Population difference between L4 and L5 Trojans due to Jupiter's outward migration

A numerical analysis using actual Trojan data

by

Bonenkamp, J.D.

Student Name	Student Number
Jasper Bonenkamp	5284023

Instructors:	dr. P.M. Visser & prof. dr. B. Rieger
Project Duration:	Feb, 2023 - Aug, 2023
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

TU Delft

Abstract

Ever since their discovery, the Trojans have raised many questions among astrophysicists. In 1989 it was found that there were more L4 than L5 Trojans and the current L4:L5 ratio value is estimated to be 1.6. However, this asymmetry could not be explained by Jupiter's current orbit and must therefore have arisen during the early development of the Solar System. It was suggested that an outward migration as described by the Nice model could cause the asymmetry. To investigate this, we extended a recent research by modelling an outward migration of Jupiter on the actual Trojans and their symmetric copies. In this manner, we had a broad initial Trojan distribution, which allowed us to investigate several effects of initial values like the Trojan inclinations and maximum angular deviations from the Lagrange point. The simulations used a recently found midpoint Yoshida integrator, which allowed fourth order symplectic simulations for our non-separable Hamiltonian problem. From the simulations it followed that the ratio could only be explained if the Trojans initially had large angular deviations from the Lagrange point, which is possible if they later lost energy due to for example mutual collisions. Also the small initial inclinations as predicted for the early Trojans would in general have risen due to the migration, but it could not explain the the inclinations up to 40 degrees for today Trojans, implying that the increase in inclination must have been caused by other reasons. We found that not only the total migration distance and duration are important, but also the function that describes the evolution of Jupiter's semi-major axis and the eccentricity of Jupiter's orbit. For further research we suggest to use data sets on those quantities from simulations with the Nice model, to investigate these related problems together. We believe that our model and code is of great advantage for such a research, because of its optimisations and it allows to implement the data on the distance and eccentricity directly into the model without having to alter the equations of motion.

Contents

1 Symbols used

In table 1.1 all the symbols used in this report are given, together with their meaning. These definitions always have the same meaning in the report, unless explicitly mentioned otherwise. For some variables

Symbol	Physical meaning
r, θ, z	Cylindrical coordinates that describe the Trojan position with respect to the centre of mass of the sun and Jupiter
p	Generalised momentum of r
l_z	Generalised momentum of θ
p_z	Generalised momentum of z
R	Distance Jupiter-sun
ω_J	Jupiters angular velocity
V	Total gravitational energy function, as derived in section 3.3.1
m_S	Mass of the Sun
m_J	Mass of Jupiter
μ	$\frac{m_J}{m_S}$
G	Gravitational constant
$\mathcal{H}$	Hamiltonian
$\mathcal{L}$	Lagrangian
τ	Constant that is correlated with Jupiters migration duration
ΔR	Total change in R
a	Semi-major axis
ϵ	Eccentricity
ω	Argument of periapsis
Ω	Longitude of ascending node
I	Inclination
ν	True anomaly
M	Mean anomaly
$\bar{v}$	Velocity quasi pre-factor
$\Delta\theta$	Resonant angle (maximum angular deviation from the corresponding Lagrange point)
AU	Astronomical unit ($1.496 \cdot 10^{11}$ km)

Table 1.1: Symbols used to describe and derive the Hamiltonian and equations of motion of a Trojan in the cylindrical rotating frame.

from the table as well as for other variables in the report, it is sometimes emphasised with a sub-scripted J that it belongs to a description of Jupiter and not of a Trojan.

2 Introduction

Figure 2.1 shows a meteorite which exploded in 2013 above the Russian city Chelyabinsk. The meteorite impact released more energy than the atomic bomb at Hiroshima [5], and caused a few hundred people to be hospitalized and many more buildings to be damaged[5]. However, the asteroid, was not discovered before the impact[5] despite all modern knowledge and technology, making it impossible to start evacuations. One could imagine the consequences would be even worse if the explosion happened above a more crowded city. Apparently, we do not have enough knowledge yet about asteroids in space to prevent potential disasters like these.

