step. Once the velocity at the next time step was calculated, it was used to calculate the position at the next time step.

2.3 DKD Leapfrog

The leapfrog method comes in two versions, "Drift-Kick-Drift" and "Kick-Drift-Kick". The first method discussed is the "Drift-Kick-Drift" version. It achieves second order accuracy by taking half-steps when computing the position of the particle. The algorithm is as follows:

$$x_{n+1/2} = x_n + v_n \frac{\Delta t}{2} \tag{6}$$

$$v_{n+1} = v_n + f(x_{n+1/2})\Delta t \tag{7}$$

$$x_{n+1} = x_{n+1/2} + v_{n+1} \frac{\Delta t}{2} \tag{8}$$

The leapfrog integrator is symplectic, meaning it preseves the Hamiltonian. It does so by splitting the Hamiltonian into kinetic and potential components (drift and kick).

2.4 KDK Leapfrog

The other leapfrog method is the "Kick-Drift-Kick" version. It takes half-steps when computing the velocity of the particle and is second order accurate. The algorithm is:

$$v_{n+1/2} = v_n + f(x_n)\frac{\Delta t}{2} \tag{9}$$

$$x_{n+1} = x_n + v_{n+1/2}\Delta t \tag{10}$$

$$v_{n+1} = v_{n+1/2} + f(x_{n+1})\frac{\Delta t}{2} \tag{11}$$

Again, this integrator is symplectic.

2.5 Fourth-Order Runge-Kutta

Runge-Kutta methods [5] are a family of numerical ordinary differential equation solvers, with Fourth-Order being one of the most common. The Runge-Kutta solves first order differential equations of the form $\dot{y} = f(t, y)$ with $y(t_0) = y_0$. Thus, in general the solution is

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k + 3 + k_4) \tag{12}$$

$$t_{n+1} = t_n + h, \tag{13}$$

where h is the step size and

$$k_1 = hf(t_n, y_n) \tag{14}$$

$$k_2 = hf(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \tag{15}$$

$$k_3 = hf(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2) \tag{16}$$

$$k_4 = hf(t_n + h, y_n + k_3) \tag{17}$$

$$\tag{18}$$

Runge-Kutta is used to solve first order differential equations, so N-Body Problem must be solved a little differently. The k_1 is calculated using our function for the acceleration, but we must note that this k_1 is for the velocity, not position. To calculate k_2, the change in position is necessary, so k_1 must be converted in such a way that it is a change in position. Then, the same must be done to k_2 so that k_3 can be calculated and so forth. However, in all of these conversion, the original k's must be saved so that Equation 12 can be calculated. Then the new position is founding using the new velocity and the time step h.

2.6 Built-in solvers (ode45)

In MATLAB, the built-in solver ODE45 solves an ordinary differential equation using Fourth Order Runga Kutta with an adaptive time step. Again, since it is used to solve first order ordinary differntial equations, the N-Body problem must be split into 2 first order differential equations as mentioned in Equations 2 and 3. We expect this method to be inaccurate because of the adaptive time step. It was suggested by Springle in Cosmological Numerical Simulations [4] that the adaptive time step is a detriment to N Body solvers.

3 Application to Solar System

We applied our N-Body ordinary differential equation to a small model of the solar system of 5 bodies, including the Sun, Mercury, Venus, Earth, and Mars. We used their actual masses and the initial conditions are spaced appropriately for their actual orbits. We also took their velocities into account to a certain extent, so the orbits formed are elliptical and should remain so as time progress. Of course, not all integrators cause the planets to maintain their orbits. Some reveal that we are doomed to spiral into the sun, while others show that we will shoot off away from the Sun into the great unknown.

Besides looking at the orbit trajectories, we need to determine the error of each method. We examined the change in energy over time to evaluate the accuracy of each integrator. The sum of the kinetic and gravitational potential energy for the system should remain constant over time, meaning that the relative error of the energy $\frac{\Delta E}{|E(t)|}$ for each time step should be rather small and constant. For our purposes, we took the absolute value of the relative energy error so that any conservation issues will result in an increase in the relative energy error. We found that our integration techniques have error matching what Springle described in his exploration of Cosmological Numerical Simulations [4].

3.1 Forward Euler

For Forward Euler, elliptical orbits are formed with 10,000 time steps. This method proved to be unstable for a large time span causing the particles orbits to grow over time and spiral outwards. The change of energy over time confirms that energy is not conserved using Forward Euler as Figure 1 illustrates, the error is increasing over time.

