Cambridge University
Physics Part II: Computing Project
Trojan Asteroids

Blind Grade Number: 6908X

May 2, 2022

Word Count: 2993

Contents

1 Introduction **4**

2 Theoretical Background **5**
2.1 Derivation of Lagrange Points 5
2.2 Units 7
2.3 Orbits about off-axis Lagrange Points 7

3 Computational Analysis **8**
3.1 Scaling and Performance 8
3.2 Integrator Choices 8

4 Code Implementation **11**
4.1 Frame of Reference for ODE Solver 11
4.2 Abstracting Code 11
4.3 Storing Data 12

5 Results and Discussions **13**
5.1 Orbits of Asteroids 13
5.2 Stability of Lagrange Point 15
5.2.1 Initial Displacement from the Lagrange Point 16
5.3 Velocity Shift in X-Y Plane 20
5.4 Long Term Stability 24
5.5 Mass Ratio of Bodies 26

6 Conclusions **32**

A Mathematical Derivations **34**
A.1 Derivation of Lagrange Points 34
A.2 Mass Ratio 34

B Source Code **35**

Abstract

This report investigates off-axis Lagrange points and their stability to small perturbations by initial displacement and velocity offsets in the plane of orbit. Python was used to compute orbits about these stationary points, with most of the simulations being run for the Sun-Jupiter system in which 'Trojan' and 'Greek' asteroids collect. The initial displacement at which an orbit could become unstable was found to be 0.7AU while the corresponding initial additional velocity was found to be 0.7AU/year. The relationship between an asteroid's wander, W, from L4 and the mass ratio of the two bodies was found to be $W = 0.3\gamma^{-0.26}$ for small $\gamma = \frac{M_1}{M_1+M_2}$. The maximum mass-ratio at which off-axis Lagrange points may be stable was 0.042, within 5% of the theoretical prediction.

1 Introduction

There are five positions in a rotating two body system at which a third object will be in equilibrium. These are called Lagrange points. There are three on the axis joining the two bodies and two which sit symmetrically on either side, equidistant from the two masses. Lagrange points are crucial to the progression of our knowledge of the universe. Currently, many space probes operate at the Lagrange points of the Earth and Sun. One recent example is the James Webb Space Telescope, which will give us readings of distant galaxies far more precise than even the Hubble Space Telescope.[1]

The off-axis Lagrange points will be investigated in this report. Asteroids (categorised 'Greek' and 'Trojan' for each off-axis Lagrange point in the Jupiter-Sun system) collect at these locations, suggesting that they are stable in some scenarios. This stability can be shown by consideration of the Coriolis Force in the rotating frame of reference. We must understand the behaviour of small perturbations around them and investigate the nature of these equilibrium points in order to make use of them in real life. This is the main aim of the report.

First, a brief overview of the mathematical details giving rise to such stationary points will be given. The computational analysis and code implementation will be discussed afterwards, followed by results with emphasis on the stability of asteroids near off-axis Lagrange points.

It was found that asteroids offset up to (0.7 ± 0.1)AU in any direction within the plane of orbit were likely to remain in stable oscillations about the Lagrange point. The same applies for asteroids given an additional velocity of up to (0.7 ± 0.1)AU/year within the plane of the orbit. The maximum mass ratio, $\frac{M_1}{M_2}$, where $M_1 < M_2$, for a stable off-axis Lagrange point was found to be 0.42 ± 0.1, which was within 5% of the theoretical prediction.

2 Theoretical Background

2.1 Derivation of Lagrange Points

A detailed derivation of each of the five Lagrange points can be found in the Appendix. A brief overview and discussion is given here.

The five Lagrange points are labelled L1, L2, L3, L4 and L5. L1, L2 and L3 lie on the axis joining the masses, while L4 and L5 lie off the axis. In the literature, L4 generally precedes the motion of the smaller orbiting mass while L5 follows it.

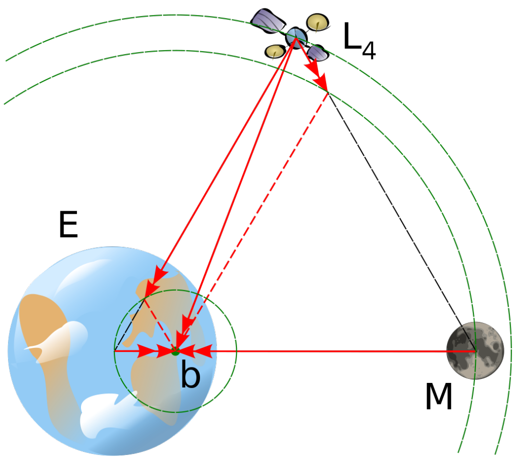


Figure 1: Diagram of Lagrange point L4 between the Earth and Moon. Point b is the centre of mass of the system.[4]

Consider two masses ($M_1 < M_2$) orbiting one another. The centre of mass of the system lies somewhere between the two masses, on the axis joining them. Let's consider the centre of mass (CoM) frame which is rotating at the angular frequency of the masses' orbits. The angular frequency about the centre of mass is given by

$$\omega_{\mathrm{CoM}}^2 = \frac{G(M_1 + M_2)}{R^3}$$

for both masses.

Considering figure 1, the Moon and the Earth both attract the satellite at L4 towards themselves with different forces which add vectorially towards the CoM. The centrifugal force holds the satellite in balance, acting radially outwards. This gives rise to the stationary points at L4 and L5. In figure 2, a contour plot of the effective potential (U_{eff}) shows the two off-axis Lagrange points.

$$U_{\text{eff}} = U_S(|\overrightarrow{r} - \overrightarrow{r_s}|) + U_J(|\overrightarrow{r} - \overrightarrow{r_j}|) - \frac{1}{2}(r\omega)^2$$

This figure is plotted for the Jupiter-Sun system.

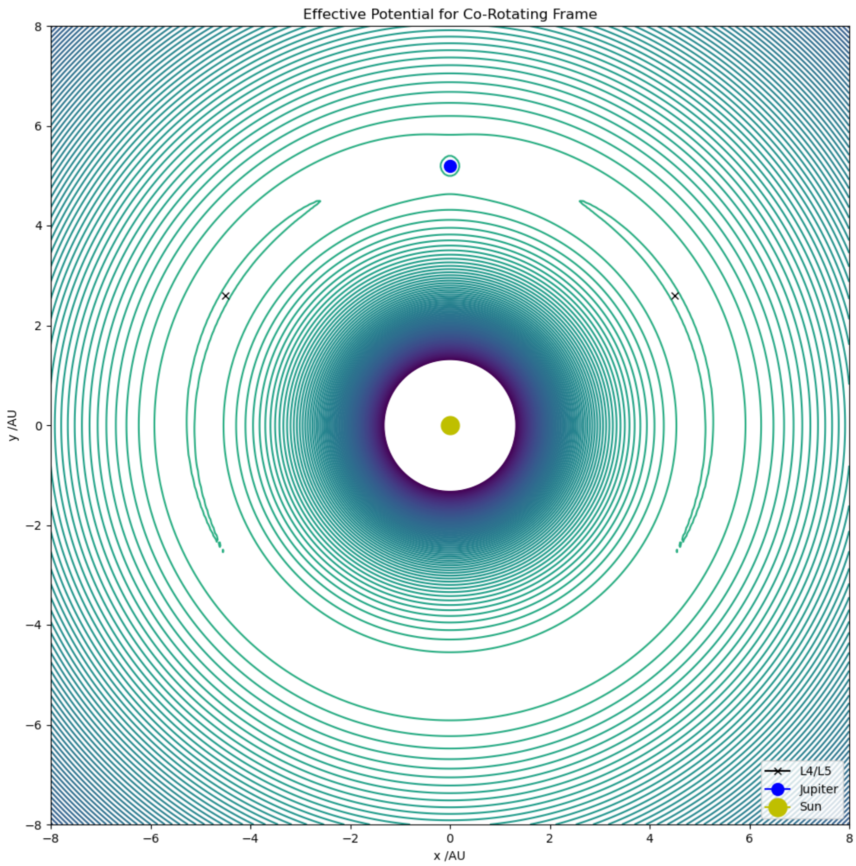


Figure 2: Effective Potential of Jupiter-Sun system in the rotating CoM frame. L4 and L5 are shown, while the other three Lagrange points lie on the axis joining Jupiter and the Sun.

In order for the Lagrange points L4 and L5 to be stationary, we must also consider the Coriolis force in the rotating frame. ($\overrightarrow{\omega} = \omega\overrightarrow{e_z}$)

$$\overrightarrow{F_{\text{cor}}} = \begin{pmatrix} 2\omega v_y \\ -2\omega v_x \\ 0 \end{pmatrix}$$

2.2 Units

In this project, the following 'solar system units' were used.

- Mass of Sun, $M_\odot$, is 1.
- Astronomical Unit (AU) is the distance between the centres of the Sun and Earth. The (average) distance between Jupiter and the Sun becomes 5.2AU.[8]
- Jupiter's semi-major axis is 5.204AU while the average is roughly 5.2AU. We may therefore approximate the orbit of Jupiter to be circular.[3]
- Mass of Jupiter is roughly $0.001 M_\odot$.[8] Other mass ratios are discussed in section 5.4.

2.3 Orbits about off-axis Lagrange Points

There are several characteristic orbits about the off-axis Lagrange points in the rotating frame. These include the tadpole orbit, curved tadpole orbit, horseshoe orbit and passing orbit.[7] In figures 5 and 6, tadpole orbits and horseshoe orbits (respectively) are shown.

3 Computational Analysis

3.1 Scaling and Performance

The plots for asteroid wander as a function of x-y offset and velocity offset took the longest time to compute. The method was to introduce many asteroids, each with different initial conditions, and solve the ODEs separately for each. For 10^4 asteroids, the computation took roughly 30 minutes for 1000 time steps per asteroid. The complexity scales as $\mathcal{O}(nt)$, where n is the number of asteroids and t is the number of time steps.

3.2 Integrator Choices

The ODEs to be solved were as follows, where superscript denotes Jupiter, Sun and Asteroid while subscript denotes Cartesian direction in x-y plane.

$$\frac{d}{dt}\begin{pmatrix} x^{(J)} \\ v_x^{(J)} \\ y^{(J)} \\ v_y^{(J)} \\ x^{(S)} \\ v_x^{(S)} \\ y^{(S)} \\ v_y^{(S)} \\ x^{(A)} \\ v_x^{(A)} \\ y^{(A)} \\ v_y^{(A)} \end{pmatrix} = \begin{pmatrix} v_x^{(J)} \\ a_x^{(J)} \\ v_y^{(J)} \\ a_y^{(J)} \\ v_x^{(S)} \\ a_x^{(S)} \\ v_y^{(S)} \\ a_y^{(S)} \\ v_x^{(A)} \\ a_x^{(A)} \\ v_y^{(A)} \\ a_y^{(A)} \end{pmatrix} \tag{1}$$

where the Newtonian laws for gravity were used to determine the accelerations of each body, neglecting the mass of the asteroid.

Several numerical integrator methods were compared in order to determine which was the most stable over long periods. The total energy of the system was calculated for each integrator and plotted as a function of time. Figure 3 shows the data obtained over roughly 2000 periods.

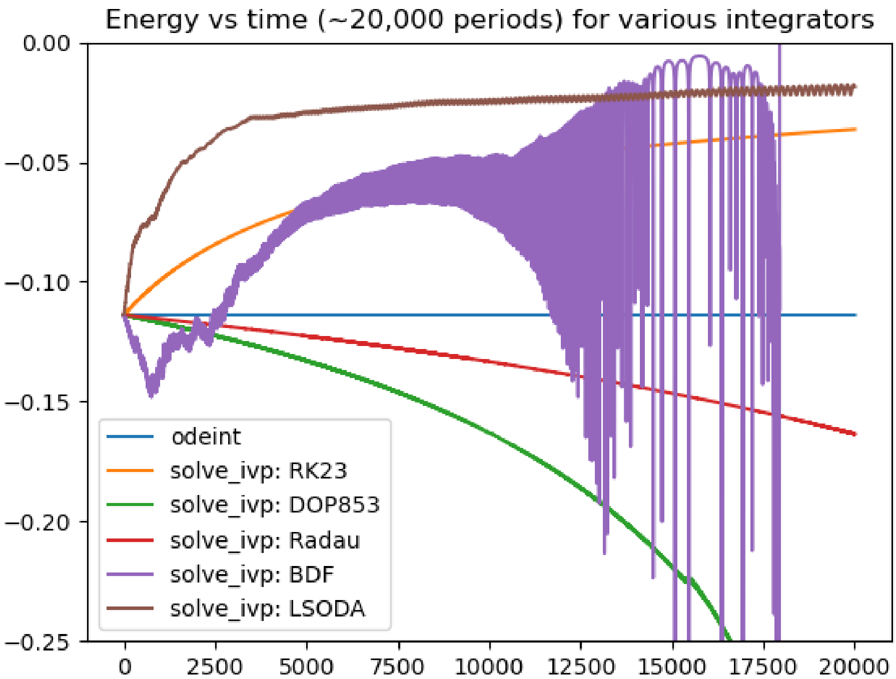


Figure 3: Comparison of long term behaviour for each integrator method used. Energy should be constant because there are no damping or resistant forces. All deviation from the initial value is due to instability of the integrator.

