# An Analysis of an N-Body Simulation

Harry Sebastian Spratt

**This project endeavoured to produce an accurate N-body gravitational simulation of the solar system. Gravitational N-body simulations use the equations of motion to model how gravitationally bound masses interact with each other. These simulations are widely used and have several applications within Astrophysics from a few bodies within a solar system up to the scale of a galaxy where it can be used to model how galaxy clusters form. A key part of the project was ensuring it was physically correct per the laws of nature, as such, conservation laws such that of energy and momentum were closely tracked in the simulation to evaluate this. I found my simulation obeyed these laws with the end values typically varying < 1% within a 150-year timeframe. In addition to the conservation laws, ideas such the Virial theorem are discussed to evaluate the accuracy and stability of the orbits in the simulation. In addition to an n-body simulation, there is also a script to evaluate the Lagrange points of a two-body system. All code was simulated with Python 3.8.8 in Visual Studio.**

## I. Introduction

The purpose of this report was to create a physically accurate simulation of N bodies, meaning that each body is interacting with all other bodies. The N bodies, in the case of this report, are built up from simple two-body configurations to that from the solar system. All of which were configured using the GUI, however, other configurations of hypothetical star systems can be computed. The code computes the position, momenta and energies (both KE and PE), of each body in the simulation at every time step. This can be compared to experimentally know data to check the validity of the simulation. The simulation makes use of Newton's Law of gravity, which dictates the equation of motion, the physics of which will be discussed in the preceding section. The later sections go on to describe the program and then provide quantitative analysis of the accuracy.

## II. Theory of the N-body problem

### A. Newton's Laws

The gravitational N-body problem relies on two of Newton's Laws. The Law of Gravitation and Newton's Second Law. The first being that two objects $B_i$ and $B_j$ with a mass $m_i$ and $m_j$ will exert an attractive force on each other ($\vec{F}_{ij}$), the magnitude of which is inversely proportional to the distance apart, and the second being that an object will experience acceleration if there is a resultant net force on that object. These two laws can be expressed mathematically as:

$$\vec{F}_{ij} = \frac{G m_i m_j}{r_{ij}^2} \hat{\vec{r}}_{ij} \tag{1}$$

$$\vec{a}_i = \frac{\vec{F}_i}{m_i} \tag{2}$$

with the acceleration being a vector with x, y and z components. The technique I have implemented in this simulation is a sequential pair algorithm that uses brute force to calculate the forces on all the bodies. It is the simplest and most accurate but also the most computationally intensive. There are other more effective algorithms such as parallelisation of pairs algorithm and Sequential Barnes-Hut Algorithm however no detail will be gone into these other algorithms.

### B. Integration Methods

The simulation required the velocity and acceleration to be calculated from a set of initial conditions which was done by numerical integration. As the solutions for the N body problem become more and more complex sophisticated integration techniques are require. Aside from the two-body problem, which has a closed-form solution, no such equivalent solution exists with the N body solution as for most initial conditions the system turns chaotic and therefore numerical solutions are needed. Exact solutions do exist for the N body problem however these require precise initial conditions and typically with masses that are all equal [2]. Any perturbation to these conditions as said will become unstable and descend into chaotic motion. Therefore the equations are solved numerically and typically by using the Runga-Kutta method. The Euler and Euler-Cromer are simple versions of the Runga-Kutta which can be coded manually, however, python has an incredibly powerful function called `scipy.integrate.solve\_ivp` which uses RK45 (4th order Runga-Kutta method), so that was used for the purposes in this simulation. The benefit of RK45 being that the total accumulated error is in the order of $\mathcal{O}(h^5)$, which provides a high level of accuracy. Another problem that is typically encountered when integrating with N-body simulations is the time-step. If the time step is too large the physics breaks down however if too short and the simulation takes too long to compute. This is something `solve\_ivp` does incredibly well at as it has inbuilt absolute and relative tolerances on the integration which can be edited in the GUI, enabling complete control over the integration. What this enables is for the time steps to vary, meaning it will take the longest time step possible while still maintaining the accuracy below the tolerance levels. This clever technique allows for faster computation times compared to constant time steps.