Apart from asteroids that might threaten our lives, there are also many asteroids in space that could not intrude the Earth. In fact, humans have used the knowledge about the orbits of those objects to created their own 'asteroids' such as the GPS satellite shown in figure 2.2. In modern society, artificial satellites are used by millions of people on a daily basis, for example to find the road to work, family or anything else. These two examples show how our knowledge of objects in space could improve our all day lives, while our lack of knowledge might threaten our lives because unstable objects intrude our Earth. It is therefore important to understand the orbits of objects in space better, in order to use the theory to our advantage when creating our own 'asteroids' and to predict the behaviour of potential harmful asteroids better. One of the aspects of the Solar Systems that is poorly understood, is its early evolution and the long term stability of its objects. While there are several theories about these problems, it is key to test and improve those theories by observing actual objects in our Solar System. In this manner, we are able to expand our knowledge piece by piece.

One of the objects that could give astronomers more insight in the current Solar System stability and its early evolution are the Jupiter Trojans. The Trojans are asteroids, which orbit around Jupiter's Lagrange points L4 and L5, as described in section 3. An example of some Trojans is given in figure 2.3 and figure 2.4 gives some insight in the distribution of Trojans relative to Jupiter. Already in the 17th century, Lagrange predicted the existence of Trojans and more than a century ago Max Wolf discovered the first Trojan[17]. Since then many questions about the Jupiter Trojans have been unanswered, such as its asymmetric distribution.

We can define three types of Trojans: a Trojan orbiting around the L4 point, around the L5 point and one in a horseshoe orbit. An example for each orbit in the xy reference plane corotating with Jupiter is shown in figures 2.5, 2.6, 2.7. If a Trojan is no longer in one of these orbits, we say that its orbit has become unstable. One notices that the orbits of L4 and L5 Trojans are all banana shaped, but have different sizes, spreads and curvatures. However, at first glance, there does not appear to be much difference between an L4 and an L5 Trojan. In fact, it will be proven in the theory chapter that under the assumption that Jupiter makes perfect circular orbits, the Trojans are completely symmetric. Due to this symmetry, one expects similar properties for L4 and L5 Trojans. However, there are numerous differences observed between L4 and L5 Trojans, from which the number of Trojans is the most noticeable. From observations, it is estimated that there are about 1.6 times as many Trojans in the L4 Lagrange point than in the L5 point[2]. Although the symmetry is in fact broken on the long

Figure 2.1: Meteorite that exploded above the Russian city Celyabinsk in 2013. The meteorite was not discovered before the disaster, but caused a few hundred hospitalizations and many more buildings to be destroyed. Source: [5]

Figure 2.2: GPS satellite, which is located in a stable orbit around Earth and is used by many people on a daily basis for navigation. Source: [4]

time scale due to for example interactions with Saturn, these effects are not sufficient to explain the current high ratio and at least part of this ratio should be related to the early formation or capture of the Trojans[3]. Another difference is the inclination distribution. The inclination is a measure that defines how skew the Trojan orbit is relative to Jupiter's orbit. Although there is not much known about the inclination distributions, there is a 99% confidence interval the distributions are different[2].

Figure 2.3: Constructed images of some Jupiter Trojans. Source: [6]

2.1. Nice model and Li

There have been several attempts to explain the ratio between the number of L4 and L5 Trojans, which will be denoted by L4:L5 in this report, and a recent attempt was made by Li et al.[2]. They simulated an outward migration of Jupiter as theorized by the Nice model. Before we are able to explain the correlation between the Nice model and the Trojans, we have to briefly introduce the Nice model.

The Nice model is one of the models that attempts to explain the origin of our Solar System. One of the peculiar things about our Solar System, is that not all planets could have been formed easily at their current positions with their current orbit shapes. Therefore, several early-Solar System models have migrations of gas giants, which is the phenomenon that the average distance of such a giant relative to the Sun either increases or decreases. Most of those models assumed that Jupiter migrated due to gravitational interactions with Saturn[2]. However, Nesvorný[15] noticed that many simulations with these models fail to match the current orbits and led to unstable systems resulting in one of the gas giants to be kicked out of the Solar System. He therefore proposed a model in which the early system had five gas giants where one was kicked out due to continuous gravitational interactions. This model was called the Nice model. The several unbounded planets observed traveling through space suggests that such an ejection of a planet is indeed possible[15].