The `solve_ivp` methods all deviate to varying extents, while the `odeint` method proved to be most stable over a long time. The method `solve_ivp` RK45 failed to produce data for this time scale, automatically setting the number of evaluated times to be 70 instead of the intended 10,000. This was possibly due to the computational burden since this method calculates terms to a demanding fifth order.

The best performing method, `odeint`, uses LSODA from the FORTRAN library. This has the advantage of automatically dealing with stiff or non-stiff ODEs by changing its method. The solver begins using a non-stiff method but monitors data to calculate whether it should switch to a stiff solver method.[2]

- Stiff ODEs have a small step size in relation to the time interval used, taking an unfeasible amount of time to compute the integral. Stiff solvers do more work per step but take much larger steps, completing the integral in a shorter time and to higher accuracy.

- Non-stiff ODEs contain a reasonable number of steps between the integration bounds.

The integrators `RK45`, `RK23`, `DOP853` are intended for non-stiff ODEs, while `Radau` and `BDF` are used for stiff ODEs. `LSODA` uses the same method as `odeint`. It is unknown why the latter two give such different results in this instance.

The integrator being stable for long times is crucial to the validity of the results especially when investigating the stability of orbits over long time scales (see section 5.4). The numerical integrating method selected for the remainder of the project was `scipy.integrate.odeint`.

4 Code Implementation

4.1 Frame of Reference for ODE Solver

The equations of motion in the rotating frame are much more complicated than in the non-rotating CoM frame. The equations were solved in the latter frame. The positions of each object were subsequently transformed into the rotating frame.

`perform_rotation_ode()` transforms the data from the stationary frame to the rotating frame. The method used incorporates the rotation matrix

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \text{ where } \theta = \omega t$$

The reliability of this matrix method was tested using:

1. Stationary planet in rest frame transforming to a circle in the rotated frame.

2. Rotating planet in the rest frame transforming to a stationary planet in the rotated frame.

Obtaining the time period of the orbit was necessary to calculate the correct angular speed of the rotating frame. This value was tested by ensuring two functions, outputting a theoretical calculation of the time period and a manually measured time period, were equal.

4.2 Abstracting Code

From (1), the ODE solver input and output was of length 12 (one for each component of each object's position and velocity). This would have required a lot of confusing indexing. It was preferable to construct classes to improve the readability of the code.

`ODE(ode)` class: This class takes the integrator solution as an input and allows the correct index to be called via an intuitive attribute. For example, the 0^{th} and 2^{nd} item in the solution list correspond to the x and y components of Jupiter's position. These values were collected in an array of size 2 and labelled `ODE.r_j` in the attributes of the `ODE` class. This position vector could then be recalled easily as `ODE.r_j`.

`Conditions(conditions)` class: This class does exactly the same as **`ODE(ode)`**, except taking conditions as an input (before the equations have gone through the ODE solver).

`Point(r)` **class:** This class takes a 2-vector as an input and labels its 0^{th} component 'x' and its 1^{st} component 'y'. This class was called within the `ODE` class, meaning variables could be called like so: `ODE.r_j.x`, meaning the x component of Jupiter's position.

This made the code more human-readable and improved code-writing efficiency.

4.3 Storing Data

Each time data was computed, it would be stored to text files using numpy methods. Making plots without computing data each time greatly improved efficiency.

5 Results and Discussions

5.1 Orbits of Asteroids

The objective of this section is to investigate the stability of orbits about off-axis Lagrange points.

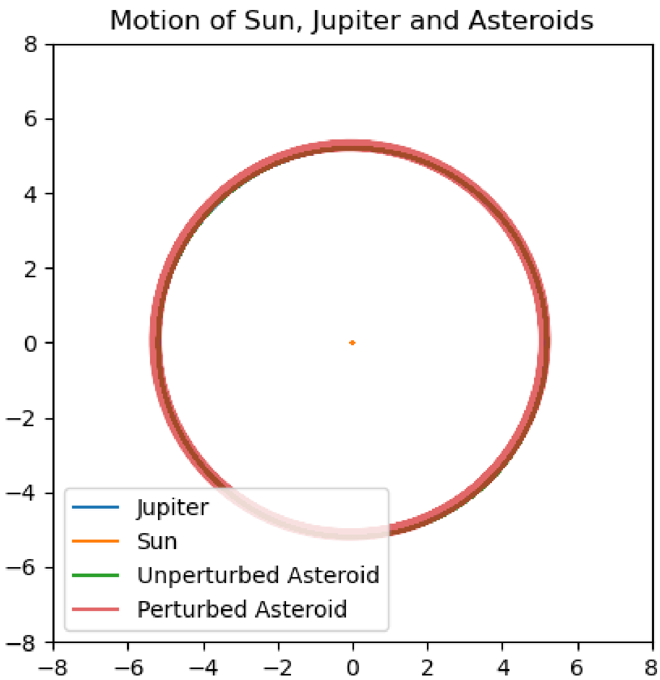


Figure 4: The orbits of the three bodies are plotted in their plane of motion over a few hundred orbits. The asteroid orbits about the centre of mass with the same period as Jupiter and the Sun. Its radius from the CoM varies with each orbit as it was initially displaced from the Lagrange point.

Figure 4 shows the motion of the asteroids in the stationary frame. The perturbed asteroid was initially displaced by a 'small amount' from the Lagrange point. The values of 0.01AU in the y direction and 0AU in the x-direction were selected. Through some experimentation, these values were deemed small enough to provide stable orbit about the L4 whilst also providing a visible wander. It is difficult to see whether the asteroid has remained tethered to L4 in the non-rotating frame. Figure 5 shows a more revealing plot.

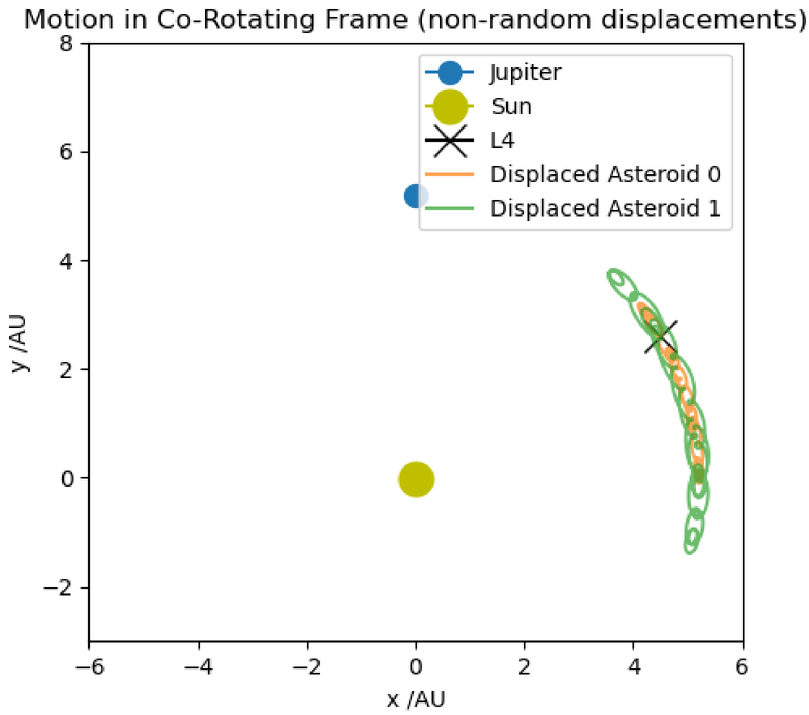


Figure 5: The rotational frame of reference shows the displaced asteroid orbiting about its local Lagrange point. This figure shows a few hundred orbital periods of motion.

Figure 5 shows two asteroids' motions in the rotating frame of reference. Their traces are characteristic of 'curved tadpole' orbits. Figure 6 shows the characteristic 'Horseshoe Orbit'. With the same Sun-Jupiter system but different initial conditions, this orbit travels past two other Lagrange points (L3 and L5) in the rotating frame. It is still tethered to L4 so is considered a stable orbit. The Horseshoe Orbit is more likely to be seen at smaller mass ratios since the effective potential allows objects to remain in stable oscillations in a more extended shape around the Sun (see figure 14).

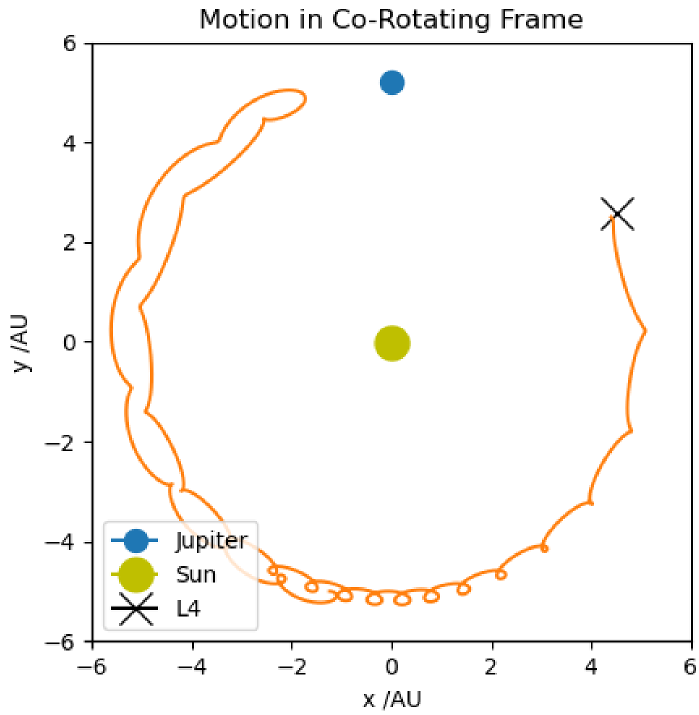


Figure 6: 'Horse-shoe' orbit. Motion of initially displaced asteroid in rotating frame of reference.

Overall, it is clear that there exist stable orbits around L4 and L5 of the Sun-Jupiter system, for small initial displacements from the equilibrium position. In the following section, the stability of these orbits will be investigated in more quantitative detail.

5.2 Stability of Lagrange Point

To investigate the stability of orbit about L4 or L5, both velocity and initial displacement must be considered. This was particularly difficult since both velocity and position include a direction and there was very little symmetry about L4. This meant there were very many combinations of position and velocity to explore. One significant simplification was to investigate the displacement and velocity independently, and only in the x-y plane.

5.2.1 Initial Displacement from the Lagrange Point

To investigate stability over initial displacement in the x-y plane, several asteroids were released from different initial positions in the vicinity of L4. Figure 8 shows the wander of these asteroids in their subsequent motion. Figure 7 shows the grid of initial displacements, overlaid on the effective potential plot. The grid extends 0.3AU (6% of the distance between L4 and the CoM) either side of L4 in the x and y directions. The expected result is that the asteroids starting closest to L4 in the shape outlined by the contours would remain in stable orbit. However, the asteroids further from L4 would stray far away from their initial position.