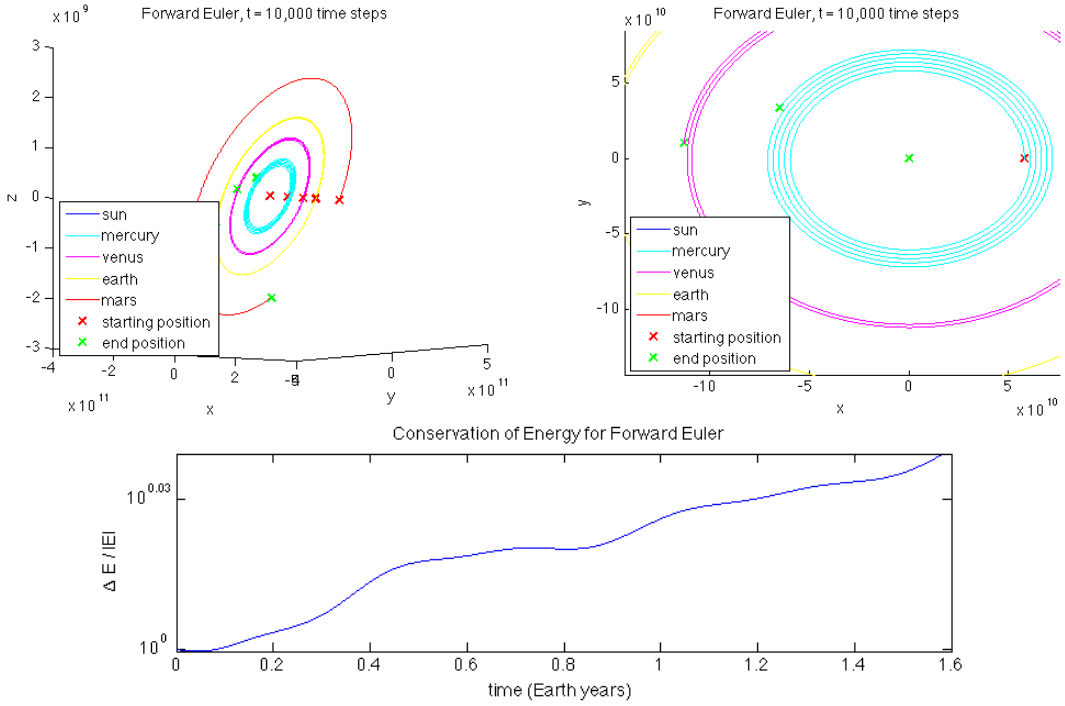

Figure 1: Forward Euler

3.2 Backward Euler

The elliptical orbits for Backward Euler were formed with 9,500 time steps. This method, like Forward Euler, proved to be unstable for a large time span. While Forward Eulers instability caused the orbits to grow, Backward Euler caused the orbits to shrink over time and spiral inwards. If allowed to run for a large enough time span, this caused the planets in the model to eventually get so close to the sun that they were repelled into space. Figure?? shows that the energy is changing over time further confirming that this method is unstable.