### C. Energy and Momenta

Using OOP (object-oriented programming) and classes, each object had an associated Kinetic energy, potential

energy, linear and angular momentum. This was stored in the form `planet.KE` (planet being the name of the particular planet). This is was so it could be seen how these values changed over time in the simulation. Before running the simulation we can predict how these variables will change over time. When there is no external torque it has been established that angular momentum is conserved, the same is true for linear momentum but in regards to an external force. The angular momentum for an orbit in the simulation of the solar system should be constant and therefore conserved. As there is no external torque applied in the system since the gravitational force on the sun and the position vector are at 180° to each other, angular momentum is conserved. This can also be shown mathematically:

$$\vec{\tau} = \vec{r} \times \vec{F} \tag{3}$$

$$\therefore \vec{\tau} = \vec{r} \times f(r)\hat{r} = 0 \tag{4}$$

where $\tau$ is the torque, $r$ is the position vector and $F$ being the force. It follows from (4) that $\frac{d\vec{L}}{dt} = 0$. So $\vec{L} = constant$. This conservation applies both locally and globally. The conservation between the sun and an orbiting planet and also the conservation over all the planets. In contrast, linear momentum is not conserved locally. Linear momentum is given by:

$$\vec{p} = m \cdot \vec{v} \tag{5}$$

where $p$ is the linear momentum and $v$ being the velocity vector. Since the path which the orbits follow are elliptical and not circular, the position vector is not perpendicular to the velocity therefore the linear momentum of a planet is not constant. However, as true with angular momentum it is conserved globally over all the planets.

Other conservation laws that have to be considered when assessing the accuracy of the system, are that of potential and kinetic energy. The calculation for the kinetic energy is straightforward and calculated from

$$T = \frac{1}{2} m \cdot \vec{v}^2 \tag{6}$$

The potential energy is also given by a simple formulae:

$$U = -\frac{G m_1 m_2}{\vec{r}} \tag{7}$$

however the complexity comes from calculating the total potential energy which has to be summed over the interactions $(n-1)$ between every object in the system, this leads to the total potential for N objects being given by:

$$U = \sum_{i \neq j}^{N} -\frac{G m_i m_j}{\vec{r_{ij}}} \tag{8}$$

Following this, both the potential energy and kinetic are summated together, the result of which should be constant, assuming the simulation is accurate.

An important consideration when assessing the accuracy of the simulation is considering that a conserved quantity $A$ is the sum of its components such $A = \sum_i A_i$, the time variation of $A$ should be compared with each $A_i$ individually. Therefore to truly check the accuracy of the simulation the conservation laws and energies need to be compared to values within the same components such as all in the $x$ direction. In short, variations much be compared with other variations.

A valuable tool to check the stability of the system using energy is the Virial theorem. This theorem relates the kinetic energy and potential energy in that of a stable system. Mathematically, in its general form, it states:

$$\langle T \rangle = -\frac{1}{2} \langle V_{TOT} \rangle \tag{9}$$

with $\langle T \rangle$ being that of the average total kinetic energy and $\langle V_{TOT} \rangle$ being the average total potential energy. This relationship can be used to analyse the stability of a system, which, for the simulation will test its accuracy.

### *D. Lagrange Points*

Lagrange points are positions of space where the gravitational force is exactly equal to the centripetal force needed to keep the objects in motion. Therefore they are points of relative stability. $L1$, $L2$, $L3$ are unstable Lagrange points as any perturbation of the position around these exact points will lead it to no longer being stable. $L4$ and $L5$ however are stable Lagrange points. These points do not lie in the plane of the two planets but instead lie on the vertices of the equilateral triangle formed between that the two large masses. These points are incredibly useful in space exploration as they provide positions in which spacecraft can orbit and remain in position with relatively little fuel consumption

## III. Program Design

The python program has four files for the simulation aspect of the N body simulation, being; `n_body_app`, `Objects`, `solar` and `Simulation (GUI)`. There is an additional script that models the Lagrange points of two given masses but is not connected to the N body simulation. In all scripts, OOP has been aimed to be used to create cleaner scripts.