In the Nice model, one of the planets initially had a very elliptical orbit leading to several close encounters between the gas giants. When a close encounter occurs, the orbits of the two planets are altered due to their mutual gravitation. One of these encounters was responsible for an outward migration of Jupiter, and Nesvorný[16] suggested that this could be the cause of the asymmetric L4:L5 ratio, due to Trojans becoming unstable. Nesvorný had quite large uncertainties in his results, but in a different research, Li et al.[2] found that such a migration might indeed cause the asymmetry in the number of L4 and L5 Trojans.

For their research, Li et al. simulated several randomised sets of Trojans using different migrations of Jupiter. Although they do not use the Nice model explicitly in their research and this migration could also be caused by other reasons, they did use this model to determine estimates for the migration rates and distances that could be considered[2]. With this assumption, a migration rate of $\dot{r} = 1.5 \cdot 10^{-4}$AU y^{-1} over a distance of 0.5 AU resulted in the estimated empirical L4:L5≈ 1.6. Provided the migration is not too fast, they found that both a longer and a faster migration lead to an increase in this ratio.

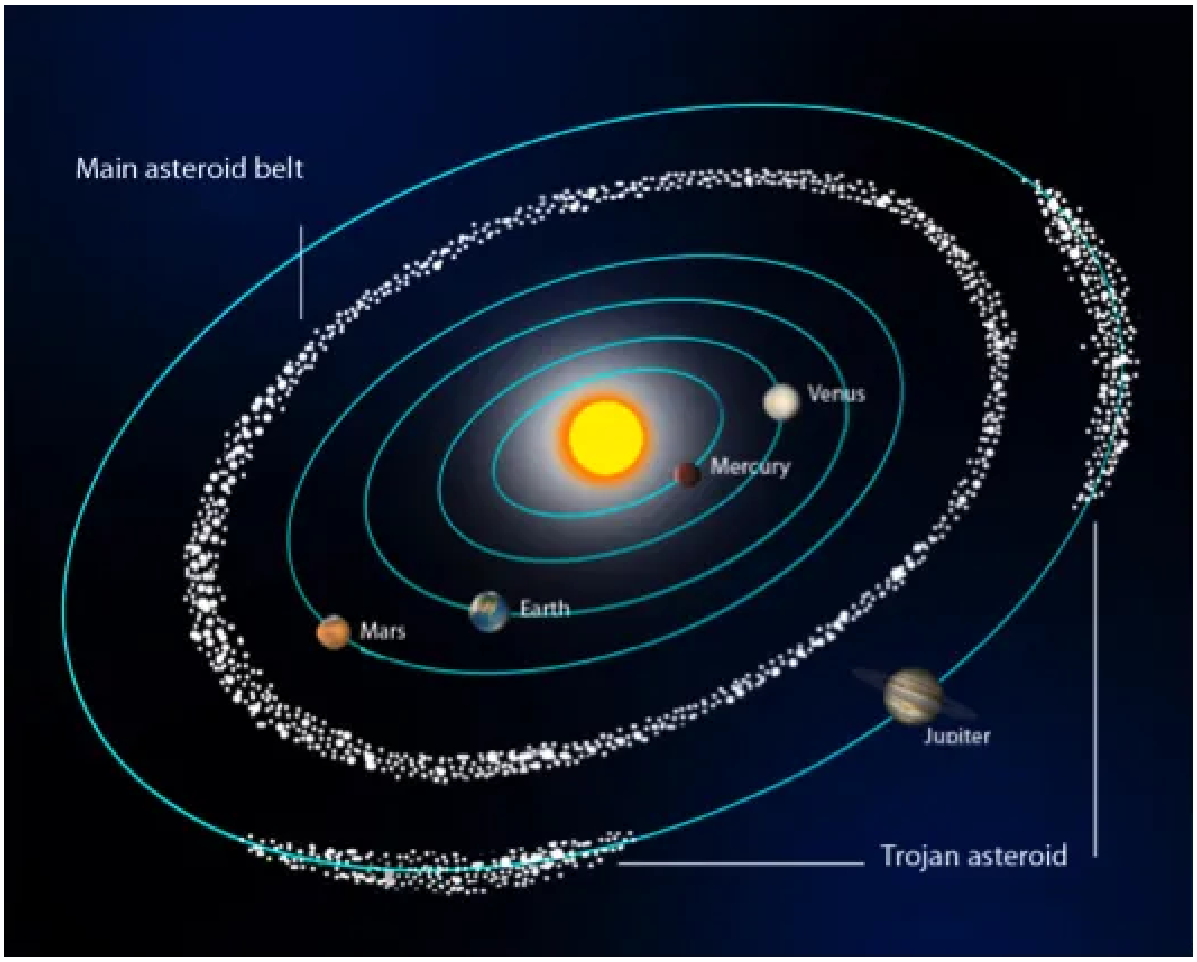


Figure 2.4: The Solar System with the main-belt asteroids and the Jupiter Trojans. Source: [18]

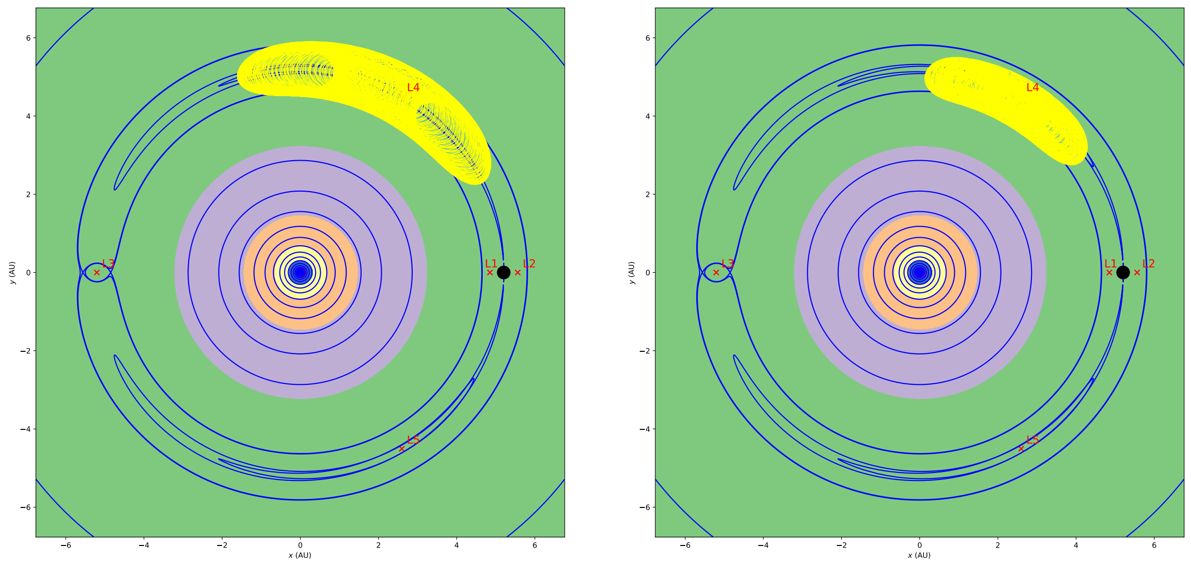


Figure 2.5: Two examples of actual Trojans orbiting the L4 point in the horizontal plane together with the equipotential lines. The black dot represents Jupiter and the colored circles ranges of the potentials, where the scale is logarithmic. One notices that the curves are both banana shaped, but have different sizes and bendings.