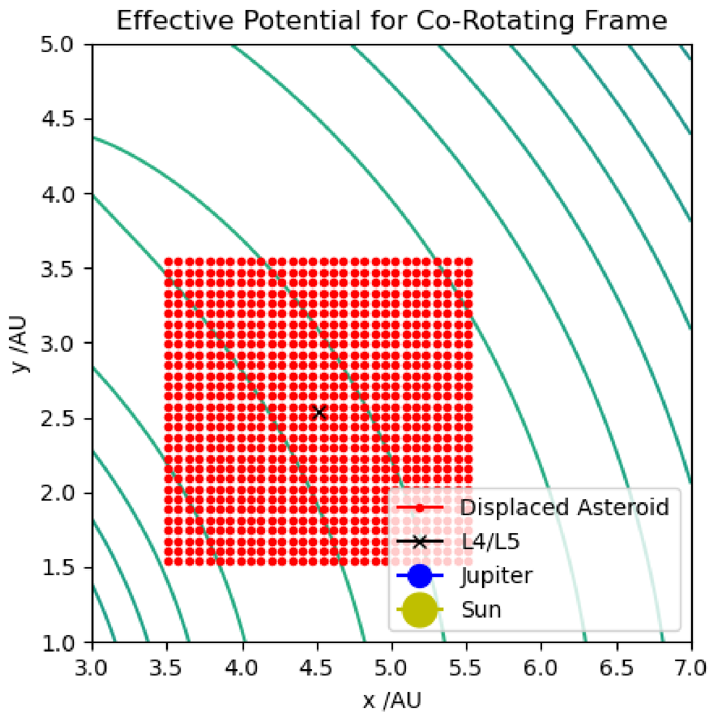
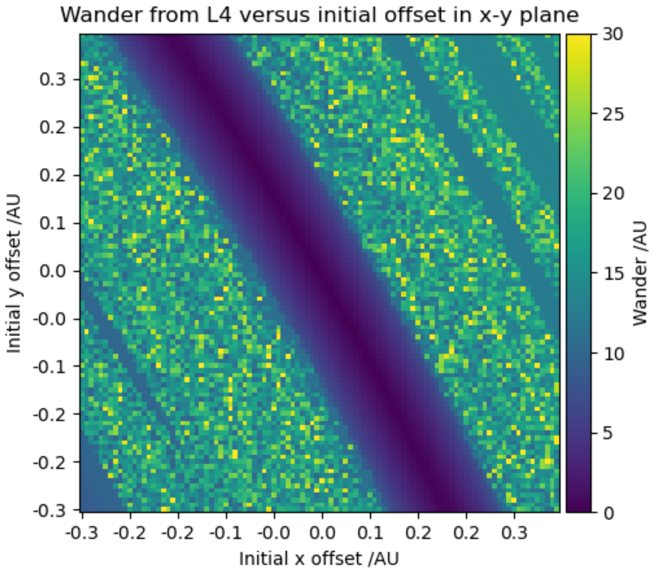
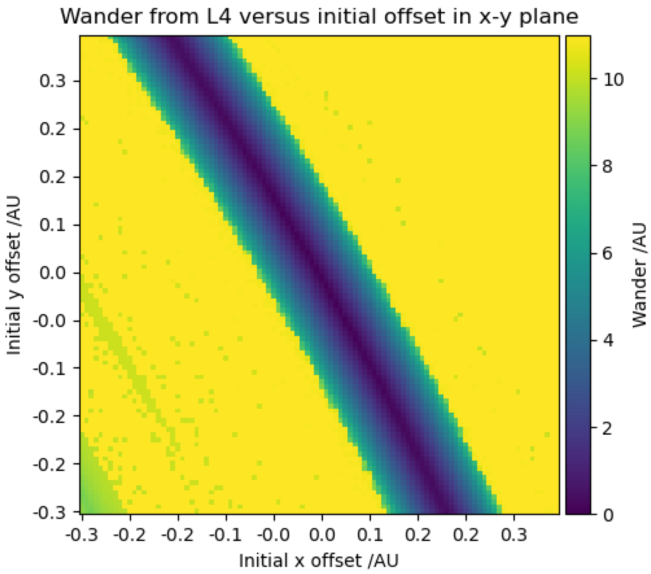


Figure 7: Initial 'grid' arrangement of asteroids overlaid on effective potential plot. Some of the asteroids will not orbit about L4 as they start too far away from the stationary point. These asteroids will wander far away in comparison to the stable orbits which remain in the vicinity of L4.



(a) Wander of asteroid from L4 given different initial displacements in the x-y plane. The maximum wander is 30AU, much larger than the orbital radius. This means yellow points are orbits which stray completely from the Sun as well as L4.

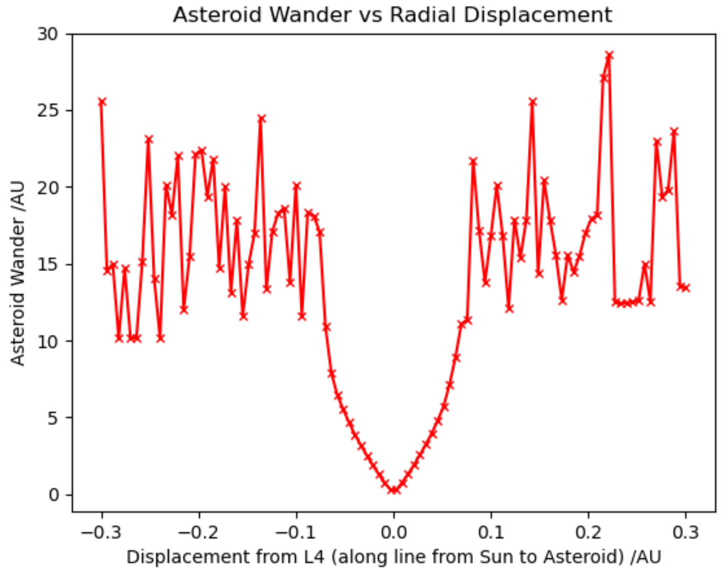


(b) Wander of asteroid from L4 given different initial displacements in the x-y plane. The maximum wander is 11AU. This implies the yellow regions are unstable orbits about L4, but may remain tethered to the Sun-Jupiter system.

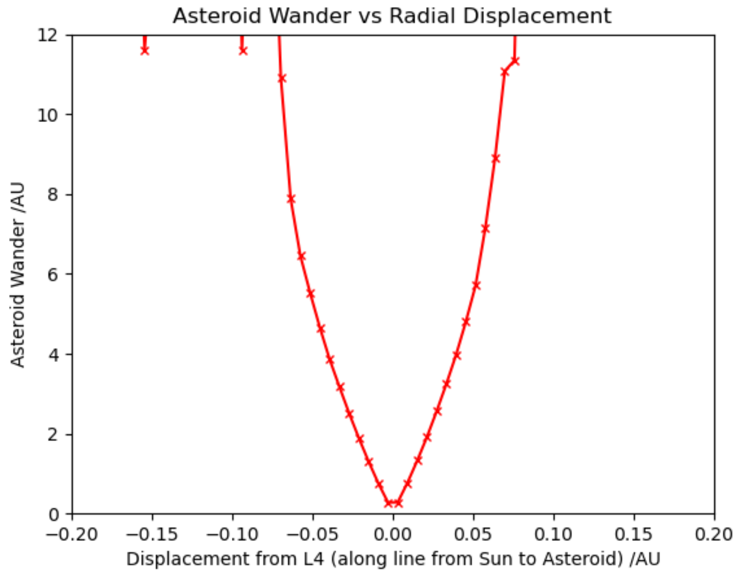
Figure 8: Heat maps showing wander of asteroids from L4 given initial displacement offsets.

Figure 8(a) and 8(b) show, as expected, that the orbits are most stable when displaced along the tangential direction. Figure 8(a) includes wander up to a limit of 30AU, which are represented by yellow dots. These asteroids will have escaped the Sun-Jupiter system entirely, whereas the turquoise color bands (outside the central dark blue band) represent wanders of roughly 15AU. These orbits, although they escape the stationary point around L4, remain within the Sun-Jupiter system.

Figure 8(b) is capped at 11AU. 11AU is chosen as this is roughly the maximum wander for a pseudo-stable horse-shoe orbit (see section 5.4 on Long-Term Stability). We can see that the fastest descent in stability is in the radial direction. Figure 9 shows the asteroids' wander as they are offset in the radial direction. From this, we may estimate a quantitative value for the shortest initial displacement which may lead to unstable orbits.



(a) Wander from L4 as a function of initial displacement along the line joining Sun to Asteroid.



(b) Wander from L4 as a function of initial displacement along the line joining Sun to Asteroid, capped at 11 AU.

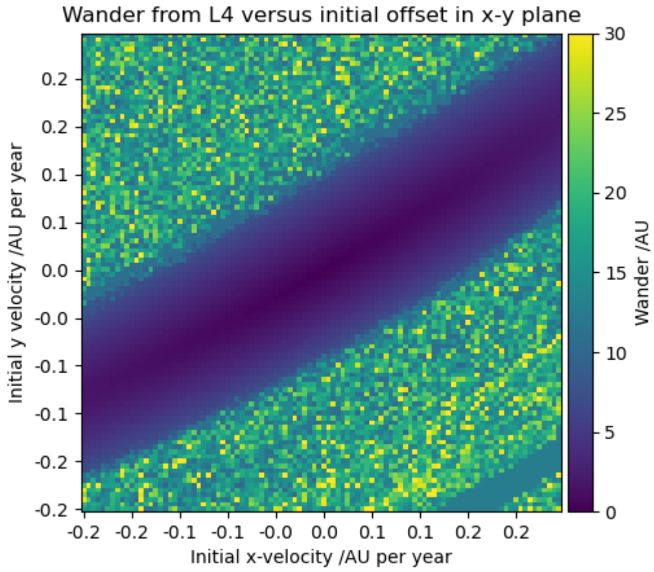
Figure 9: Plots showing wander of asteroid from L4 given initial radial displacement along the line from Sun to Asteroid. For small displacements, the wander of asteroids appears to be approximately quadratic with displacement.

From figure 9(b), an estimate of the furthest radial offset for a stable orbit is 0.08 ± 0.01AU in the positive radial direction and 0.07 ± 0.01AU in the negative radial direction. The asymmetry is expected here as the shape of the effective potential about L4 has very little symmetry itself.

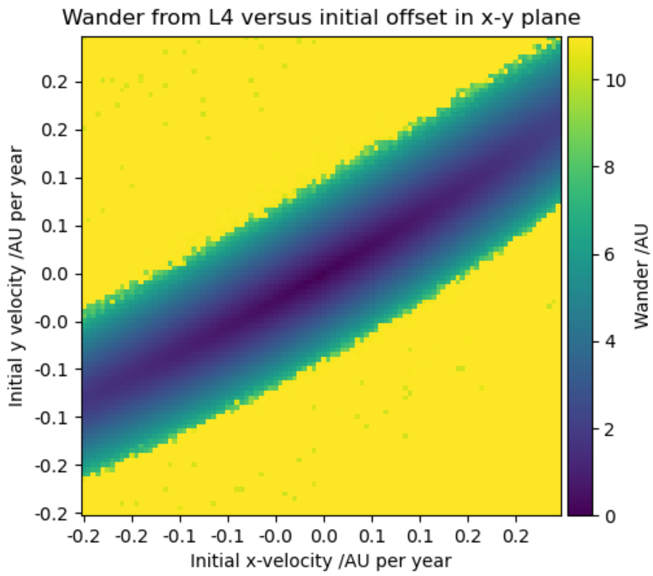
In conclusion, any displacement within the x-y plane with a magnitude less than 0.7AU is likely to remain in stable orbit about L4 for this particular mass ratio representing the Sun-Jupiter system.

5.3 Velocity Shift in X-Y Plane

To investigate the effect of changing initial velocity, the initial positions of the asteroids were held at L4. Small velocities (up to 5% of total equilibrium velocity in magnitude) were added vectorially to equilibrium velocity (defined as the velocity needed to keep the asteroid perfectly at the Lagrange point in the non-rotating frame). The same heat map plots showing different asteroids' wander from L4 are seen in figure 10.



(a) Wander from L4 as a function of **velocity** offset in the x-y plane, capped at 30 AU. The yellow squares are initial velocity offsets which give rise to orbits that escape the Sun as well as L4.

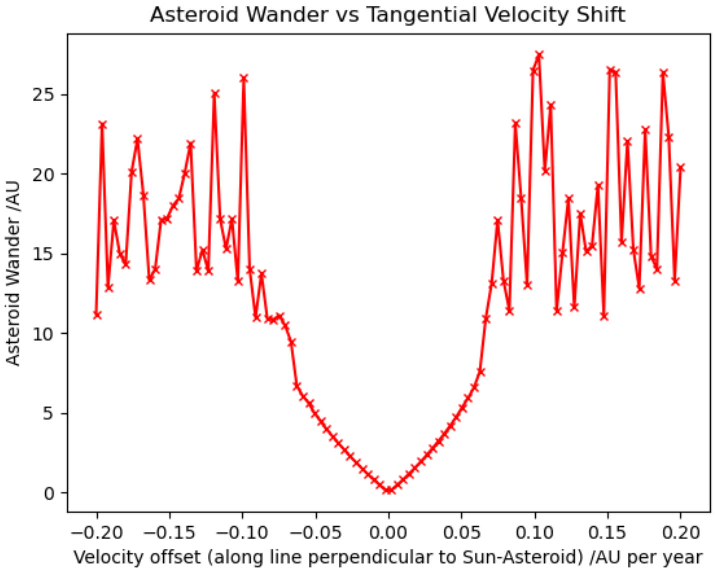


(b) Wander from L4 as a function of **velocity** offset in the x-y plane, capped at 11 AU. The yellow region shows initial velocity offsets which give rise to unstable orbits about L4.

