Planetary Orbit Integration

Jason Phelan, Max Paik, Orion Forowycz, Valeriia Rohoza

Physics 352: Final Project - March 2021

1 Github Link

https://github.com/maxpaik16/352FinalProject

2 Introduction

While a single planet orbiting a fixed, massive sun is a stable system, adding additional planets to the system will cause the orbits to proceed in unexpected ways due to the gravitational interactions among the planets themselves. Our project was to numerically model the evolution and stability of planetary orbits over many thousands of years.

3 Planetary System

We first consider a single planet of mass m_1 orbiting a fixed sun of mass M_S. The gravitational force between the two objects is therefore

$$F_{G,1} = -\frac{GM_s m_1}{r_1^2}\hat{\mathbf{r}}_\mathbf{1},$$

where G is the gravitational coefficient and $\mathbf{r}$ is the vector point from the sun to the planet. Assuming that the sun is stationary, it follows that the acceleration of the planet is given by

$$\mathbf{a_1} = -\frac{GM_s}{r_1^2}\hat{\mathbf{r}}_\mathbf{1}.$$

As we introduce additional planets into the system, we must consider the gravitational force between each pair of planets. So, with only one additional planet with index $i = 2$, the total gravitational force on the first planet is:

$$F_{G,1} = -\frac{GM_s m_1}{r_1^2}\hat{\mathbf{r}}_\mathbf{1} + \frac{Gm_1 m_2}{r_{1,2}^2}\hat{\mathbf{r}}_\mathbf{1,2},$$

where $\mathbf{r_{1,2}}$ is the vector pointing from planet 1 to planet 2. Then, to obtain the force on planet two, we simply swap the indices. From this pattern, it follows that the gravitational force on the i^{th} planet (with, say, N total planets) is given by:

$$F_{G,i} = -\frac{GM_s m_i}{r_i^2}\mathbf{\hat{r}_i} + \sum_{j\neq i} \frac{Gm_i m_j}{r_{i,j}^2}\mathbf{\hat{r}_{i,j}}$$

Dividing $F_{G,i}$ by m_i gives us the acceleration, and we use these equations to model our solar system.

4 Symplectic Integrator

The instability in planetary systems manifests over many thousands of years, due to the relative weakness of the gravitational force. Thus, we must choose a method of solving the system that is robust over long periods of time. Symplectic integrators are a particularly useful class of integrators for this problem. For a Hamiltonian system with coordinates p and q, a symplectic integrator is one that conserves the Hamiltonian as well as the volume in phase $(p-q)$ space. Importantly, for planetary orbits, the Hamiltonian of the system is given by $\mathcal{H} = T + U$, where T and U are the kinetic and potential energies, respectively, and therefore, the energy of the system should be conserved. [1]

4.1 Velocity Verlet Method

Initially to solve for planetary orbits, we implemented the velocity Verlet method, which is given by the equations:

$$\begin{gathered} a_n = F(x_n) \\ v_{n+1/2} = v_{n-1/2} + (a_n)\Delta t \\ x_{n+1} = x_n + (v_{n+1/2})\Delta t \\ v_{n+1} = v_{n+1/2} + \frac{1}{2}(a_n)\Delta t \end{gathered}$$

To evaluate the $n+1$ position element, the velocity Verlet method uses the $n+1/2$ velocity element, giving it a 'leapfrog' nature. Despite its similarities to the first order Euler method, the velocity Verlet method is a second order method of integration. [1]

We tested the velocity Verlet method by simulating a single planet in a circular orbit about the fixed sun, and the Verlet integrator performed as expected. For small enough time steps, as in Figure 1, the energy-time trace is sinusoidal with a fixed offset. Further, this offset did not drift with time, so the Verlet integrator conserved energy though only on average. Comparatively, the fourth order Runge-Kutta method will consistently experience drift in its energy, but for small enough time steps, this drift is negligible compared to the oscillations in the energy using the velocity Verlet method. So, since the Runge-Kutta method is fourth order, for small enough time steps, it out-performs the second order velocity Verlet method.