### *A. n_body_app*

The first file `n_body_app`, contains the class n_body_app. This class initialises the GUI. Within this script, the frame is created with `tkinter`, and both the text and buttons are designed here. The main use of this GUI is to use user inputs such as choosing the planets, integrator and constant to use in the simulation. The aim is that the user can have as much control over the main controls of the simulation as changing it directly into the code, allowing for different initial conditions to be tested. This allows the exploration of hypothetical constellations of planets and stars with relative ease. The user must choose from the list of planets and

constellations. Nine planets and two suns can be added to the simulation, in any combination chosen. In addition to these, there are set configurations of planets that have been preset which can be chosen, all of these configure 3 stars orbiting each other with carefully chosen initial conditions. For each text box, the user has to decide what set-up is desired, after which they confirm the selection. If no edits are made the default simulation will be run which be that of the solar system, with RK45 ODE over 100 years. The typical time for this script is ≈ 5s.

### B. Objects

This file contains the Objects class. This initialises all the objects used in the simulation. To initialise an object the name, mass and initial conditions such as the position, velocity and acceleration need to be provided. The initial conditions in the case of planets in the solar system were found on the JPL NASA website. This is the core behind the simulation as all values will be assigned and saved to a class. In addition to this, it also provides the function which details Newton's Law of Gravitation needed for an integrator to calculate the new position and velocities.

### C. solar

Similar to the Objects class, the solar class has key functions which run the simulation. The main aspect of this script is to return the values for the energies and momenta of each particle. An array of $(t, 3N)$ (t being the umber of iterations and N the number of particles), will be passed through the function `GetEnergy` and for each time point all these values will be calculated for ever object in the class.

### D. Simulation (GUI)

The final file is the one in which the data is produced, plotted and where the simulation runs. The main component is the integrator `scipy.integrator.solve_ivp` which cleverly computes the acceleration and velocity with a variable time step. Enabling faster computation time as when it is not needed the time step will not be as low as long as its within the `rtol` and `atol` limits. Both of which can be changed in the GUI along with the integrator.

### E. For loops and matrices

One important point to mention is how values such as the distances between planets, acceleration and gravitational potential energy are calculated. Typically to calculate these values a two for loops are run as such:

```
for planet in self.planets:
        for otherplanet in self.planets:
                if planet != otherplanet:
```

However, these for loops are time-consuming and there is a much more efficient way to compute these values using matrices. It is not deemed "mathematically" correct but considering calculating the distances between all particles, using: $dx = x - x.T$ is far more efficient. x being the matrix of x positions. Typically you are not allowed to add or subtract matrices with dimensions as it has no mathematical meaning, however, doing this produces an array with all the distances between every other particle and without the need for any for loops or if statements. The use of matrix manipulation is used through these scripts and allow the efficiency of the scripts to improve.

## IV. Testing the simulation

### A. Planetary Orbits

This project aimed to obtain a working code for an N body simulation, however, this needed to be built up. Firstly the code was probed to check the code and solutions for a two-body problem in 2D. The aim of which was to produce closed elliptical orbits to assess if the simulation was working.

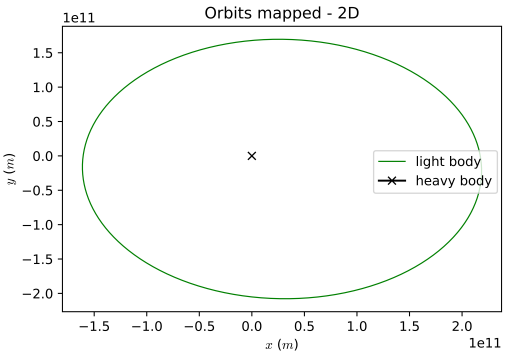



**Fig. 1:** 2D elliptical orbit of planet - sun like system

This is exactly what Figure 1 shows. It shows the 2D elliptical orbit of a light planet around that of a heavier one. Similar to the orbit that of a planet around the sun. After this graph had been obtained showing a closed orbit that obeyed the conservation laws, the code was then expanded to allow for interactions between N particles. This particular simulation of a small mass body orbiting a large one is the simplest mathematically as it can be simplified to a one-body problem. As it is typically assumed that the large bodies orbit is negligible to that of the smaller planets.

So far the simulation demonstrates that it can successfully produce orbits for a Sun-Earth configuration and binary star configuration in an accurate manner being that of a closed-loop. To further test the 2D simulation two N-body choreographies were simulated. These are periodic solutions to the N-body equations in which N equal masses chase each other around a fixed closed curve, equally spaced in phase along the curve.[source] The two solutions tested were the figure of eight solutions and the triple ring solution.