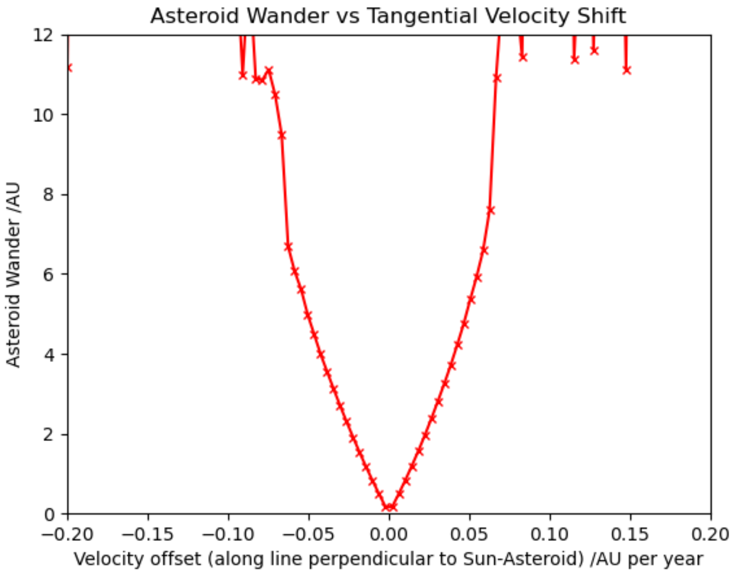
Figure 10: Heat maps showing wander of asteroids from L4 given initial velocity offsets.

The heat maps show that the maximum instability is achieved when applying additional velocity in the tangential direction. This was surprising as the additional velocity was expected to throw the asteroid into a horse-shoe orbit instead of a completely unstable one. Once again, figure 10(a) shows a large range of wanders up to 30AU, while figure 10(b) shows a smaller range only up to 11AU to highlight which asteroids (inside the dark blue band) remain stable about L4.

Since the direction of maximum instability was the tangential direction, this was investigated in further detail to obtain quantitative results.



(a) Wander from L4 as a function of initial velocity offset along the line perpendicular to that joining Sun to Asteroid.



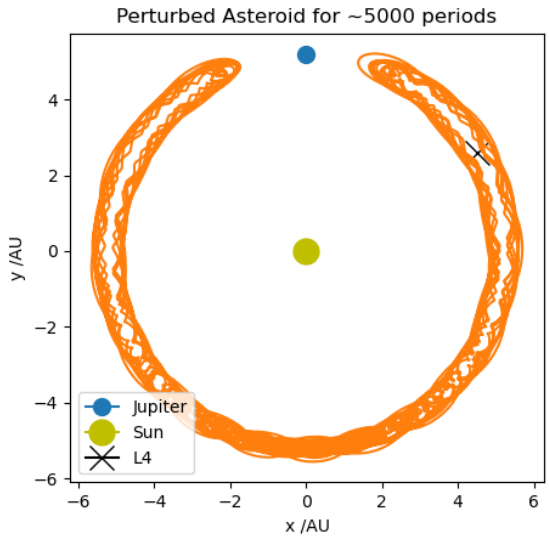
(b) Wander from L4 as a function of initial velocity offset along the line perpendicular to that joining Sun to Asteroid, capped at 11 AU. For small velocity offsets, the asteroids' wander appears again to be approximately quadratic with velocity.

Figure 11: Plots showing wander of asteroids from L4 given additional velocity perpendicular to the line from Sun to Asteroid (in the tangential direction).

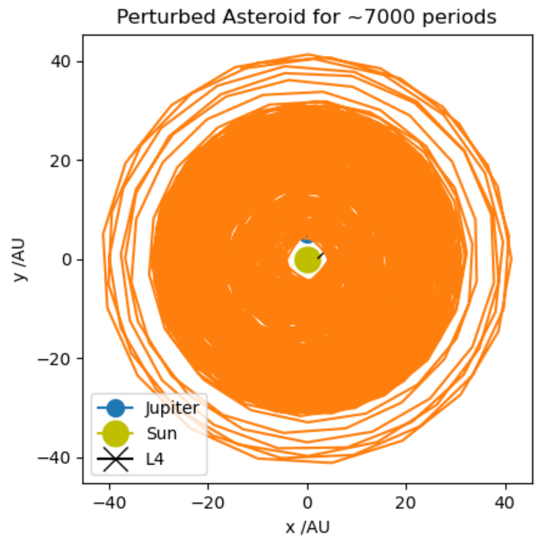
The modulus of the velocity at which the asteroids become unstable about L4 is (0.08 ± 0.01)AU/year in the $\overrightarrow{e_\theta}$ direction (pointing anti-clockwise from the positive x-axis) which corresponds to motion against the rotation of the frame. In the motion *with* the rotation of the frame, the maximum stable velocity is (0.07 ± 0.01)AU/year.

5.4 Long Term Stability

Can orbits which initially appear stable eventually become unstable and stray from their local Lagrange point? This was the question to be investigated in this section.



(a) Plot of Horseshoe orbit over roughly 5000 periods. This orbit is still stable about its Lagrange point.



(b) Plot of Horseshoe orbit over roughly 7000 periods. This orbit is now unstable about its Lagrange point.

Figure 12: Plots of Horseshoe orbits for different time scales, showing the orbit stray from L4 after roughly 7000 time periods.

After roughly 5000 periods, the horseshoe orbit was still stable. It was only after 7000 periods that the asteroid began to wander from its Lagrange point and even escape the orbit of the sun. The ODE solver used has been shown to give the correct energy even at 20,000 periods (see figure 3), making this a reliable source for data. It appears that the horse-shoe orbit is stable only pseudo-stable, over a given timescale depending on its initial position. This dependence on initial position is apparent from the fact that curved tadpole orbits are stable for a much longer time scale.

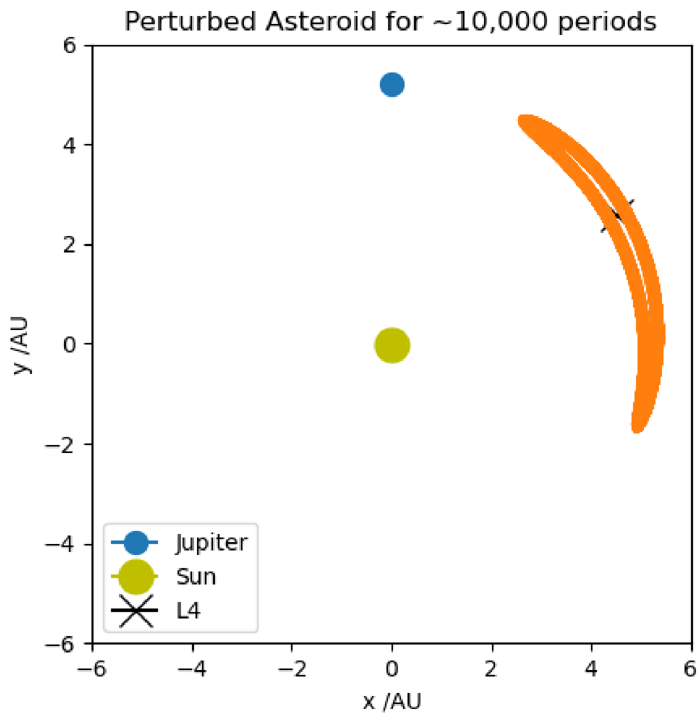


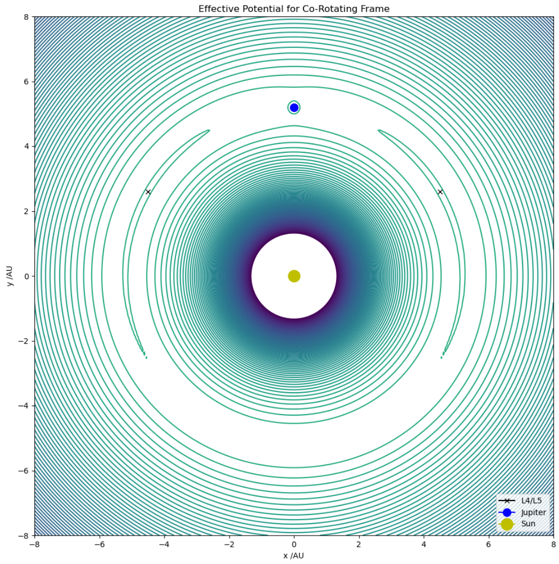
Figure 13: Plot of curved tadpole orbit over 10,000 periods. This orbit is still stable about its Lagrange point.

5.5 Mass Ratio of Bodies

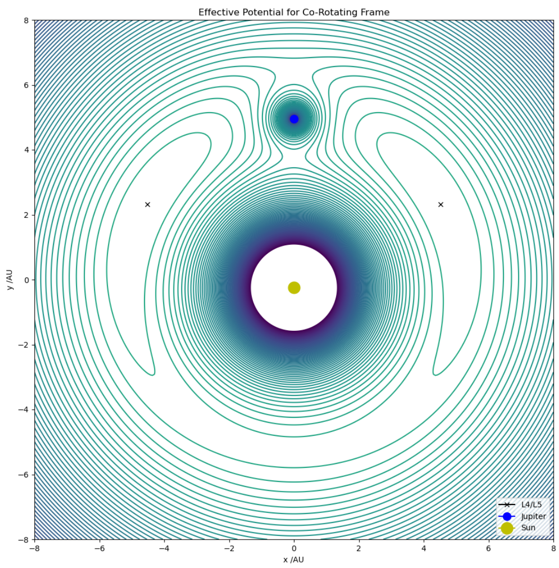
It can be shown that there exists an upper bound on the mass ratio, $\gamma = \frac{M_1}{M_1+M_2}$, where $M_1 < M_2$ by assumption.

$$\gamma < 0.0385... \rightarrow \frac{M_1}{M_2} < 0.400...$$

The aim of this section is to compute the critical mass ratio and obtain a relationship for wander in terms of mass ratio for stable stationary points.



(a) Effective potential for mass ratio 0.001



(b) Effective potential for mass ratio 0.05

Figure 14: Comparison of the Effective Potential contour plot for different mass ratios.

As $\frac{M_1}{M_2}$ changes, the shape of the effective potential changes as seen in figure 14. In the limit of small $\frac{M_1}{M_2}$, the total centre of mass tends towards the CoM of M_2 and the effective potential contours tend towards circles around M_2. At this point, the entire orbit of radius R defines a 'Lagrange Point' since there is zero attraction towards the smaller mass; it becomes a one-body problem. As $\frac{M_1}{M_2}$ increases, the 'kidney-bean shaped' effective potential contours at L4 and L5 become more fat and rounded.

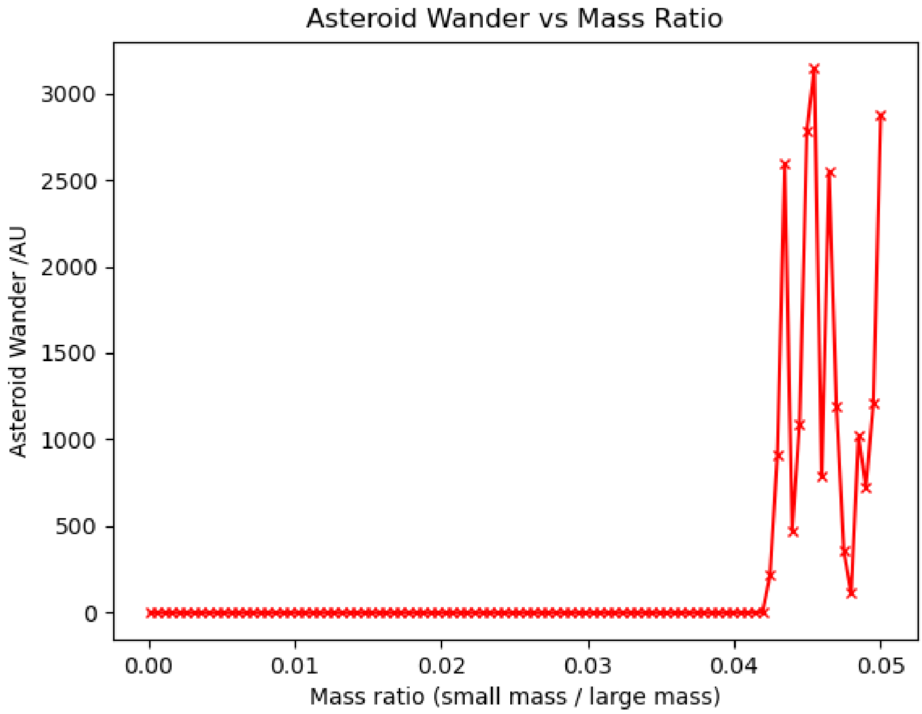


Figure 15: A slightly displaced (1×10^{-3}AU in x direction) orbit was allowed to run for roughly 2000 periods. The wander from L4 is plotted against the mass ratio for that system.