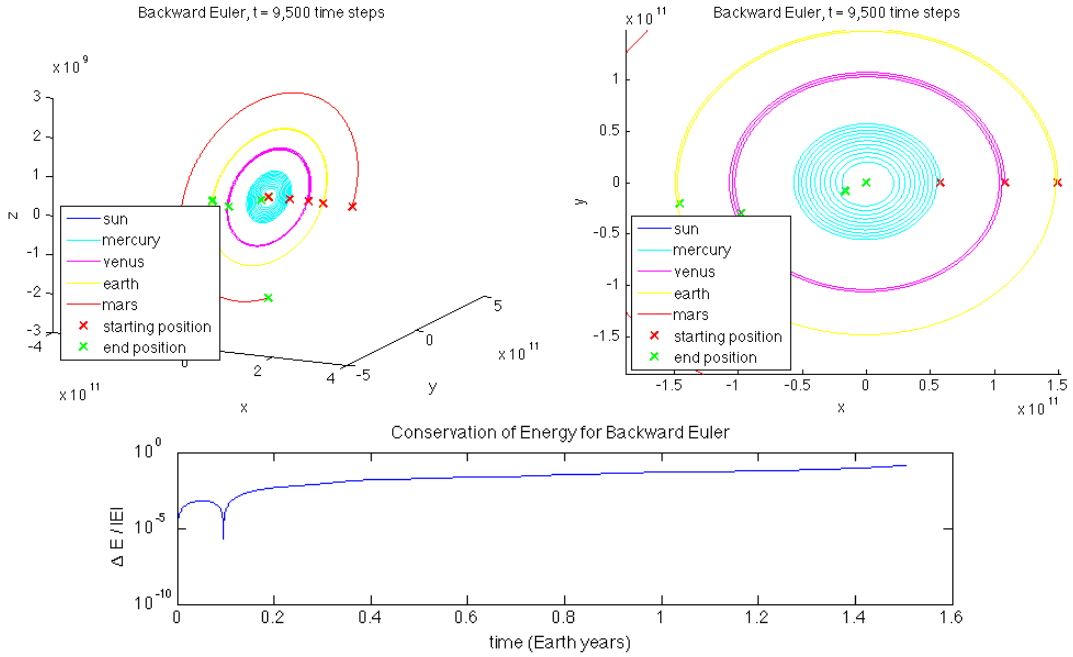

Figure 2: Backward Euler

3.3 KDK Leapfrog

For the leapfrog integrator, we ran the simulation for 100,000 time steps. The first 10,000 time steps are plotted on Figure 3. We see that circular orbits are formed.

Figure 3 shows the relative change in energy for the 5 error for the 5-body system. We see that it osciallates between zero and 7e-5, but remains steady over time, validating the symplectic behaviour of our method.

3.4 DkD Leapfrog

The for "Drift-Kick-Drift" mechanism, the results are essentially identical to the "Kick-Drift-Kick". We see the same orbits formed and the same symplectic behavior over time, as shown in Figure 4.