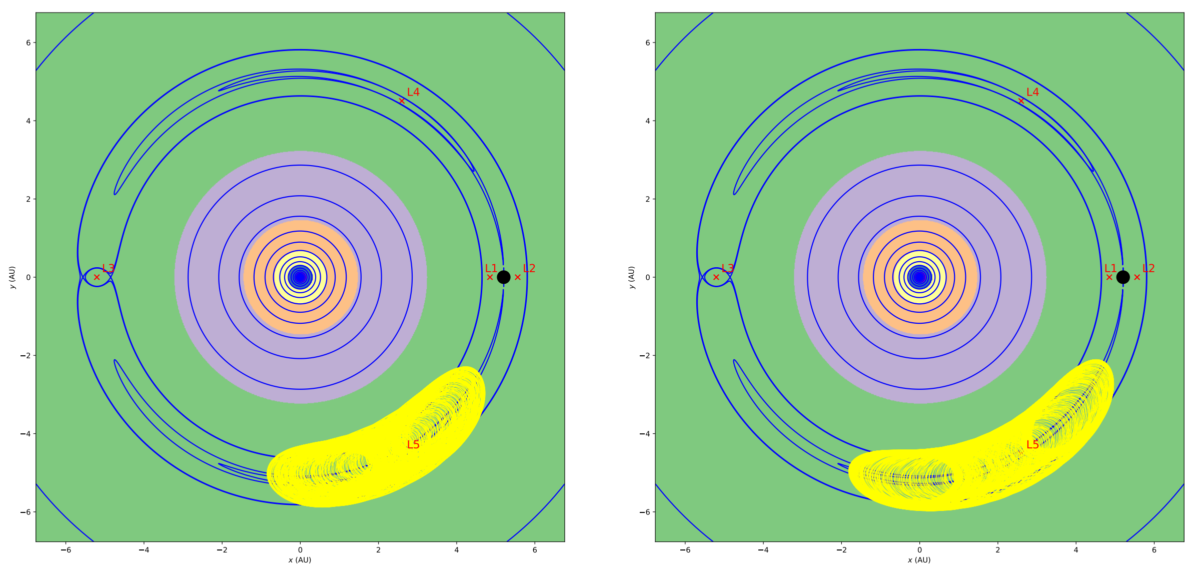


Figure 2.6: Two examples of actual Trojans orbiting the L4 point in the horizontal plane together with the equipotential lines. The black dot represents Jupiter and the colored circles ranges of the potentials, where the scale is logarithmic.

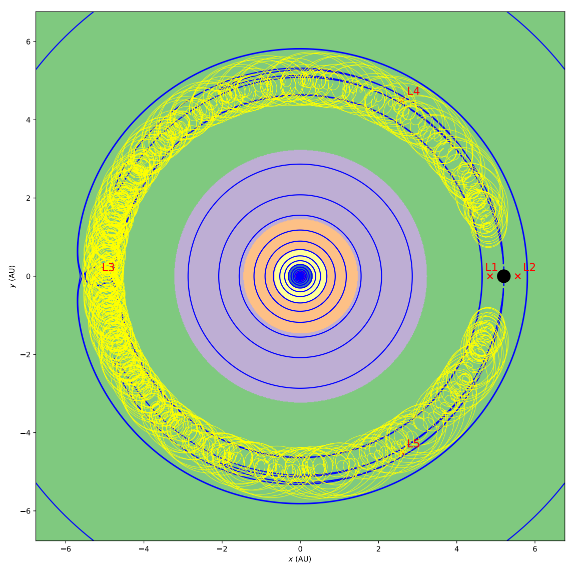


Figure 2.7: Examples of an actual Trojan in a horse shoe orbit in the horizontal plane together with the equipotential lines. The black dot represents Jupiter and the colored circles ranges of the potentials, where the scale is logarithmic.

However, according to the authors, the model is quite crude and they suggest some improvements. One of the shortcomings of the model they mention are the behaviour of inclinations. Li et al. showed that the ratio is dependent on the inclination distribution, although not strongly. They simulated most results with low inclinations, but also mention that the inclination distribution of the L4 and L5 Trojans are different but not well known. They suggest the model to be generalised for the actual Trojans with a broad inclination distribution.

We believe that the restricted initial condition could be seen in a much broader manner. Li et al. did not only use low inclinations, but also a very restricted range in other quantities. For example, the Trojans they used all had the same semi-major axis initially and had relatively high deviations from the Lagrange points in the angular directions. This is not necessarily wrong since the initial conditions are not known, but it does not allow a proper research what initial conditions could lead to a high L4:L5 ratio. Therefore, their research could be improved in general by using a broader initial range of Trojans, to investigate the effect of the initial conditions.