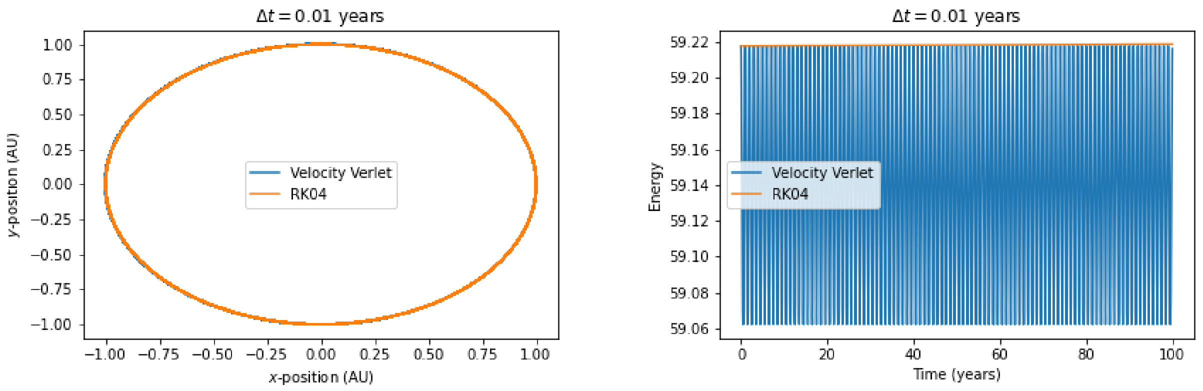

Figure 1: Trajectory and energy for Velocity Verlet method and Runge-Kutta method, with $\Delta t = 0.01$ years

As the time step increased, we begin to see this behavior break down. For the Verlet method, as the time step increases, we begin to see ripples in the amplitude of the energy-time plot, whereas the Runge-Kutta method experiences energy runaway. Moreover, the trajectory computed using the velocity Verlet method oscillates about a fixed circular orbit, as in Figure 2. The Runge-Kutta method, on the other hand, is unable to maintain an orbit due to the energy runaway. This illustrates the utility of symplectic integrators, since the velocity Verlet method can more robustly compute an orbit at larger time steps, and this data motivates us to implement a fourth order symplectic integrator.

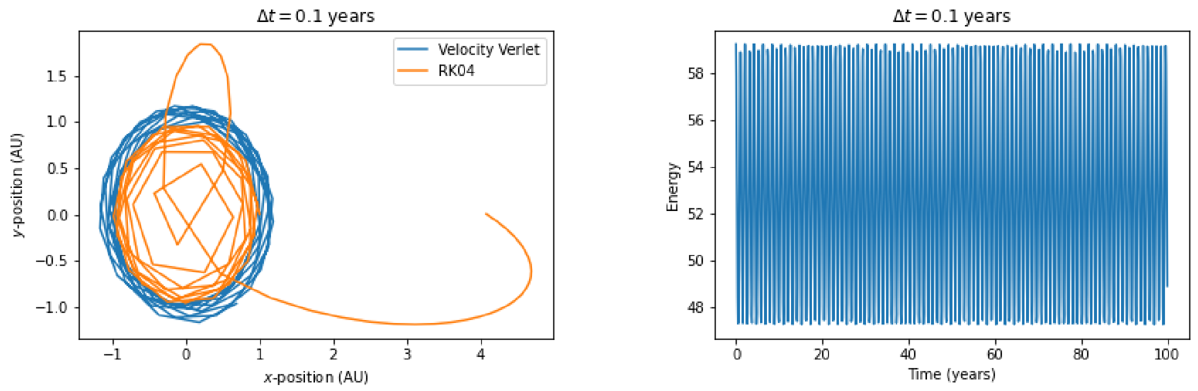

Figure 2: Trajectory and energy for Velocity Verlet method and Runge-Kutta method, with $\Delta t = 0.1$ years

4.2 Yoshida Method

To improve our analysis, we next implemented the fourth order Yoshida method, which offers several improvements to the Verlet method. Like the Verlet method, the Yoshida method is symplectic, but it is also fourth order, which should reduce the error on the energy in comparison to the second order Verlet method. Moreover, the Yoshida method conserves angular momentum, and it is time reversible. See [2] for the equations and coefficients for the Yoshida method.