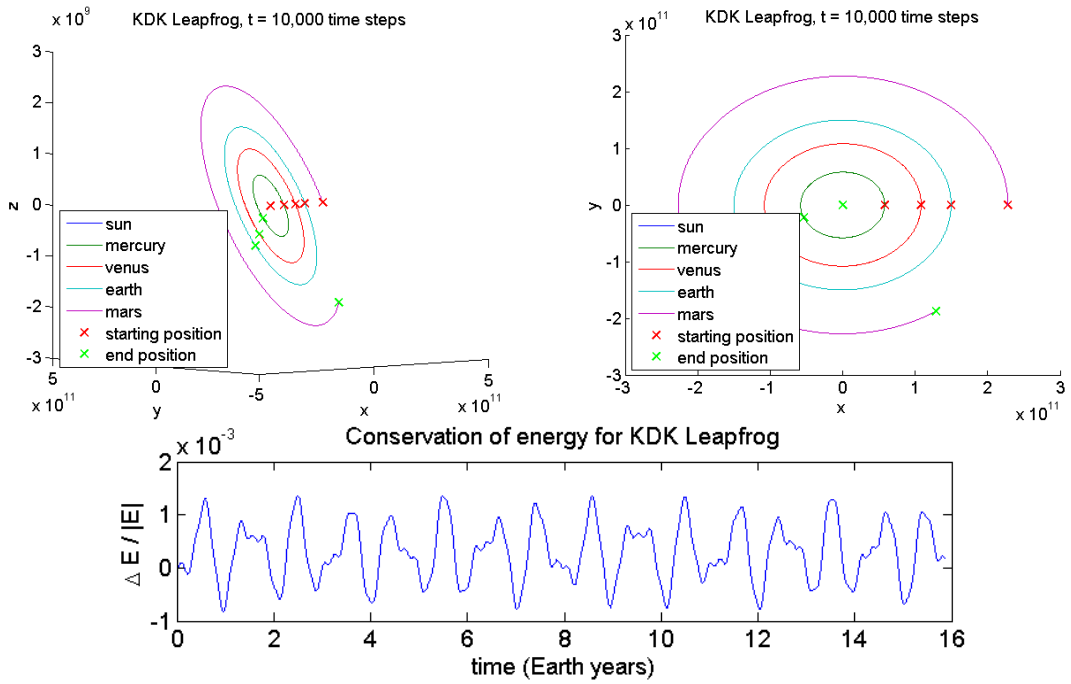

Figure 3: KDK

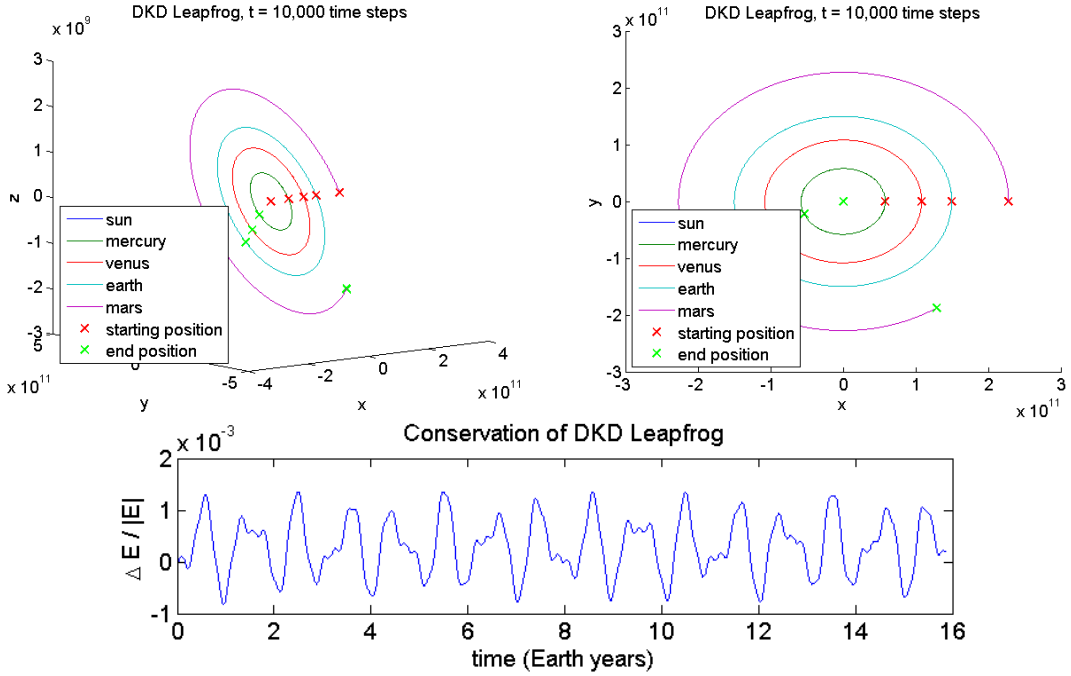

Figure 4: DKD

3.5 Fourth Order Runge Kutta

For the Fourth Order Runge Kutta, elliptical orbits are formed with 10,000 time steps, but over time the planets spiral inward. The change of energy over time confirms that energy is not conserved using Runge Kutta as Figure 5 illustrates, and it does not take long to

discover that the error is increasing.