We can see that the critical mass ratio, $\frac{M_1}{M_2}$, where $M_1 < M_2$ by assumption, appears to be 0.042$\pm$0.001, which is within 5% of the theoretical value.

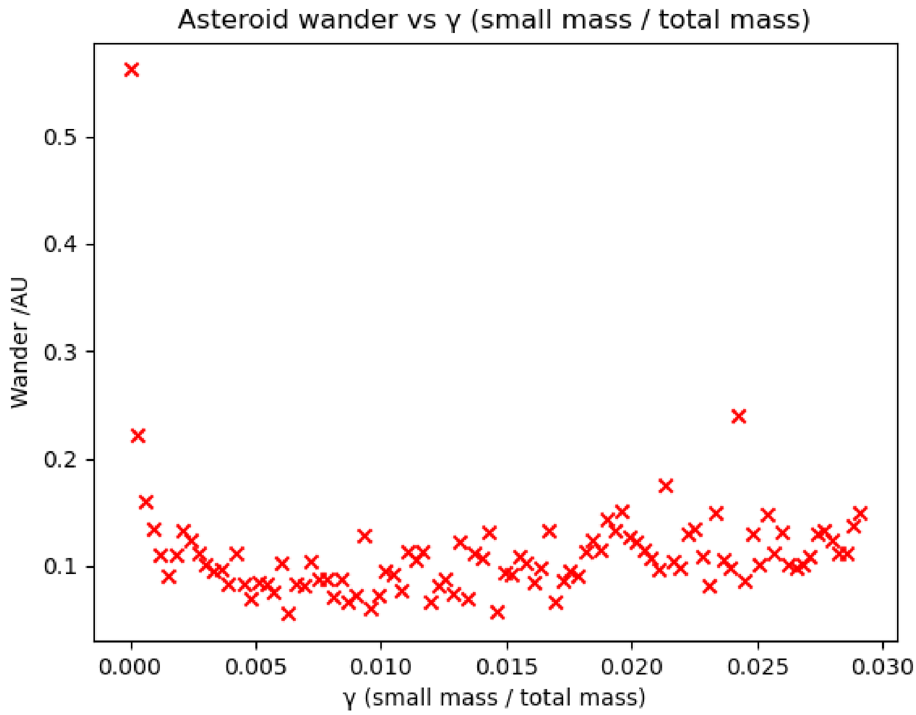
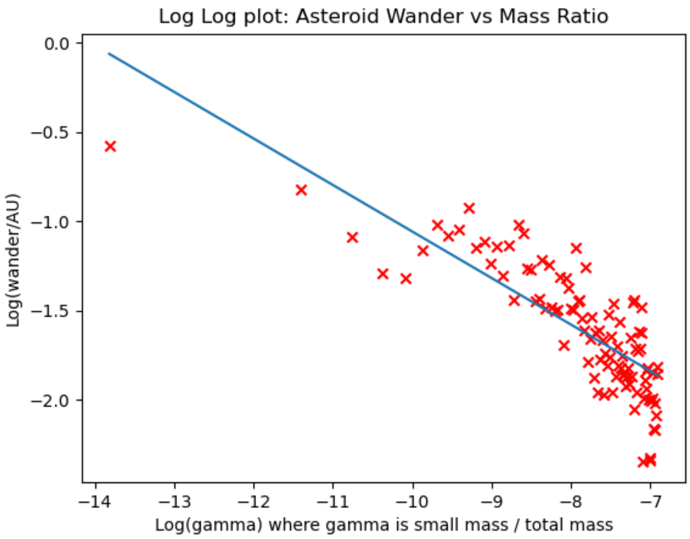
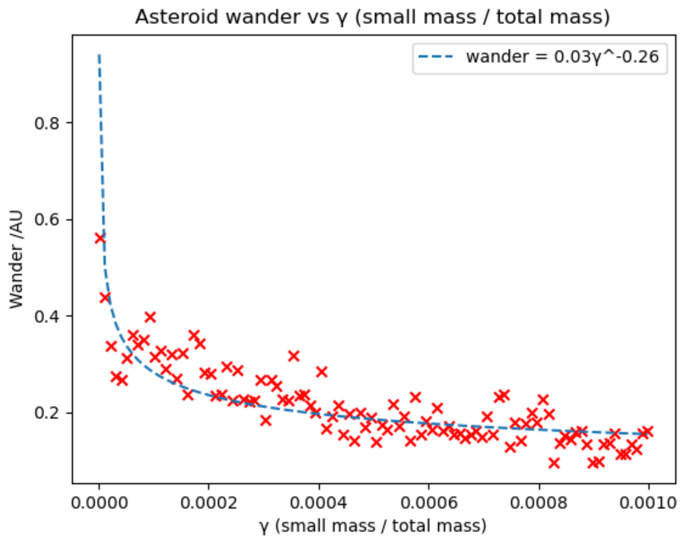


Figure 16: A *very* slightly displaced (1×10^{-4}AU in x direction) orbit was allowed to run for roughly 2000 periods. The wander from L4 is plotted against γ. Note the scale on the y-axis is much smaller than that of figure 15.

From figure 16, there is little evidence for a clear relationship between γ and the displaced asteroid's wander. We can however more reliably estimate a relationship for smaller values of γ.



(a) Plot of log(Wander) versus log(γ) for a *very* slight initial displacement (1×10^{-4}AU) from L4.



(b) A *very* slightly displaced (1×10^{-4}AU in x direction) orbit was allowed to run for roughly 2000 periods. The wander from L4 is plotted against γ for that system.

Figure 17: Plots to obtain a quantitative relationship between wander of asteroid and mass ratio.

The fitted curve was found in the following way:

1. Assume dependence of wander, W, on $\gamma = \frac{m}{m+M_\odot}$ as $W = A\gamma^n$
2. Plot a log-log graph $\log(W) = n\log(\gamma) + \log(A)$
3. Use numpy methods to find the best fit line for this linear equation, and substitute the values back into the original plot of $W = A\gamma^n$.

The results achieved here are limited in their reliability. Firstly, the assumed form of the asteroids' wander may be incorrect. Secondly, the log-log plot shows data spread relatively far from the regression line suggesting there is large error in the values of A and n.

Overall, $W = 0.3\gamma^{-0.26}$ was found to be the relationship between W and γ for small γ values. The critical mass ratio, $\frac{M_1}{M_2}$, was found to be 0.042 ± 0.001 which falls within 5% of the theoretical prediction[5, 6].

6 Conclusions

Overall, there were found to be stable stationary points, L4 and L5, in rotating two-body systems for mass ratios less than 0.042 ± 0.001. The maximum displacement for Trojan or Greek Asteroids to remain in stable orbit about L4 or L5 was found to be (0.07 ± 0.01)AU in the worst-case-scenario of radial displacement. Similarly, the corresponding additional velocity an asteroid could be given was found to be (0.07 ± 0.01)AU/year in the least stable tangential direction.

Some orbits were found to be pseudo-stable, particularly horseshoe orbits, which wandered from their Lagrange points after roughly 7000 periods. Finally, the wander, W, of a slightly deviated orbit was related to the gamma-ratio $\gamma = \frac{M_1}{M_1+M_2}$ of the two bodies by the following expression.

$$W = 0.3\gamma^{-0.26}$$

Some orbits are found to escape their local Lagrange point but remain within the Sun-Jupiter system while others entirely escape from the system. There was little trend in predicting which orbits would follow which path once they had escaped the Lagrange point.

References

[1] “James Webb Space Telescope lifts off on historic mission”. BBC News. Retrieved 25-12-2021.

[2] L. Petzold, “Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations”, SIAM Journal on Scientific and Statistical Computing, Vol. 4, No. 1, pp. 136-148, 1983.

[3] Dr. David R. Williams, “Jupiter Fact Sheet”, NASA. Last updated 23-12-2021.

[4] Dr. Drang, “Lagrange Point Redux”, `https://leancrew.com/all-this/2016/08/lagrange-points-redux/`, August 17, 2016

[5] B. Sicardy. Stability of the triangular Lagrange points beyond Gascheau’s value. Celestial Mechanics and Dynamical Astronomy, Springer Verlag, 2010, 107 (1-2), pp.145-155. ff10.1007/s10569-010-9259-5ff. ffhal-00552502

[6] Kemp, Sean, "An Examination of the Mass Limit for Stability at the Triangular Lagrange Points for a ThreeBody System and a Special Case of the Four-Body Problem" (2015). Master’s Theses. 4546. DOI: https://doi.org/10.31979/etd.4tf8-hnqx

[7] Alexandre G. Van Anderlecht, “Tadpole Orbits in the L4/L5 Region: Construction and Links to Other Families of Periodic Orbits”, Purdue University, May 2016.

[8] Buscher, David, “Computational Physics”, Part II Physics, Cavendish Laboratory, 2021-22.

A Mathematical Derivations

A.1 Derivation of Lagrange Points

This section contains a proof for location of Lagrange points L4 and L5.

$$\overrightarrow{g_{\text{stat}}} = -\frac{GM_\odot(\vec{r} - \vec{R_s})}{|\vec{r} - \vec{R_s}|^3} - \frac{GM_j(\vec{r} - \vec{R_j})}{|\vec{r} - \vec{R_j}|^3}$$

In the rotating frame, there are an additional two contributions to the field at any point: the second term is the Coriolis force and the third is the Centrifugal force. With $\vec{\Omega} = \Omega\vec{e_z}$,

$$\overrightarrow{g_{\text{rot}}} = \overrightarrow{g_{\text{stat}}} + 2\vec{\Omega} \times \vec{v} - \vec{\Omega} \times (\vec{\Omega} \times \vec{r})$$

Solving for $\overrightarrow{g_{\text{rot}}} = 0$, letting $\vec{v} = 0$, and noting that $r_z = 0$,

$$\vec{0} = \overrightarrow{g_{\text{stat}}} - \vec{\Omega} \times (\vec{\Omega} \times \vec{r})$$

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} g_x^{\text{stat}} + \Omega^2 r_x \\ g_y^{\text{stat}} + \Omega^2 r_y \\ g_z^{\text{stat}} \end{pmatrix} \tag{2}$$

We shall plug in the solution as an ansatz for simplicity:

$$\vec{r} = \begin{pmatrix} \frac{R}{2} \\ \pm\frac{\sqrt{3}}{2}R \\ 0 \end{pmatrix}$$

where $R = R_j + R_s$, $\vec{r_j} = R_j\vec{e_y}$, and $\vec{r_s} = -R_s\vec{e_y}$
This ansatz satisfies (2) if

$$\Omega = \sqrt{\frac{G(M_\odot + M_j)}{R^3}}$$

A.2 Mass Ratio

Gascheau showed that orbits about L4/L5 are stable up to a given mass ratio $\mu = \frac{M_1}{M_1+M_2}$. [4]

$$\mu_G = \frac{1}{2}\left(1 - \sqrt{\frac{23}{27}}\right) \approx 0.0385$$

This is the approximate solution to the inequality

$$\frac{(M_1 + M_2 + m)^2}{M_1 M_2 + m M_1 + m M_3} \leq 27$$

where we set m, the mass of the asteroid in this case, to be approximately zero.

$$\frac{(M_1 + M_2)^2}{M_1 M_2} \leq 27$$

$$\frac{1}{\mu(1-\mu)} \leq 27$$

This yields a quadratic with critical points

$$27\mu^2 - 27\mu + 1 = 0$$

$$\mu_{\pm} = \frac{1}{2}\left(1 \pm \sqrt{\frac{23}{27}}\right)$$

The positive root can be disregarded since we assumed $\mu < 1$.