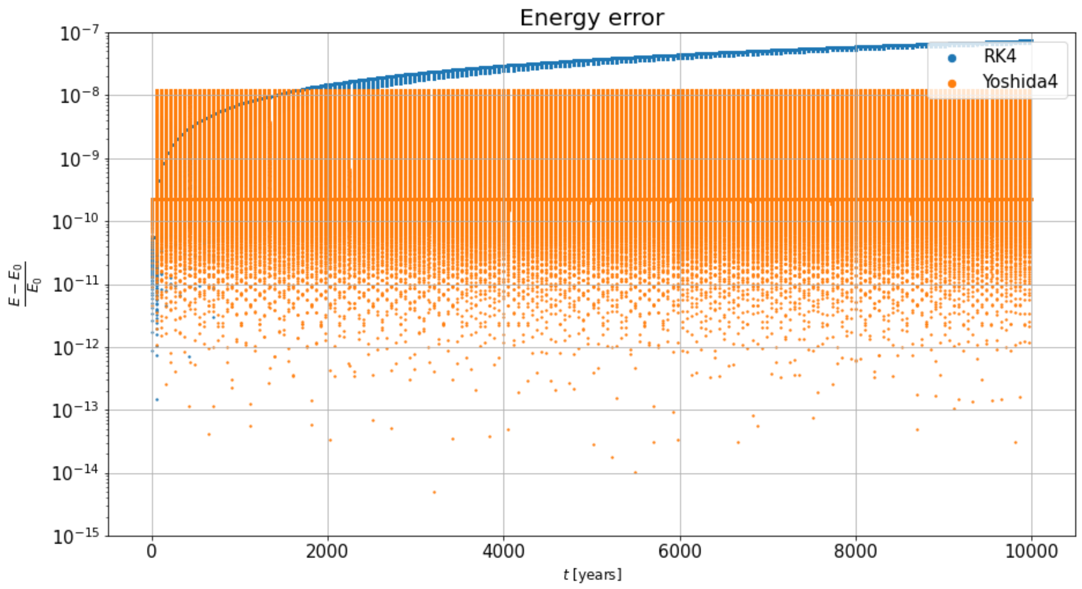

Figure 3: Error in energy as a function of time for Yoshida method and Runge-Kutta method, $\Delta t = 0.001$ years

In Figure 3, we again consider a single planet with a circular orbit ($\Delta t = 0.01 =$ years), and we compare the error in the energy for the fourth order Yoshida method and for the fourth order Runge-Kutta method. We have seen already that the error for the Verlet method has a large amplitude compared to the drift of the Runge-Kutta method. For the Yoshida method, however, the error in the energy is on the same order of magnitude as the Runge-Kutta method for small enough periods of time. Moreover, the error on the Yoshida method is bounded, unlike the Runge-Kutta method, which gives the Yoshida method much greater stability over long periods of time. Thus, the Yoshida method both yields a smaller error compared to the Verlet method due to being fourth order, and it is significantly more stable than the Runge-Kutta method, due to its symplectic nature.