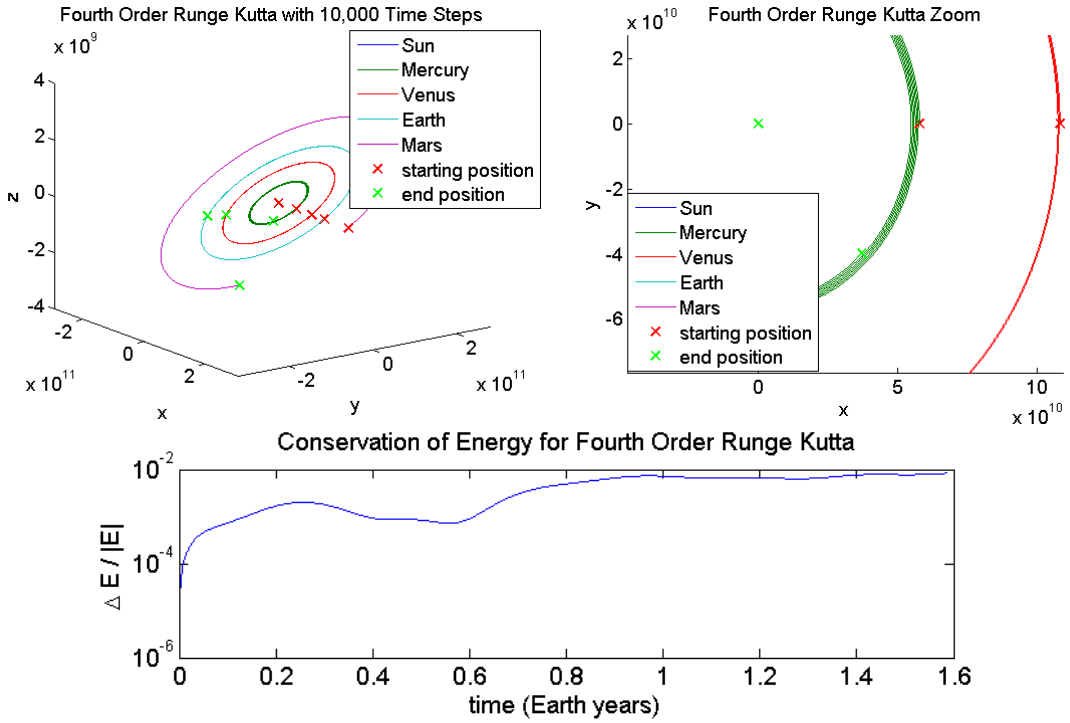

Figure 5: ODE45

3.6 ODE45

In order to see ODE45 degenerate, it was run for a time span of 0 to 1,000,000,000 with variable time steps, which turns out to be around 40 years. The orbits formed are elliptical, but they are spirlaing inward, like was seen in the Fourth Order Runge Kutta. Therefore, since the orbits are not maintained, the relative change in energy for the error increases as time goes on, which can be seen in Figure 6. This particular case was observed over a longer time because of the oscillations present. The long time span allowed us to see that the oscillations dissipated as the error got larger. It is likely that these oscillations occur because of the variable time steps.

3.7 The Most Stable Integrator

The Leapfrog Integrators are the most accurate for solving our second order ODE's of the N-Body Problem because they keep the planets in the most stable orbit, losing the least amount of energy. Using the KDK Leapfrog, we can examine special cases of N-Body Problem. For instance, in this paper we have looked at the Kozai Mechanism and how to solve a system of large N.

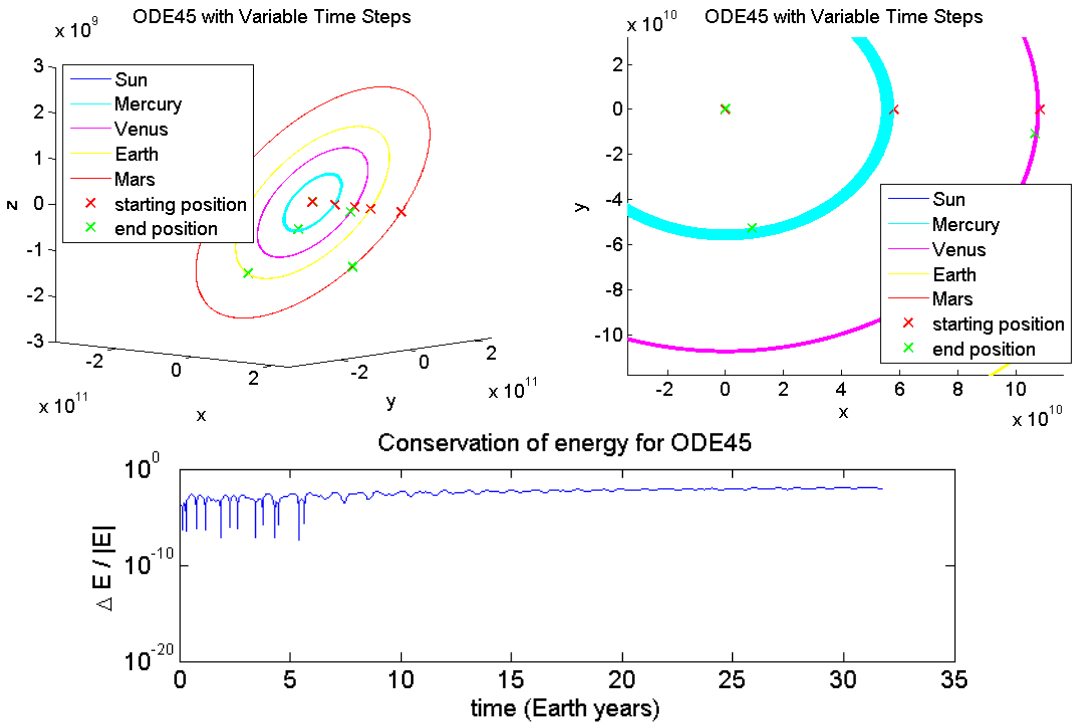

Figure 6: ODE45

4 Application to Kozai Mechanism

Using our most stable method, the Leapfrog Integrator, we can examine how certain initial conditions can lead to the Kozai Mechanism[3], which reveals a periodic eccentricity of an orbit. To test our integrator for Kozai conditions, we simulated a 3 Body system, which consists of a Sun and 2 small planets that begin very close to each other.