2.2. Actual Trojans

This project attempts to investigate part of the effect of a broad Trojan distribution. This is done by using the current actual Trojan data as start, since this gives a wide range of initial conditions for initially stable orbits. The migration used will be described in the same manner as Li et al. and we will use their found migration rates and distances as an estimate of what rates could be adopted. We will use several of the migration durations to give a statistically more significant conclusion, but it is not our goal to find the actual value of the migration rate. This research aims to find the required initial Trojan distribution such that the outward migration as prescribed by Li et al. could explain the current asymmetry in the population numbers. To exclude any effect of the initial data asymmetry between the L4 and L5 Trojans, we created a symmetric initial distribution by adding minor particles for each actual Trojan.

We will also have a closer look on the inclinations distributions. It will not only be investigated what the effect is of the initial inclinations as suggested by Li, but it will also be investigated how they change due to the migration. Many theories namely predict that the Trojans had low initial inclinations relative to Jupiter[11], while the present Trojans have inclinations up to 40 degrees. Furthermore, it can be checked whether the migration could explain the asymmetry in inclination distribution between the L4 and the L5 point.

It has already been clear that a research on the effect of the migration might give insight in the original Trojan distribution, but there are at least two other reasons to investigate the effect of the migration on the actual Trojans. First of all, especially for the ratio L4:L5, there is no absolute certainty of the distribution from only the observations. When Shoemaker et al. (1989)[19] observed the difference in population, the L4 point was better observable than the L5 point[2]. Although the asymmetry is nowadays commonly excepted as more and more Trojans are observed[2], the simulations help to strengthen the claim that the asymmetry is real and not due to any observational bias. Also, scientists that hypothesise the distribution, might have a bias in observing. It must be noted, however, that all biases are taken into account when Li et al. estimated the ratio L4:L5 and therefore one has to emphasize the simulations only strengthen or weaken the possible estimates. The second reason is that any theory about the early Trojan development leading to an asymmetry are strengthened or weakened by simulating these models. In our research, the asymmetry is hypothesised by the Nice model, which models the origin of our Solar System. Therefore, if the simulations with this model indeed lead to more L4 than L5 Trojans, the likeliness of the Nice model increases and it could help to describe the migrations by this model better. Although this is investigated by Li et al. our research is of a major importance to improve such a research, because it takes into account the effect of several initial conditions that might be relevant for the L4:L5 ratio and therefore for the actual migration description.

3 Theory

In this chapter, we will derive all needed theory about the Trojans. For this, we will firstly discuss the three body problem and the Lagrange points. This theory partly requires the theory of two body systems because we will need the Kepler elements defined for this problem. Once all theory is set up, we will define a coordinate system for our Trojans using cylindrical coordinates in the rotating frame, since this is the most insightful set of coordinates. We can then derive the equations of motion using Hamiltonian mechanics. This chapter will end with a more detailed look on the symmetry and asymmetry of the L4 and L5 Lagrange points.

3.1. Kepler elements

In space, the standard way to determine an object's orbit, is to consider it as part of a two body system, which is a system containing only two masses. For two body systems the orbits are called Kepler orbits and can be described using the Kepler elements. Five of the Kepler elements are shown with name and symbol in figure 3.1. The sixth is the eccentricity ϵ. In a two body system, the Kepler elements are all constant over time, except for the True anomaly, and fully define the exact orbit of a two body system relative to a reference vector. To describe a system with many small bodies orbiting a heavy central mass, we pretend it is in a Kepler orbit where the Kepler elements are constantly altered due to mutual gravitational attractions. Because the Kepler elements are no longer constant, it is important to know at which time these coordinates are determined. This time is called the epoch and defined with the letter E in this report.

Although we will need to be able to transform all Kepler elements to Cartesian coordinates, we need to know the meaning of only two of them in this report: the eccentricity ϵ and the inclination I. Therefore, we will start to give the transformations from Kepler coordinates to Cartesian coordinates and then discuss the meaning of the eccentricity and the inclination.