B Source Code

trojan.py

```python
'''
Main file with all functions and classes
'''

import numpy as np
import scipy
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
from scipy.integrate import odeint
from mpl_toolkits.axes_grid1 import make_axes_locatable
import math
import random
import matplotlib

'''
Setting up parameters
'''

G = 4*math.pi**2 # value of Gravitational Constant in solar system units
N = 10000 # number of evaluated points on the orbit
```

```
t_span = [0,2000] # time span of orbit
t = np.linspace(t_span[0], t_span[1], N)
asteroids = 100 # number of asteroids
m_j = 0.001 # in units of solar mass
m_s = 1 # in units of solar mass
R = 5.2 # distance from Jupiter to Sun, in AU
# max_delta = 1 # for grids and heat maps
max_delta = 0.3 # horseshoe orbit 0.4??
min_delta = 0 # horseshoe orbit (single_deltas used, not grid)
max_delta_v = 0.2
min_delta_v = 0
ms = np.linspace(0.000001, 0.03, 100)

def time_period(R, m_j, m_s):
    '''
    returns time period of an asteroid in equilibrium position
    (same T as Jupiter around Sun)
    '''
    T = ( (4 * math.pi**2 * R**3 )/ (G * (m_s + m_j)) ) ** (1/2)
    return T

T = time_period(R, m_j, m_s)



def get_ds(bottom, top, asteroids):
    return np.linspace(bottom, top, asteroids)

def get_deltas(asteroids):
    delta_xs = np.zeros(asteroids)
    delta_ys = np.linspace(min_delta, max_delta, asteroids, endpoint = True)
    deltas = np.array(list(zip(delta_xs, delta_ys)))
    return deltas

def get_delta_vs(asteroids):
    delta_vxs = np.linspace(min_delta, max_delta_v, asteroids, endpoint = True)
    delta_vys = np.zeros(asteroids)
    delta_vs = np.array(list(zip(delta_vxs, delta_vys)))
    return delta_vs

# Randomised deltas in the square with corners (-0.1, -0.1) and (0.1, 0.1)
# added to eqm positions to test stability of orbit
def get_rand_deltas(asteroids):
    rand_deltas = []
    for i in range(asteroids):
        rand_delta_x = max_delta/1000 * random.randint(-1000,
```

```
1000)
        rand_delta_y = min_delta/1000 * random.randint(-1000, 1000)
        rand_deltas.append([rand_delta_x, rand_delta_y])
    rand_deltas = np.array(rand_deltas)
    return rand_deltas

# Random (small) velocities, added to the eqm velocity to test stability of orbit
def get_rand_vels(asteroids):
    rand_vels = []
    for i in range(asteroids):
        rand_v_x = (1/10000) * random.randint(-1000,1000)
        rand_v_y = (1/10000) * random.randint(-1000,1000)
        rand_vels.append([rand_v_x, rand_v_y])
    rand_vels = np.array(rand_vels)
    return rand_vels



def circ_orbit_conditions_new(R, m_j, m_s, delta_r_a, v_a, delta_v_a, v_eqm = True, dv = False):
    '''
    returns initial conditions for all objects (J, S, Asteroid)
    to be in circular orbit about the CoM (which lies at the origin)
    if eqm is set to True, the function calculates the correct asteroid velocity to maintain circ orbit
    if eqm is set to False, the function takes the input rand_vel as the asteroid's initial velocity
    '''

    M = m_j + m_s # total mass
    gamma = m_j / M

    r_j = np.array([0, (1 - gamma) * R])
    r_s = np.array([0, - gamma * R])

    mu = (G * M / R) **(1/2)

    v_j = np.array([(1-gamma) * mu, 0])
    v_s = np.array([- gamma * mu, 0])


    X = R * (1 - gamma + gamma**2)**(1/2) # distance between origin and Lagrange point
    th = (1 + gamma) * math.pi / 3 # angle between line to Lagrange point and y axis

```

```
    T = time_period(R, m_j, m_s)

    # print(f"calculated time period is {T}")
    omega = 2*math.pi / T
    v = omega * X

    # asteroid conditions
    r_a_eqm = np.array([X * np.sin(th), X * np.cos(th)])
    r_a = r_a_eqm + delta_r_a

    # the following statements were constructed to allow the initial conditions to be called with varying values for velocity of asteroid
    # the default value (near equilibrium) is used when v_eqm == True and dv == False
    if v_eqm and not dv:
        v_a = np.array([v * np.cos(th) , - v * np.sin(th)])
    elif v_eqm and dv:
        v_a = np.array([v * np.cos(th) , - v * np.sin(th)]) + np.array(delta_v_a)
    elif v_eqm == False:
        v_a = np.array(v_a)
    else:
        print("logic not right!")

    conditions = format_conditions(r_j, v_j, r_s, v_s, r_a, v_a)

    return conditions



def format_conditions(r_j, v_j, r_s, v_s, r_a, v_a):
    '''
    puts readable data into necessary form for ODE solver
    '''
    return np.array([

        r_j[0], v_j[0], r_j[1], v_j[1],

        r_s[0], v_s[0], r_s[1], v_s[1],

        r_a[0], v_a[0], r_a[1], v_a[1],

    ])


class Point:
    '''
    Abstracting Class
```

```
    '''
    def __init__(self, r):
        self.x = r[0]
        self.y = r[1]

class ODE:
    '''
    Abstracting Class
    Input is solution to ode solver
    '''
    def __init__(self, ode):
        self.ode = ode

        self.r_j = Point([ode[:, 0], ode[:, 2]])
        self.v_j = Point([ode[:, 1], ode[:, 3]])

        self.r_s = Point([ode[:, 4], ode[:, 6]])
        self.v_s = Point([ode[:, 5], ode[:, 7]])

        self.r_a = Point([ode[:, 8], ode[:, 10]])
        self.v_a = Point([ode[:, 9], ode[:, 11]])

class Conditions:
    '''
    Abstracting Class
    Input is array in form returned by format_conditions()
    '''
    def __init__(self, conditions):
        self.conditions = conditions

        self.r_j = Point([conditions[0], conditions[2]])
        self.v_j = Point([conditions[1], conditions[3]])

        self.r_s = Point([conditions[4], conditions[6]])
        self.v_s = Point([conditions[5], conditions[7]])

        self.r_a = Point([conditions[8], conditions[10]])
        self.v_a = Point([conditions[9], conditions[11]])


class Two_Body_System:
    '''
    creates an orbit of Jupiter/Sun/Asteroid around the CoM of
    the system
    can solve the differential equations
    can find and plot the effective potential about the initial
    setup
    '''

```

```python
    def __init__(self, m_j, m_s):
        self.m_j = m_j
        self.m_s = m_s


    def general_orbit_conditions(self, r_j, v_j, r_s, v_s, r_a, v_a):
        '''
        test solver for any general orbit, not necessarily circular
        gives complete control for me to vary system as needed for testing
        '''

        conditions = format_conditions(r_j, v_j, r_s, v_s, r_a, v_a)

        return conditions


    def interact_ivp(self, t_span, t, method, delta_r_a, delta_v_a, v_eqm = True, dv = False):
        '''
        ODE solver with various methods, taken as an input
        Called in run_orbit.py and performances are tested in test_integrator.py
        '''

        y_0 = circ_orbit_conditions_new(R, self.m_j, self.m_s, delta_r_a, [0,0], delta_v_a, v_eqm = v_eqm, dv = dv)

        sol = scipy.integrate.solve_ivp(
            self.ODE,
            t_span,
            y_0,
            method=method,
            t_eval = t,
            )

        return sol


    def interact_ode(self, t_span, delta_r_a, v_a, delta_v_a, v_eqm, dv):
        '''
        ODE solver using method scipy.integrate.odeint
        This is the main solver used in the bulk of the project.
        '''

```

```python
        y_0 = circ_orbit_conditions_new(R, self.m_j, self.m_s,
    delta_r_a, v_a, delta_v_a, v_eqm, dv)

        sol = scipy.integrate.odeint(
            self.ODE,
            y_0,
            t,
            tfirst = True
        )

        return sol


    def ODE(self, t, y):

        # transform y into a class where its data is readable
        conditions = Conditions(y)

        r_j = np.array([conditions.r_j.x, conditions.r_j.y])
        v_j = np.array([conditions.v_j.x, conditions.v_j.y])

        r_s = np.array([conditions.r_s.x, conditions.r_s.y])
        v_s = np.array([conditions.v_s.x, conditions.v_s.y])

        r_a = np.array([conditions.r_a.x, conditions.r_a.y])
        v_a = np.array([conditions.v_a.x, conditions.v_a.y])

        # convention is dx_js is the x vector from Jupiter to the
    Sun, etc.
        dx_js = r_s[0] - r_j[0]
        dx_as = r_s[0] - r_a[0]
        dx_aj = r_j[0] - r_a[0]

        dy_js = r_s[1] - r_j[1]
        dy_as = r_s[1] - r_a[1]
        dy_aj = r_j[1] - r_a[1]


        accel_j = np.array([
            G * self.m_s * dx_js / (dx_js**2 + dy_js**2)**(3/2),
            G * self.m_s * dy_js / (dx_js**2 + dy_js**2)**(3/2)
        ])

        accel_s = np.array([
            - G * self.m_j * dx_js / (dx_js**2 + dy_js**2)**(3/2)
    ,
            - G * self.m_j * dy_js / (dx_js**2 + dy_js**2)**(3/2)
        ])