The initial conditions needed to implement the Kozai Mechanism were found in *The Role of Kozai Cycles in Near-Earth Binary Asteroids* by Fang and Margot[2]. However, we could not use these conditions exactly as they used a different set of differential equations. We had to transform their initial conditions. The initial positions merely had to be transformed from Astronomical Units to meters, and the two small planets were positioned using their center of mass. The calculation of initial velocity was more complicated, and needed a few intrinsic properties of ellipses. Figure 7 shows an ellipse with certain parts of its geometry marked. As a rule, adding the distance from the distance from F_1 to any point on the ellipse X to the distance from F_2 to that same point will always be equivalent to $2a$ where a is the semi-major axis length. By selecting X such that it is concurrent with point D, a right triangle is formed with legs of distance ea (where e is eccentricity) and b with hypotenuse of length a. Knowing $a = 0.82$AU and $e = 0.3$AU as the initial conditions the calculation for b is trivial. Using a and b we calculated the circumference of the initial ellipse. From there it was necessary to do a unit conversion from AU to m/s using the number of degrees per day the two planets traveled around the sun which was given in the paper. Once the magnitude of the velocity was calculated, the direction and components were calculated using the given initial inclination of 40°.

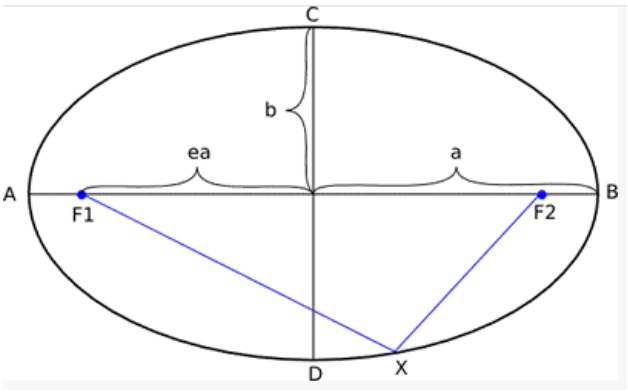

Figure 7: Elliptical Geometry

Though the orbits seem close together, this is only due to the scale of the problem. Closer inspection would reveal a separation of the orbit, which shows the eccentricity is changing. The eccentricity for each planet was calculated for each orbit using the semi major and minor axes,

$$e = \frac{r_{major} - r_{minor}}{r_{major} + r_{minor}}.$$

The eccentricity oscillates for both small masses as shown on the right of Figure 8. Furthermore, the values of the eccentricity match the paper from which we found the initial conditions. This result instills further confidence in our N-Body solver, since we replicated the results from a paper, which used a different set of differential equations.

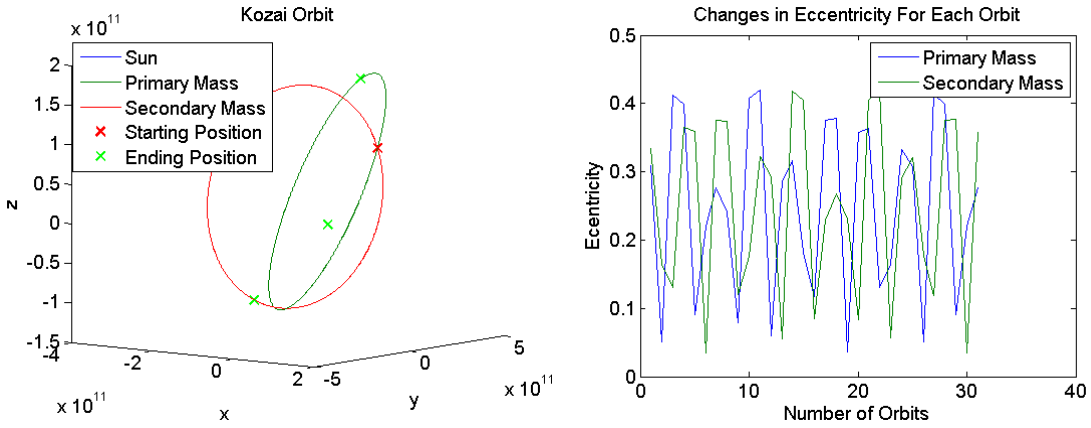

Figure 8: Kozai Orbit and Eccentricity

5 Methods for larger simulations

Our previous applications were for a 5-body and 3-body system, respectively. At this scale, it is not computationally intensive to compute all the pairwise interactions between all the particles. However, if we wanted to perform simulations for larger systems, this could become computationally expensive and impractical. There have been several methods developed to make approximations that speed up the computation. Two commonly used methods are the

tree-code method and the particle-mesh method. Due to its complexity, we will not discuss the paticle-mesh method in this paper.