Figures 3a and 3b show the orbits of the analytical. Both orbits are over a 200-year time frame, the reason for this is to demonstrate the stability of the orbit and therefore confirm the accuracy of the simulation. These

are known stable orbits, but if the simulation was not computing Newton's Laws correctly they would not appear stable. Units are not shown on these graphs, as these constellations are hypothetically, therefore, the units are arbitrary and meaningless.

Therefore to test the simulation, a simulation of two heavy bodies orbiting each other was run as shown in Figure. In this simulation, both masses are significantly affecting the orbit of the other leading to it being more complex mathematically. The figure shows the orbits of Alpha Centauri A & Alpha Centauri B, a binary star system 4.37 light-years away. This demonstrates that the simulation can successfully compute a stable two-body orbit.

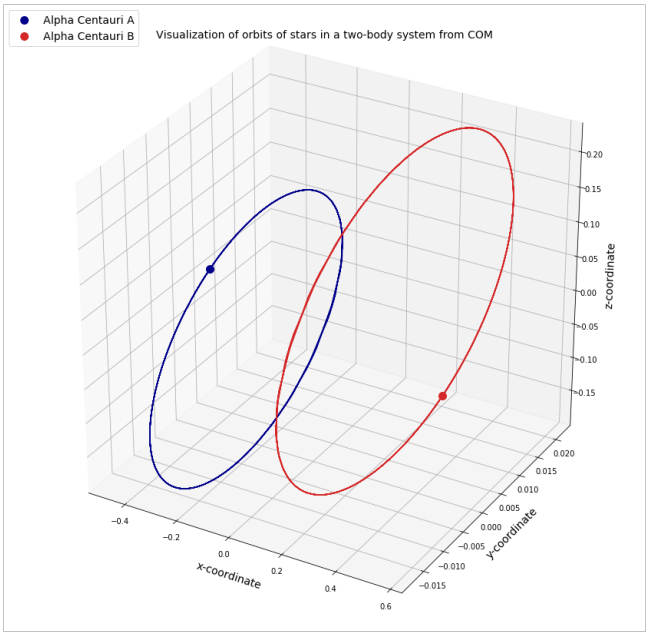
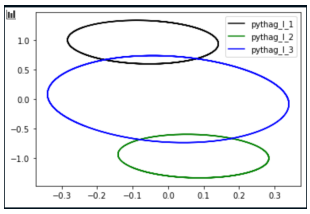
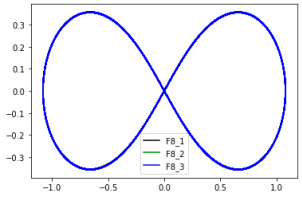



**Fig. 2:** Orbits of stars in known exoplentry system Centauri

After confirming the accuracy and stability of the analytical solutions above the N body code was expanded to 3D. In particular, with the 3D code, the solar system will be analysed as precise ephemeris data of all the planets is known allowing for comparisons to be made as there are known solutions. With the code expanded to 3D $(x, y, z)$, the accuracy of the simulation can again be probed. Firstly to test the simulation will run for the Sun-Earth system for one exact year or 31536000s. If the simulation is accurate