```

```
        accel_a_x = G * self.m_j * dx_aj / (dx_aj**2 + dy_aj**2)**(3/2) + G * self.m_s * dx_as / (dx_as**2 + dy_as**2)**(3/2)
        accel_a_y = G * self.m_j * dy_aj / (dx_aj**2 + dy_aj**2)**(3/2) + G * self.m_s * dy_as / (dx_as**2 + dy_as**2)**(3/2)

        accel_a = np.array([accel_a_x, accel_a_y])

        conditions = format_conditions(v_j, accel_j, v_s, accel_s, v_a, accel_a)

        return conditions


    def field(self, x, y):
        '''
        returns the effective potential for contour map
        '''

        conditions = circ_orbit_conditions_new(R, self.m_j, self.m_s, [0,0], [0,0], [0,0], v_eqm = True, dv = False)
        omega = 2 * math.pi / T
        conds = Conditions(conditions)

        r_j_array = np.array([conds.r_j.x, conds.r_j.y])
        r_s_array = np.array([conds.r_s.x, conds.r_s.y])

        r = (x**2 + y**2)**(1/2)
        delta_x_j = (x - conds.r_j.x)
        delta_y_j = (y - conds.r_j.y)
        delta_r_j = (delta_x_j**2 + delta_y_j**2)**(1/2)

        delta_x_s = (x - conds.r_s.x)
        delta_y_s = (y - conds.r_s.y)
        delta_r_s = (delta_x_s**2 + delta_y_s**2)**(1/2)

        U_j = - G * self.m_j / delta_r_j
        U_s = - G * self.m_s / delta_r_s
        U_rot = - 1/2 * r**2 * omega**2

        U_eff = np.array(U_rot) + np.array(U_j) + np.array(U_s)

        return U_eff


    def planar_plot(self, x_min, x_max, y_min, y_max, sols, unpert_ode):
        '''
        Plot contour map of effective potential in 2D space
        '''
```

```

        conditions = circ_orbit_conditions_new(R, self.m_j, self.m_s, [0,0], [0,0], [0,0], v_eqm = True, dv = False)
        conds = Conditions(conditions)

        x = np.linspace(x_min, x_max, 256)
        y = np.linspace(y_min, y_max, 256)
        X, Y = np.meshgrid(x, y)
        U_eff = self.field(X, Y)

        fig, ax = plt.subplots(figsize = (12,12))

        levels = np.arange(-30, 0, 0.2); # plot min U_eff, max U_eff, step in U_eff
        ax.contour(X, Y, U_eff, levels);
        ax.set_aspect('equal', adjustable='box');


        # # Use the following if the >1 displaced asteroids are to be plotted

        # for i, sol in enumerate(sols):
        #     r_primed_delta = perform_rotation_ode(unpert_ode, sol)
        #     initial_x = r_primed_delta[0][0]
        #     initial_y = r_primed_delta[1][0]
        #     if i == 0:
        #         ax.plot(initial_x, initial_y, marker = 'o', color = 'r', ms = 3, label = "Displaced Asteroid")
        #     else:
        #         ax.plot(initial_x, initial_y, marker = 'o', color = 'r', ms = 3)

        # plots Jupiter, Sun and L4/L5 for single asteroid
        ax.plot(conds.r_a.x, conds.r_a.y , marker="x", color='k');
        ax.plot(-conds.r_a.x, conds.r_a.y , marker="x", color='k', label = "L4/L5");
        ax.plot(conds.r_j.x, conds.r_j.y , marker="o", markersize=10, color='b', label = 'Jupiter');
        ax.plot(conds.r_s.x, conds.r_s.y , marker="o", markersize=15, color='y', label = 'Sun');
        ax.set_xlabel("x /AU");
        ax.set_ylabel("y /AU");
        ax.set_aspect('equal');
        ax.set_xlim(x_min, x_max)
        ax.set_ylim(y_min, y_max)
        ax.legend(loc='lower right');
        ax.set_title("Effective Potential for Co-Rotating Frame")
```

```python
;
    plt.savefig(f'figures/U_eff(1)')


def measure_period(ode):
    '''
    T/2 is when the x coordinate of Jupiter becomes negative
    To check reliability of calculated time_period() function
    '''

    index = np.where(ode.r_j.x < 0)[0][0] # first index that
    contains negative x value

    half_period = t[index]
    period = 2 * half_period
    # print(f"measured period is {period}")
    return period


def make_plots_ode(unpert_ode, odes):
    '''
    Plots motion of Jupiter, Sun and Asteroids in the non-
    rotating frame.
    '''

    fig, ax = plt.subplots()
    ax.plot(unpert_ode.r_j.x, unpert_ode.r_j.y, label = "Jupiter"
    )
    ax.plot(unpert_ode.r_s.x, unpert_ode.r_s.y, label = "Sun")
    ax.plot(unpert_ode.r_a.x, unpert_ode.r_a.y, label = "
    Unperturbed Asteroid")
    ax.set_xlim(-8,8)
    ax.set_ylim(-8,8)
    ax.set_aspect('equal')
    ax.plot(odes[1].r_a.x, odes[1].r_a.y, label = "Perturbed
    Asteroid", alpha = 0.7)
    ax.legend()
    ax.set_title("Motion of Sun, Jupiter and Asteroids")


def stationary_initial_position(unpert_ode, odes, xlabel, ylabel,
     title):
    '''
    Plots initial positions of each object in the system
    '''

    r_primed_unpert = perform_rotation_ode(unpert_ode, unpert_ode
    )
    unpert_x = r_primed_unpert[0][0]
```

```
    unpert_y = r_primed_unpert[1][0]

    fig, ax = plt.subplots()
    for i, ode in enumerate(odes):
        r_primed_delta = perform_rotation_ode(unpert_ode, ode)
        initial_x = r_primed_delta[0][0]
        initial_y = r_primed_delta[1][0]
        ax.plot(initial_x, initial_y, marker = 'o', ms = 3, color = 'r')

    ax.plot(unpert_x, unpert_y, marker = 'o', color = 'b', label = "Asteroid at L4")
    ax.plot(unpert_ode.r_j.x[0], unpert_ode.r_j.y[0], marker = 'x', label = "Jupiter")
    ax.plot(unpert_ode.r_s.x[0], unpert_ode.r_s.y[0], marker = 'x', label = "Sun")
    ax.set_xlim(-4, 4)
    ax.set_ylim(-2, 6)
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.legend()
    ax.set_title(title)


def angle(unpert_ode, t):
    '''
    Obtains angle called in perform_rotation_ode function
    '''

    omega = 2*math.pi / T
    thetas = np.zeros(len(t))

    for i, t in enumerate(t):
        thetas[i] = omega*t

    return thetas



def perform_rotation_ode(unpert_ode, ode, m = m_j):
    '''
    rotate the x-y frame at the same rate as the eqm asteroid
    returns two vectors in a tuple, each length N and containing x/y coordinates in the co-rotated frame
    '''
    period = time_period(R, m, m_s)
    omega = 2*math.pi / period

    t = np.linspace(t_span[0], t_span[1], N)
```

```
    thetas = np.zeros(len(t))

    for i, t in enumerate(t):
        thetas[i] = omega*t

    # thetas = angle(unpert_ode, t)

# for asteroid
    x_primed = np.cos(thetas) * ode.r_a.x - np.sin(thetas) * ode.
    r_a.y
    y_primed = np.sin(thetas) * ode.r_a.x + np.cos(thetas) * ode.
    r_a.y

# for jupiter
    # x_primed = np.cos(thetas) * ode.r_j.x - np.sin(thetas) *
    ode.r_j.y
    # y_primed = np.sin(thetas) * ode.r_j.x + np.cos(thetas) *
    ode.r_j.y

# for sun
    # x_primed = np.cos(thetas) * ode.r_s.x - np.sin(thetas) *
    ode.r_s.y
    # y_primed = np.sin(thetas) * ode.r_s.x + np.cos(thetas) *
    ode.r_s.y

    r_primed = np.array([x_primed, y_primed])

    return r_primed


def corotated_deviation(unpert_ode, ode, title, m = m_j):
    '''
    Plots the motion of 'perturbed' asteroids about L4 in the
    rotating frame
    '''

    r_primed_unpert = perform_rotation_ode(unpert_ode, unpert_ode
    , m = m)

    fig, ax = plt.subplots()
    # ax.plot(r_primed_unpert[0], r_primed_unpert[1], label = "
    Undisplaced Asteroid")
    ax.plot(unpert_ode.r_j.x[0], unpert_ode.r_j.y[0], marker = 'o
    ', ms = 10, label = "Jupiter")
    ax.plot(unpert_ode.r_s.x[0], unpert_ode.r_s.y[0], marker = 'o
    ', ms = 15, color = 'y', label = "Sun")
    ax.plot(unpert_ode.r_a.x[0], unpert_ode.r_a.y[0], marker = 'x
    ', ms = 15, color = 'k', label = "L4")

```

```python
    # for i, ode in enumerate(odes):
    #     r_primed_delta = perform_rotation_ode(unpert_ode, ode, m = m)
        # diff_xs[i] = r_primed_delta[0] - r_primed_unpert[0]
        # diff_ys[i] = r_primed_delta[1] - r_primed_unpert[1]
        # ax.plot(r_primed_delta[0], r_primed_delta[1], alpha = 1) #, label = f"delta_r = {float(round(np.linalg.norm(deltas[i]), 3))}" , delta_v = {float(round(np.linalg.norm(rand_vels[i]), 3))}")

    r_primed_delta = perform_rotation_ode(unpert_ode, ode, m = m)
    ax.plot(r_primed_delta[0], r_primed_delta[1])
    ax.set_xlim(-6,6)
    ax.set_ylim(-6,6)
    ax.set_xlabel('x /AU')
    ax.set_ylabel('y /AU')
    ax.set_aspect('equal')
    ax.legend(loc = 'lower left')
    ax.set_title(title)


def energy(ode, m_j, m_s):
    '''
    Gets total energy of masses Jupiter and the Sun to compare between ODE solvers in test_integrator.py
    '''

    v_j = (ode.v_j.x**2 + ode.v_j.y**2)**0.5
    v_s = (ode.v_s.x**2 + ode.v_s.y**2)**0.5
    r_js = ((ode.r_s.x - ode.r_j.x)**2 + (ode.r_s.y - ode.r_j.y)**2)**0.5
    T_j = (1/2) * m_j * v_j**2
    U_j = - (G * m_s * m_j) / r_js
    T_s = (1/2) * m_s * v_s**2
    U_s = - (G * m_s * m_j) / r_js

    E = T_j + U_j + T_s + U_s
    return E


def maximum_deviation(odes, initial_dxs, unpert_ode, direction, cap = False, graph = True):
    '''
    Finds maximum deviation in the rotating frame between 'perturbed' asteroid and L4 within its lifetime (set by t_span)
    This function is used when there is one axis of initial displacement, not a whole grid.
    '''
```

```
    max_mod_delta_rs = []

    for ode in odes:
        r_primed = perform_rotation_ode(unpert_ode, ode)
        r_primed_unpert = perform_rotation_ode(unpert_ode,
    unpert_ode)
        delta_r = r_primed - r_primed_unpert
        mod_delta_r = np.linalg.norm(delta_r, axis = 0)
        max_mod_delta_r = np.amax(mod_delta_r)
        max_mod_delta_rs.append(max_mod_delta_r)

    # mod_deltas = np.repeat(mod_deltas, asteroids)

    if not graph:
        return max_mod_delta_rs

    fig, ax = plt.subplots()
    ax.plot(initial_dxs, max_mod_delta_rs, marker = 'x', color =
    'r', ms = 5)
    if cap:
        ax.set_ylim(0,12)
        ax.set_xlim(-0.2, 0.2)
    ax.set_ylabel('Asteroid Wander /AU')
    ax.set_xlabel(f"Velocity offset (along line perpendicular to
    Sun-Asteroid) /AU per year")
    ax.set_title(f"Asteroid Wander vs Tangential Velocity Shift")


def heatmap(sols, unpert_ode, xlabel, ylabel):
    '''
    plots a heatmap of maximum deviation from L4 of asteroid as a
    function of x and y displacement/velocities
    '''

    max_mod_delta_rs = []
    arr = np.zeros(asteroids**2)
    i = 0
    for ode in sols:
        r_primed = perform_rotation_ode(unpert_ode, ode)
        r_primed_unpert = perform_rotation_ode(unpert_ode,
    unpert_ode)
        delta_r = r_primed - r_primed_unpert
        mod_delta_r = np.linalg.norm(delta_r, axis = 0)
        max_mod_delta_r = np.amax(mod_delta_r)
        max_mod_delta_rs.append(max_mod_delta_r)

        arr[i] = max_mod_delta_r
        i += 1
```

```python

    X = np.reshape(arr, (asteroids, asteroids), order = "C")
    v_values = np.linspace(-max_delta_v, max_delta_v, asteroids // 10 )
    r_values = np.linspace(-max_delta, max_delta, asteroids // 10)

    fig, ax = plt.subplots()
    im = ax.imshow(X, vmin = 0, vmax = 11, origin = 'lower')
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.set_xticks(np.arange(0, asteroids, 10), labels = np.around(v_values, 1))
    ax.set_yticks(np.arange(0, asteroids, 10), labels = np.around(v_values, 1))
    divider = make_axes_locatable(ax)
    cax = divider.append_axes("right", size="5%", pad=0.05)
    cbar = fig.colorbar(im, cax = cax)
    cbar.set_label("Wander /AU")
    ax.set_title("Wander from L4 versus initial offset in x-y plane")

def get_velocity(dx, dy):
    '''
    gets velocity for a point NOT at the lagrange point
    '''

    initial_conditions = circ_orbit_conditions_new(R, m_j, m_s, [0,0], [0,0], [0,0], v_eqm = True, dv = False)
    initial_conds = Conditions(initial_conditions)

    # r is the displacement from the origin of the asteroids, which are displaced from L4 by (dx, dy)
    r = np.array([initial_conds.r_a.x, initial_conds.r_a.y]) + np.array([dx, dy])
    omega = 2 * math.pi / T
    mod_v = np.linalg.norm(r)*omega
    theta = np.arctan(r[0] / r[1])
    v = np.array([mod_v*np.cos(theta), - mod_v*np.sin(theta)])

    return v


def wander(ode, m, unpert_ode):
    '''
    The mass ratio changes the location of Lagrange point wrt the origin, because the origin is the CoM!!
    '''

```

```python
    M = m + m_s # total mass
    gamma = m / M

    X = R * (1 - gamma + gamma**2)**(1/2) # distance between origin and Lagrange point
    th = (1 + gamma) * math.pi / 3 # angle between line to Lagrange point and y axis

    # asteroid conditions
    lagrange = np.array([X * np.sin(th), X * np.cos(th)])
    l_x = lagrange[0]
    l_y = lagrange[1]

    r_primed = perform_rotation_ode(unpert_ode, ode, m = m)
    rs = ((l_x - r_primed[0])**2 + (l_y - r_primed[1])**2)**(1/2)
    max_r = np.amax(rs)
    print(max_r)
    return max_r, gamma

def mass_ratio(odes, ms, unpert_odes):

    max_rs = []
    gammas = []

    zipped_data = zip(odes, unpert_odes, ms)

    for (ode, unpert_ode, m) in zipped_data:

        max_r, gamma = wander(ode, m, unpert_ode)
        max_rs.append(max_r)
        gammas.append(gamma)

    gammas = np.array(gammas)
    max_rs = np.array(max_rs)

    log_gammas = np.log(gammas)
    log_rs = np.log(max_rs)
    fig, ax = plt.subplots()
    fig, ax1 = plt.subplots()
    ax.scatter(log_gammas, log_rs, marker='x', color='r')
    ax.set_ylabel('Log(wander/AU)')
    a, b = np.polyfit(log_gammas, log_rs, 1)
    '''
    b = log(A)
    a = power of gamma
    max_rs = A*gammas^power
    '''
    print(f"a = {a}")
    print(f"b = {b}")
```

```
    ax.plot(log_gammas, a*log_gammas + b)
    ax.set_xlabel('Log(gamma) where gamma is small mass / total mass')
    ax.set_title("Log Log plot: Asteroid Wander vs Mass Ratio")

    y = np.e**b * gammas ** a

    ax1.scatter(gammas, max_rs, marker='x', color = 'r')
    ax1.plot(gammas, y, linestyle='--', label = f'wander = {round(np.e**b, 2)}\u03B3^{(round(a, 2))}')
    ax1.legend()
    ax1.set_title("Asteroid wander vs \u03B3 (small mass / total mass)")
    ax1.set_ylabel("Wander /AU")
    ax1.set_xlabel("\u03B3 (small mass / total mass)")
```