5.1 Tree codes

The concept of the tree-code was first implemented by Barnes & Hut [1] and is also called the Barnes-Hut algorithm [6]. The idea is to recursively divide the space occupied by the particles into quadrants until each quadrant contains only one particle. This is accomplished through a quad-tree structure in 2 dimensions, or an oct-tree structure in 3 dimensions. To construct the tree, begin with a quadrant the size of the entire space occupied by all the particles. We insert each particle into the tree one by one. First we check to see if the quadrant that we want to insert the particle into already contains a particle. It it does not, the particle is inserted into that quadrant and we move on to the next particle. If it does already contain a particle, we replace the particle with the center of mass and combined mass of the existing particle and the new particle, and then attempt to insert the particle into the appropriate child node of that quadrant. The process is repeated until an empty quadrant is found. If the quadrant contains an existing particle but no children, we first replace the existing particle with the center of mass information for the existing and new particle, and then recursively insert both the new and existing particle into the appropriate quadrants.
Once we have the tree structure, we can compute the gravitational interactions between the particles. For each particle, start at the root of the tree, and compute a quantity called theta, that represents the ratio between the size of the current quadrant to the distance between the current star and the center of mass of the other quadrant.

$$\theta = \frac{D}{r}$$

where D is the length of one side of the current quadrant and r is the distance between the current start and the center of mass of the quadrant we are calculating pairwise interactions with. If θ is less than a prescribed value, usually 0.5, then we can go ahead with the force calculation, even if quadrant that we are looking is an aggregate of many stars.
We have attempted an implementation of the tree code. However, we were unable to write a tree code that was computationally faster than the brute-force computation. The creation of the tree turned out to be computationally expensive, even if we could reduce the number of pairwise interactions that needed to be calculated.

For our implementation of the tree code, we used the Object Oriented Programming ability of Matlab to create a "quadrant" class that contained the location of the lower left corner of the quadrant, and the length of the quadrant. Another class classed "quadTree" was created that contained information about the largest quadrant in the tree, the mass and location of the star stored in that quadrant, and pointers to 4 other quadtrees representing the northwest, northeast, southwest, and southeast quadrants of the larger quadrant. Theta was set equal to 0.5.
In Figure 9, the top two graphs show the n-body simulation for 500 sun-sized stars randomly located. Their initial velocities are all zero. The simulation is only run for 10 time steps

because it is computationally intensive.

As we can see, the tree code reproduces the results of the brute force calculations. Note