5 Code Structure

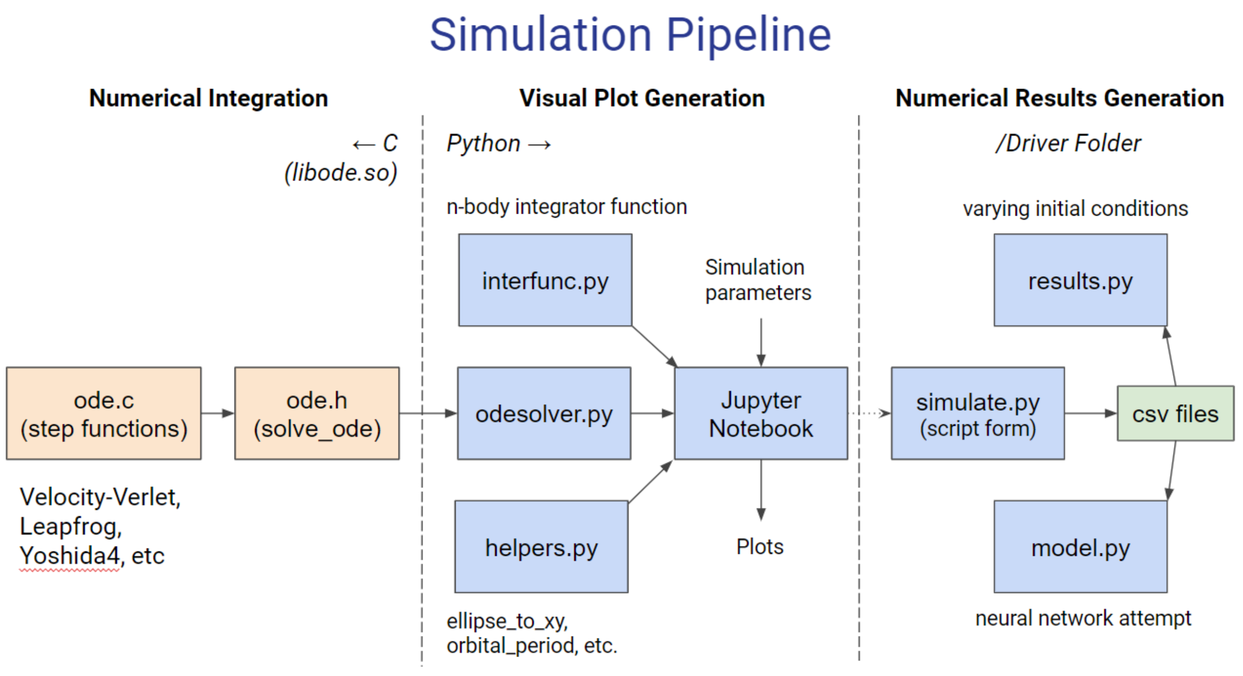

Figure 4

The code to run the simulations is comprised of three main branches. First, we implemented the velocity Verlet and Yoshida solvers in C. Then, we use Python to load the C libraries and call the solvers. For our initial results, we then generated plots in Jupyter Notebooks. The third section of our code (in /Driver folder on Github) is used for scripting and running large numbers of simulations with varying initial conditions, so that the stability results can be numerically quantified and also attempted to be modelled using neural networks.

6 Results

6.1 Orbital Plots

The system we consider for our simulations is our solar system, with initial parameters given by [3]. All planets are initially set to their average orbital radius and eccentricity, with $\theta = \theta_E = 0$