**(a)** Orbit of triple rings configuration




**(b)** Orbit of Figure of Eight configuration

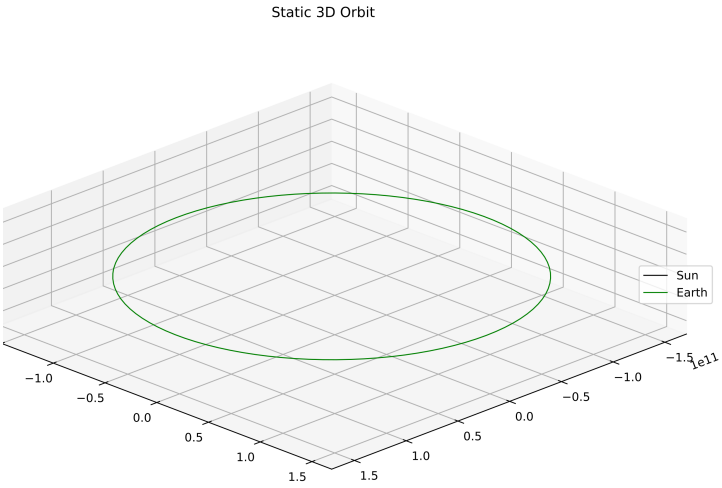



**Fig. 4:** Orbit of Earth over 1 year

| Planet | $\Delta x$ (%) | $\Delta y$ (%) | $\Delta z$ (%) | $\Delta r$ (%) |
|---|---|---|---|---|
| Mercury | 1.21 $\times 10^{-3}$ | -1.33 $\times 10^{0}$ | -2.69 $\times 10^{-1}$ | 8.89 $\times 10^{-2}$ |
| Earth | -0.188 | 0.126 | 1.25 | 1.78 $\times 10^{-3}$ |
| Neptune | 3.02 $\times 10^{-4}$ | -4.61 $\times 10^{-3}$ | -1.12$\times 10^{-2}$ | 4.66 $\times 10^{-4}$ |

**TABLE I:** Displays comparison of values of Simulation vs JPL

and correct it should produce exactly 1 full orbit around the sun. As shown in Figure 4 one full orbit is completed and shows a closed orbit indicating it has a high degree of accuracy. Next the simulation was run for the whole solar system, the orbits of which are shown in Figure 5. An alternate method of testing the simulation is to check whether the code computes the correct acceleration values. As the equations are known the first few values can be calculated by hand and compared to the simulation values. It was found by this method that both values were in agreement with each other. Another method to verify the accuracy of the simulation would be to compare the simulation positions of planets after a year in comparison to the values given by JPL's Horizon system. The ephemeris data provided by JPL is tied to the International Celestial Reference Frame with an accuracy of 0".0002 [source]. This corresponds to an accuracy of approximately a few hundred meters. A comparison of was performed under the following parameters changed; Number of periods = 1, Iterations per period = 50 and relative tolerance = $\times 10^{-5}$. This yielded the data shown in Table I. The comparison shows there is little difference between that the precisely calculated JPL ephemeris data and the data provided in the simulation. Even 78 years, the Simulation is in concurrence with the JPL ephemeris data with a maximum disparity of 0.44%.