run_orbit.py

```
'''
This script computes all the data for orbits and saves them to files in /data folder
'''

from trojan import *

# deltas = get_deltas(asteroids)
# delta_vs = get_delta_vs(asteroids)
# rand_deltas = get_rand_deltas(asteroids)
# rand_vels = get_rand_vels(asteroids)
dxs = get_ds(-max_delta, max_delta, asteroids)
dys = get_ds(-max_delta, max_delta, asteroids)
dv_xs = get_ds(-max_delta_v, max_delta_v, asteroids)
dv_ys = get_ds(-max_delta_v, max_delta_v, asteroids)

'''
Cannot run the random functions again in another file as they will be different
Save them to .txt and call the data in another file
'''
# np.savetxt('data/rand_vels.txt', rand_vels)
# np.savetxt('data/rand_deltas.txt', rand_deltas)


system = Two_Body_System(m_j, m_s)

unperturbed = system.interact_ode(t_span, [0,0], [0,0], [0,0], v_eqm=True, dv=False)
```

```
unpert_ode = ODE(unperturbed)
np.savetxt('data/unperturbed.txt', unperturbed)

# slightly perturbed asteroid for long term stability investigation
perturbed = system.interact_ode(t_span, [0.1,0.1], [0,0], [0,0], v_eqm = True, dv = False)
pert_ode = ODE(perturbed)
np.savetxt('data/perturbed.txt', perturbed)

def solver_methods():
    unperturbed_ivp_RK45 = system.interact_ivp(t_span, t, 'RK45', [0,0], [0,0], eqm = True).y
    unperturbed_ivp_RK45 = np.transpose(unperturbed_ivp_RK45)

    unperturbed_ivp_RK23 = system.interact_ivp(t_span, t, 'RK23', [0,0], [0,0], eqm = True).y
    unperturbed_ivp_RK23 = np.transpose(unperturbed_ivp_RK23)

    unperturbed_ivp_DOP853 = system.interact_ivp(t_span, t, 'DOP853', [0,0], [0,0], eqm = True).y
    unperturbed_ivp_DOP853 = np.transpose(unperturbed_ivp_DOP853)

    unperturbed_ivp_Radau = system.interact_ivp(t_span, t, 'Radau', [0,0], [0,0], eqm = True).y
    unperturbed_ivp_Radau = np.transpose(unperturbed_ivp_Radau)

    unperturbed_ivp_BDF = system.interact_ivp(t_span, t, 'BDF', [0,0], [0,0], eqm = True).y
    unperturbed_ivp_BDF = np.transpose(unperturbed_ivp_BDF)

    unperturbed_ivp_LSODA = system.interact_ivp(t_span, t, 'LSODA', [0,0], [0,0], eqm = True).y
    unperturbed_ivp_LSODA = np.transpose(unperturbed_ivp_LSODA)

    print(np.shape(unperturbed_ivp_RK45)) # for some reason, this method overrides the t and plots only 70 points not 10,000
    print(np.shape(unperturbed_ivp_RK23))
    print(np.shape(unperturbed_ivp_Radau))
    print(np.shape(unperturbed_ivp_DOP853))
    print(np.shape(unperturbed_ivp_BDF))
    print(np.shape(unperturbed_ivp_LSODA))


    np.savetxt('data/solver/unperturbed_ivp_RK45.txt', unperturbed_ivp_RK45)
    np.savetxt('data/solver/unperturbed_ivp_RK23.txt', unperturbed_ivp_RK23)
    np.savetxt('data/solver/unperturbed_ivp_DOP853.txt',
```

```
    unperturbed_ivp_DOP853)
    np.savetxt('data/solver/unperturbed_ivp_Radau.txt', unperturbed_ivp_Radau)
    np.savetxt('data/solver/unperturbed_ivp_BDF.txt', unperturbed_ivp_BDF)
    np.savetxt('data/solver/unperturbed_ivp_LSODA.txt', unperturbed_ivp_LSODA)

solver_methods()

def polar(R, theta):
    x = R * np.sin(theta)
    y = R * np.cos(theta)
    return np.array([x,y])

# get data for a line of asteroids in the radial direction
for i, dy in enumerate(dys):
    [x,y] = polar(dy, np.pi/3)
    solved_dy = system.interact_ode(t_span, [x,y], get_velocity(x,y), [0,0], v_eqm = False, dv=False)
    np.savetxt(f'data/solved_dy_{i}.txt', solved_dy)

# get data for asteroids with varying velocities in tangential direction
for i, dv_y in enumerate(dv_ys):
    solved_dv_y = system.interact_ode(t_span, [0,0], get_velocity(0,0), polar(dv_y, 5 * np.pi/6), v_eqm = True, dv = True)
    np.savetxt(f'data/solved_dv_y_{i}.txt', solved_dv_y)

# get data for a grid of displaced asteroids in the x-y plane about L4
for i, dx in enumerate(dxs):
    for j, dy in enumerate(dys):
        solved_grid = system.interact_ode(t_span, [dx,dy], get_velocity(dx, dy), [0,0], v_eqm = False, dv=False)
        np.savetxt(f'data/position_grid_{i}_{j}.txt', solved_grid)

# get data for a grid of velocities to add to the equilibrium velocity, all asteroids are positioned at L4
for i, dv_x in enumerate(dv_xs):
    for j, dv_y in enumerate(dv_ys):
        solved_v = system.interact_ode(t_span, [0,0], get_velocity(0,0), [dv_x, dv_y], v_eqm=True, dv=True)
        np.savetxt(f'data/velocity_grid_{i}_{j}.txt', solved_v)

# mass ratio changing, x displacement is 0.0001 and y displacement is zero at eqm velocity
for i, m in enumerate(ms):
```

```
    mass_variation_system = Two_Body_System(m, 1)
    unperturbed = mass_variation_system.interact_ode(t_span,
   [0,0], [0,0], [0,0], v_eqm=True, dv=False)
    solved_m = mass_variation_system.interact_ode(t_span,
   [0.0001, 0], get_velocity(0.0001, 0), [0,0], v_eqm = True, dv
   =False)
    np.savetxt(f'data/solved_m_{i}.txt', solved_m)
    np.savetxt(f'data/unperturbed_{i}.txt', unperturbed)

# # check periods match up
# measured_period = measure_period(ODE(unperturbed))
# print(f"measured time period is {measured_period}")
# print(f"calculated time period is {T}")
```

test_integrator.py

```
'''
script that tests the stability and accuracy of the integrators
'''

from trojan import *

unpert_ode = ODE(np.loadtxt('data/unperturbed.txt'))
unpert_ivp_RK45 = ODE(np.loadtxt('data/solver/
    unperturbed_ivp_RK45.txt'))
unpert_ivp_RK23 = ODE(np.loadtxt('data/solver/
    unperturbed_ivp_RK23.txt'))
unpert_ivp_DOP853 = ODE(np.loadtxt('data/solver/
    unperturbed_ivp_DOP853.txt'))
unpert_ivp_Radau = ODE(np.loadtxt('data/solver/
    unperturbed_ivp_Radau.txt'))
unpert_ivp_BDF = ODE(np.loadtxt('data/solver/unperturbed_ivp_BDF.
    txt'))
unpert_ivp_LSODA = ODE(np.loadtxt('data/solver/
    unperturbed_ivp_LSODA.txt'))


E_ode = energy(unpert_ode, m_j, m_s)
E_ivp_RK45 = energy(unpert_ivp_RK45, m_j, m_s)
E_ivp_RK23 = energy(unpert_ivp_RK23, m_j, m_s)
E_ivp_DOP853 = energy(unpert_ivp_DOP853, m_j, m_s)
E_ivp_Radau= energy(unpert_ivp_Radau, m_j, m_s)
E_ivp_BDF = energy(unpert_ivp_BDF, m_j, m_s)
E_ivp_LSODA = energy(unpert_ivp_LSODA, m_j, m_s)


fig, ax = plt.subplots()
```

```
ax.plot(t, E_ode, label = "odeint")
# ax.plot(t, E_ivp_RK45, label = "solve_ivp: RK45")
ax.plot(t, E_ivp_RK23, label = "solve_ivp: RK23")
ax.plot(t, E_ivp_DOP853, label = "solve_ivp: DOP853")
ax.plot(t, E_ivp_Radau, label = "solve_ivp: Radau")
ax.plot(t, E_ivp_BDF, label = "solve_ivp: BDF")
ax.plot(t, E_ivp_LSODA, label = "solve_ivp: LSODA")
ax.legend()

ax.set_ylim(-0.25, -0)
ax.set_title("Energy vs time (~20,000 periods) for various integrators")
plt.show()
```

plots.py

```

'''
Creates plots for asteroids' motion in non-rotating and rotating frames of reference
'''
from trojan import *

system = Two_Body_System(m_j, m_s)

dxs = get_ds(min_delta, max_delta, asteroids)
dys = get_ds(min_delta, max_delta, asteroids)
unpert_ode = ODE(np.loadtxt('data/unperturbed.txt'))

solved_dys = []
for i, dy in enumerate(dys):
  solved_dy = ODE(np.loadtxt(f'data/solved_dy_{i}.txt'))
  solved_dys.append(solved_dy)

system.planar_plot(-8, 8, -8, 8, solved_dys, unpert_ode)
corotated_deviation(unpert_ode, solved_dys, "Motion in Co-Rotating Frame")
make_plots_ode(unpert_ode, single_deltas) # plot in non rotated frame
plt.show()
```

stability.py

```
'''
Plots graphs to investigate stability of L4 for various displacements, velocities and mass ratios
```

```
'''

from trojan import *

system = Two_Body_System(m_j, m_s)

# rand_vels = np.loadtxt('data/rand_vels.txt')
# rand_deltas = np.loadtxt('data/rand_deltas.txt')
deltas = get_deltas(asteroids)
delta_vs = get_delta_vs(asteroids)
dxs = get_ds(-max_delta, max_delta, asteroids)
dys = get_ds(-max_delta, max_delta, asteroids)
dv_xs = get_ds(-max_delta_v, max_delta_v, asteroids)
dv_ys = get_ds(-max_delta_v, max_delta_v, asteroids)

unpert_ode = ODE(np.loadtxt('data/unperturbed.txt'))
pert_ode = ODE(np.loadtxt('data/perturbed.txt'))

solved_grids = []
for i, dx in enumerate(dxs):
  for j, dy in enumerate(dys):
    solved_grid = ODE(np.loadtxt(f'data/position_grid_{j}_{i}.txt
    '))
    solved_grids.append(solved_grid)

solved_vs = []
for i, dv_x in enumerate(dv_xs):
  for j, dv_y in enumerate(dv_ys):
    solved_v = ODE(np.loadtxt(f'data/velocity_grid_{j}_{i}.txt'))
    solved_vs.append(solved_v)

solved_dys = []
for i, dy in enumerate(dys):
  solved_dy = ODE(np.loadtxt(f'data/solved_dy_{i}.txt'))
  solved_dys.append(solved_dy)

solved_dv_ys = []
for i, dv_y in enumerate(dv_ys):
  solved_dv_y = ODE(np.loadtxt(f'data/solved_dv_y_{i}.txt'))
  solved_dv_ys.append(solved_dv_y)

unpert_odes = []
for i, m in enumerate(ms):
  unpert_ode = ODE(np.loadtxt(f'data/unperturbed_{i}.txt'))
  unpert_odes.append(unpert_ode)

solved_ms = []
for i, m in enumerate(ms):
  solved_m = ODE(np.loadtxt(f'data/solved_m_{i}.txt'))
```

```
    solved_ms.append(solved_m)

mass_ratio(solved_ms, ms, unpert_odes)
# corotated_deviation(unpert_ode, pert_ode, f'Perturbed Asteroid for ~{t_span[1]} periods')


maximum_deviation(solved_dys, dys, unpert_ode, direction = "y")
maximum_deviation(solved_dys, dys, unpert_ode, direction = "y", cap = True)
maximum_deviation(solved_dv_ys, dv_ys, unpert_ode, direction = "y")
maximum_deviation(solved_dv_ys, dv_ys, unpert_ode, direction = "y", cap = True)
system.planar_plot(-8, 8, -8, 8, solved_grids, unpert_ode) # plot for initial displacements of asteroids for the grid
system.planar_plot(3, 7, 1, 5, solved_grids, unpert_ode) # plot for initial displacements of asteroids for the grid (zoomed in)
heatmap(solved_grids, unpert_ode, "Initial x offset /AU", "Initial y offset /AU")
heatmap(solved_vs, unpert_ode, "Initial x-velocity /AU per year", "Initial y velocity /AU per year")
plt.show()
```