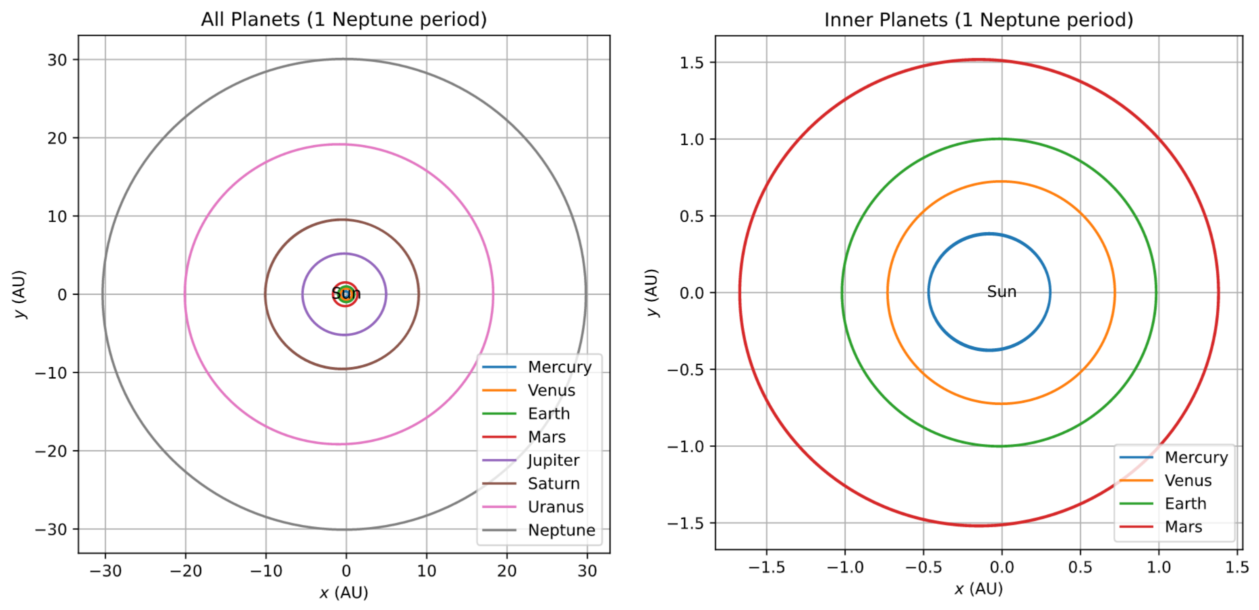

Figure 5: For the solar system plots shown above, the simulations proceed as expected. The left plot shows the 2D-orbits of all planets whereas the right plot only shows such for the inner planets. The simulation time length is set to 1 Neptune period, so that all planets have a chance to complete at least one orbit. These results indicated that the n-body integration was working properly, at least for the initial timescales.

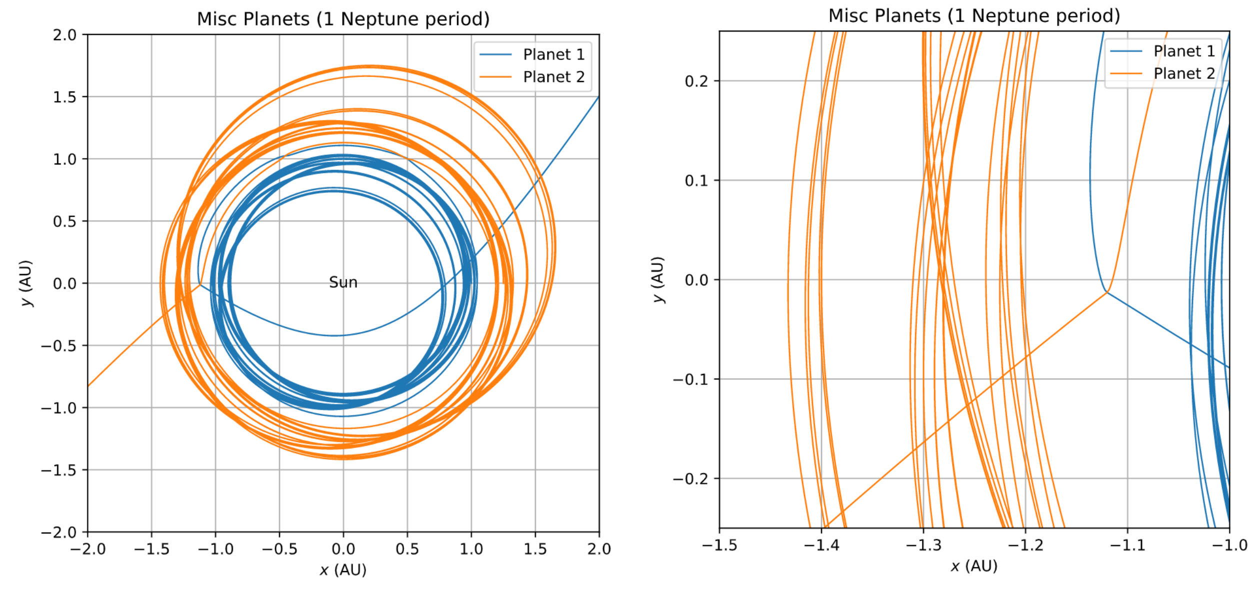

Figure 6: As a test to find an unstable configuration of hypothetical planets, we simulated two Jupiter-mass Planets orbiting the Sun. They start at 1 AU and 1.25 AU respectively, but their orbits become unstable within a Neptune period timescale, as seen in the left plot above. Eventually, their varying orbits bring them close enough to each other so that the gravitational forces between them skyrocket, causing them to slingshot each other out of the designated system (our unstable criteria). This near-collision is given a closer zoom in the plot on the right above.

6.2 Stability for Many Simulations

Our criteria for determining that a particular simulation was unstable were cases where Mercury's orbital radius eventually exceeded 10 AU, far greater than Mercury's average orbital radius of 0.387 AU. Thus, for our analysis, a case is either stable or unstable, based on whether Mercury is ejected from its orbit.