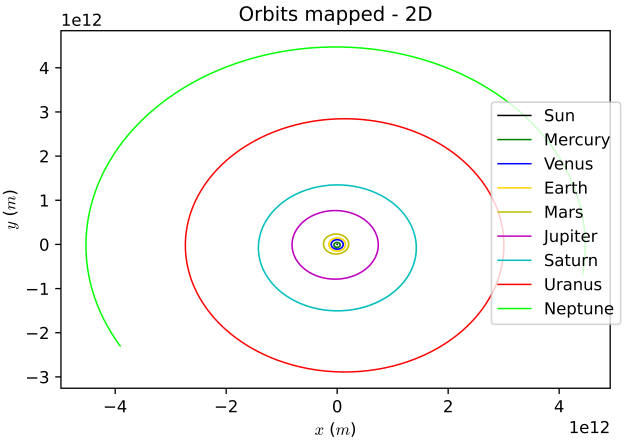



**Fig. 5:** Orbit planets in solar system over 100 years

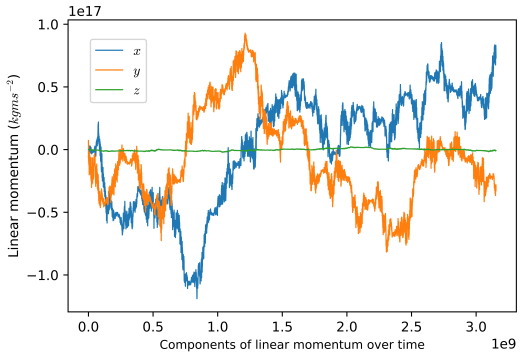



**Fig. 6**

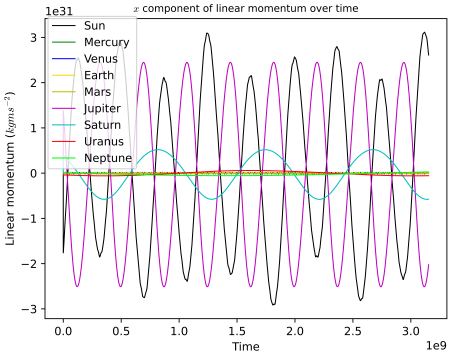



**Fig. 7**

### *B. Momenta*

Additional variables to consider the conservation of are that of linear and angular momentum. How these two variables change over time are demonstrated in Figures x and y. A system conserves linear momentum if there is no external force. This is true of the whole system however given the sun-earth system, linear momentum will not be conserved, only when taking the solar system as a whole. Figure 6 shows the linear momentum in the $x$, $y$ and $z$-direction over time. It is most useful and prudent to compare variations to variations such as comparing $x$ components of linear momentum. The linear momentum of the planet in the system fluctuates as shown in Figure 7 showing the linear momentum fluctuates and does not seem to be conserved. However this discrepancy is due to the non-exact numerical conservation which is a combination of the error due to the integration algorithm, and round-off error. As seen there is irregular variation: just round-off error such behaviour is expected if you are using a symplectic algorithm. As shown in Figure 6, 7 and 8 although the linear momentum fluctuates, a direct result of using an integration algorithm, the values fluctuate around a fixed point, showing that energy is not being lost in the system. As shown in Figure 7 the individual components of the total momentum vary almost 15 orders of magnitude less than each planet x component. This implies that assuming 64-bit representation for your floating-point numbers, the integration algorithm conserves the momentum within machine precision. To overcome these problems, the distances could be converted from meters to a natural unit such as AU. This is regarded as non-dimensionalisation and as well as cutting down on computation time, avoids the floating number errors.

Much like the linear momentum the angular momentum of the system should also be conserved. Figure 9 shows that in the simulation angular momentum is conserved. Although the axis on the matplotlib graph can be deceiving the angular momentum is only varying by $2.46 \times 10^{-5}$.

### *C. Energy and Virial's Theorem*

The final aspect of the system to analyse is that of its energy. As discussed before it would be expected that, according to the conservation of energy, the total energy of the system to be constant. Although the kinetic energy varies sinusoidally, so too does that of the potential energy, out of phase of each other. Therefore this still allows for a constant total energy as they superimpose to cancel flatten to a straight line. Figure 10 shows the kinetic energy contribution of each planet in the system. As shown, the kinetic energy of the planet fluctuate over time. With Jupiter having the greatest KE, not from being the furthest away and therefore the greatest speed, but due to its mass contribution, being the heaviest in the solar system bar the sun.

Figure 11 energy nicely shows the combination of the kinetic and potential energy, along with the total energy and a demonstration of the Virial theorem. From $t = 0$ of the system till the end the total energy of the system changed by $6.348507900516524e-06\%$ and there is no visible spiralling of the system which is typically a problem with numerical integrators. Therefore it has been shown that the system obeys the conservation of energy accurately. In addition to this Figure 11 also demonstrates the Virial theorem, which states that, in its simplest form, $2KE + PE = 0$, shown in 9 . This relationship is demonstrated by the graph. The Virial theorem is true for a stable