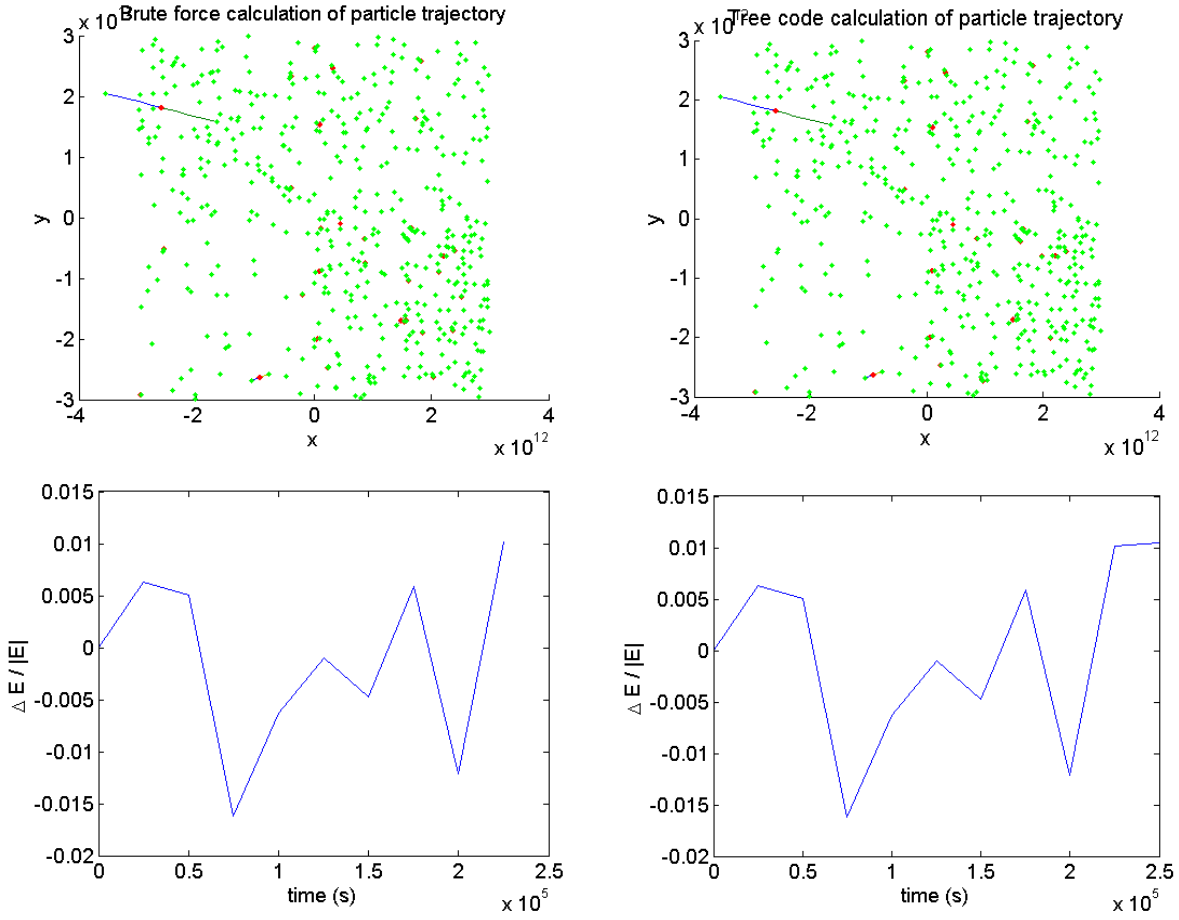

Figure 9: N-body simulation for 500 stars for 10 time steps

that for the tree code, an extra time step was accidentally plotted for the error plot, but this was not redone because of the computation time required. However, this computation only required 2.8 seconds for the brute force method, but 389 seconds for the tree code, making it impractical to use.

Future work on this project would involve refining the tree code. Currently, quadrants are represented with their own data type, but converting these to a matrix field within the quadTree data type might speed up computation. A faster way of constructing trees could perhaps be implemented by first sorting the array containing the mass and location of stars.

6 Conclusion

Using an analysis of the energy conservation, we determined that the Leapfrog Integrators were the most efficient numeric integrators for Newton's Law of Universal Gravitation with generalization to N bodies. We further verified their credibility by replicating results from a paper on the Kozai Mechanism that solved a different set of differential equations. We then attempted to implement a tree code to make implementation of the N body solver more

practical for large N. We believe it could be useful to remove the tree code implementation from MATLAB to another programming language to make the process faster, since MATLAB does not have a tree class.

7 Guide to the MATLAB Code

1. @quadrant= Is called by the Tree Method (Contains quadrant.m)
2. @quadTree= Is called by the Tree Method (Contains quadTree.m and center_of_mass.m)
3. accel= Computes the acceleration according to Newton's Law of Universal Gravitation in 2 dimensions
4. accel_3d= Computes the acceleration according to Newton's Law of Universal Gravitation in 3 dimensions
5. center_of_mass= Computes the center of mass
6. energy= Computes the energy of the system at a time t
7. EulerComparison=Compares Forward and Backward Euler for 5 bodies
8. galaxy_simulator=Solves the N Body problem for large N in 2 dimenions. It compares Leapfrog and the Tree Method.
9. integrator_comparision=The code for the Leapfrog Methods with 5 bodies
10. Kozai= Computes the Kozai orbit and eccentricity for 3 bodies
11. ode_accel=The function needed for ode45 in the runge kutta file
12. runge_kutta=The code for Fourth Order Runge Kutta and ODE 45 with 5 bodies

References

[1] J. Barnes and P. Hut. A Hierarchical O(N log N) Force Calculation Algorithm. *Nature*, 324:446–449, 1986.

[2] J. Fang and J. Margot. The Role of Kozai Cycles in Near-Earth Binary Asteroids. *The Astronomical Journal*, pages 143–159, 2012.

[3] Y. Kozai. Secular Perturbations of Asteroids with High Inclination and Eccentricity. *The Astronomical Journal*, 67(9):591–598, 1962.

[4] V. Springle. Cosmological Numerical Simulations, July 2006.

[5] G. Strang. *Computational Sciene and Engineering*. Wellesley-Cambridge Press, 2007.

[6] T. Ventimiglia and K. Wayne. The Barnes-Hut Algorithm, 2003.