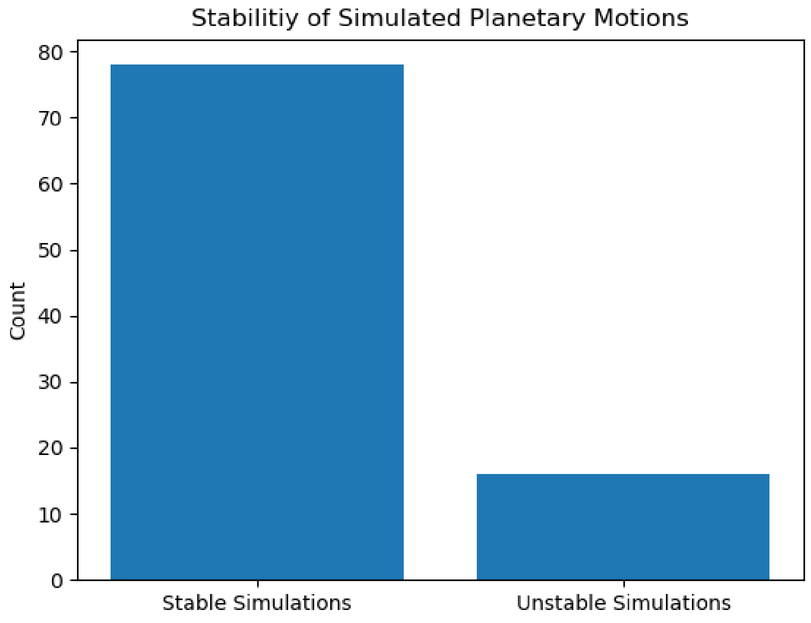

Figure 7

We ran about one hundred simulations of the solar system, each with slight variations in the initial conditions. In particular, we varied Mercury's position and velocity by selecting random values from Gaussian distribution with standard deviations of .05 AU and .05 $\frac{\text{AU}}{\text{Year}}$, respectively, and then adding them to the known parameters for Mercury. We chose this variation to be relatively high due to computational constraints. For small perturbations, Mercury would be ejected from orbit on a time scale of 10^9 years, and it took nearly an entire day to run a hundred simulations over about $150,000$ years. Thus it was necessary to introduce somewhat large perturbations for Mercury to occasionally be ejected from its orbit. Even so, we found that about 17% of our simulations were unstable.

There are two immediate improvements that we could consider implementing. First, we only consider the stability of Mercury, since it is the least massive planet and closest to the sun. Thus, it is the most volatile and mostlikely to be ejected from orbit. However, it is feasible that another planet could be ejected from orbit as well, so we could check these planets to improve the analysis. Another improvement we could make would be to quantify how unstable an orbit is, rather than simply analyzing stability as a binary function. One possible method of implementing this would be to consider the rate of change of the amplitude of a planet's perihelion.

6.3 Neural Network

Using the our generated stability data, we attempted to build a neural network that could predict the stability of a given planetary system. With PyTorch, we built the framework for a machine learning model that took as input the positions and velocities of every planetary body in the system and outputted a value between 0 and 1 representing the predicted stability of the system after about $150,000$ years. When we attempted to train the model, its performance improved greatly at first from 43% accuracy to 57% accuracy after one round of training. After that, however, more training did not improve the model. We believe that a larger dataset with more balance between stable and unstable samples could improve the performance of the network.

7 Conclusion

In conclusion, we successfully implemented a fourth order symplectic integrator to simulate planetary motions. While there is room to improve our analysis, we found that over long periods of time, planetary orbits tend to be stable and require relatively large perturbations to become unstable over these time periods. We also began implementing a neural network, although more data is needed to successfully complete this task.

References

[1] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations.* Springer-Verlag Berlin Heidelberg, 2006.

[2] Haruo Yoshida. Construction of higher order symplectic integrators. *Physics Letters A*, 150(5):262–268, 1990.

[3] David R. Williams. Planetary fact sheet - ratio to earth values. `https://nssdc.gsfc.nasa.gov/planetary/factsheet/planet_table_ratio.html`, 2019.