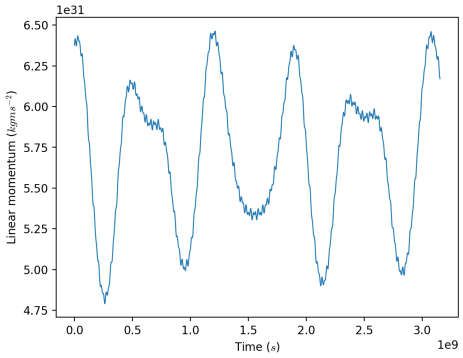



**Fig. 8**

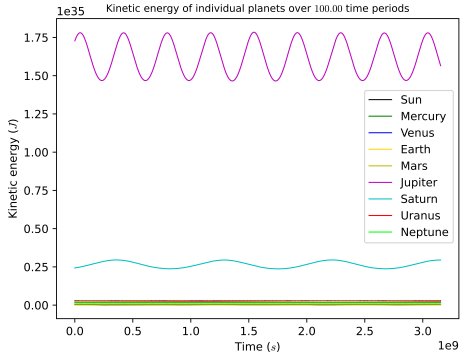



**Fig. 10**

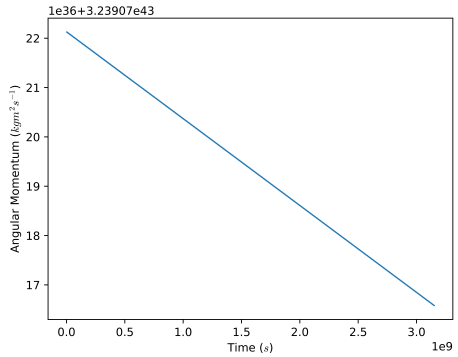



**Fig. 9:** Orbit of Earth over one year

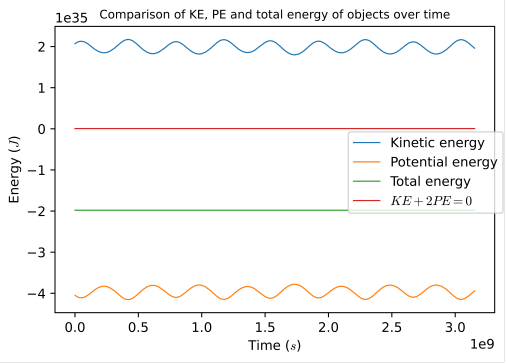



**Fig. 11**

system of discrete particles. This again demonstrates the stability of the system, such that Virial's theorem applies.

## V. Solar System under close encounter

A small bit of analysis was done when the system encounter another star. Where is is shown via the virial theorum that the orbit is not stable [expansion needed]

## VI. Conclusion

In this project, initially, a simple 2D orbit between two orbits was completed, now however it has been expanded to an N body simulation in three dimensions the conservation laws are adhered to and the system can be deemed accurate.

The table above gives a nice summary as to what my simulation can run.

| Checklist | Completed |
|---|---|
| 2D code with closed orbits (mercury) \| | ✓ |
| Two light bodies orbiting a heavy one | ✓ |
| Two heavy bodies in close orbit | ✓ |
| Expanded 2D code to 3D with N bodies | ✓ |
| Checked conservation laws of system | ✓ |
| Studied evolution of solar system under encounter | ✓ |
| Identified the Largrange points between two large masses | ✓ |

**TABLE II:** Displays comparison of values of Simulation vs JPL

## References

[1] En.wikipedia.org. (2018). N-body problem. [online]. Available: `https://en.wikipedia.org/wiki/N-body_problem`. [Accessed 7 Jan. 2018].

[2] Li, Xiaoming, (Jul - 2019). Collisionless periodic orbits in the free-fall three-body problem.

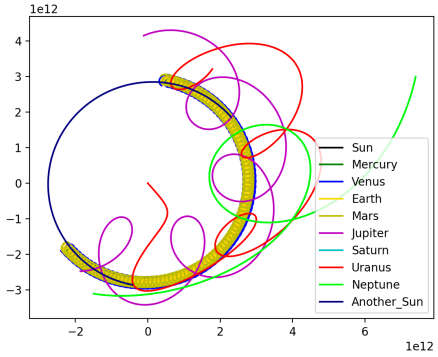



**Fig. 12**

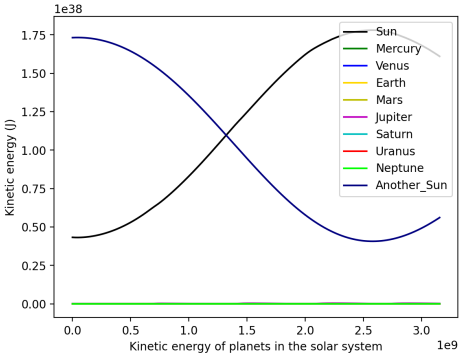



**Fig. 13**

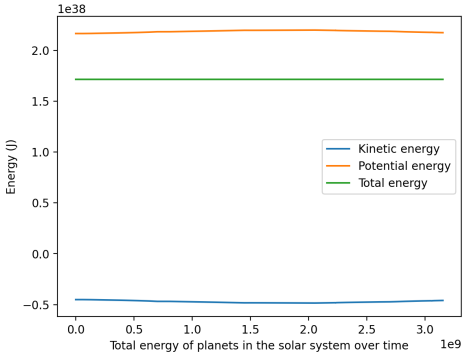



**Fig. 14**