3.1.1. Conversion to cylindrical system

As seen in figure 3.1, the Kepler coordinates are determined relative to a plane of reference. For the data used in this project[14][13], the central reference point is the Sun. It is known how to convert the Kepler elements to Cartesian coordinates with respect to the same reference frame. However, in this project we will describe the positions of the Trojans relative to the position of Jupiter. We will therefore choose our coordinate system such that Jupiter is stationary on the horizontal x-axis. That is, we describe the Trojans as if the observer rotates around the centre of mass with the same angular velocity ω_J as Jupiter. This frame is called the rotating frame. Therefore, for a correct conversion for a Trojan in Kepler coordinates to Cartesian coordinates within the rotating frame of Jupiter, we will firstly calculate the Cartesian coordinates in the non-rotating frame in the first three steps by known theory and than rotate the system such that Jupiter is on the horizontal axis with the centre of mass on the origin. In total, we have to execute the following 7 steps:

1. Calculate the absolute value of the distance to the Sun and the quasi velocity pre-factor of the Trojan

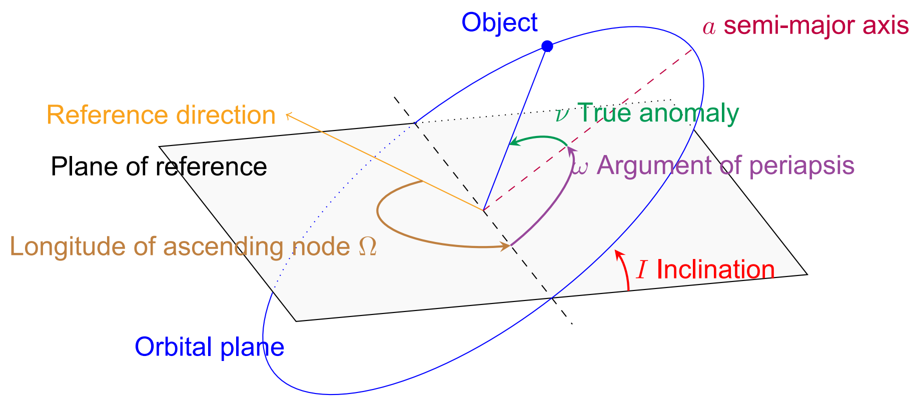


Figure 3.1: Names and symbols of five of the Kepler elements. The sixth is called the eccentricity ϵ, which determines whether the orbit is more circular or more elliptical. For a non two body system, the Kepler elements are all functions of time, thus the time at which the elements are computed is necessary to define the orbit. This time is called the epoch. Generated with help of the code from: [1]

at the epoch of the data in Cartesian coordinates.
2. Use these quantities and the anomaly to create vectors for the position and velocity in the xy-plane.
3. Use a rotation matrix to rotate the vectors to the plane of the orbit.
4. Rotate the Trojan and Jupiter by some rotation matrix such that Jupiter moves in the polar plane.
5. Calculate Jupiter's anomaly at the epoch of a Trojan by assuming it is on a perfect Kepler orbit.
6. Rotate the position and velocity vectors of the Trojans such that Jupiter is on the positive x-axis.
7. Shift the Trojans such that the origin is at the centre of mass.

Mathematically, these steps are executed in the following manner:

1. From [25], one extracts for the distance r_S of the Trojan to the Sun

$$r_S = \frac{b^2}{a + c\cos\nu}, \quad b := \sqrt{1-\epsilon^2}a, \quad c := a\epsilon$$

Combining these results yield

$$r_S = a\frac{1-\epsilon^2}{1+\epsilon\cos\nu} \tag{3.1}$$

Then, by combing some results in Visser[25], we define for simplicity the quasi velocity pre-factor $\bar{v}$.

$$\bar{v} = \sqrt{\frac{Gm_S}{a(1-\epsilon^2)}}$$

2. Using the results above, the vectors in the Cartesian non-rotating frame are[25]:

$$\vec{r} = r\begin{bmatrix}\cos\nu\\ \sin\nu\\ 0\end{bmatrix}, \quad \vec{v} = \bar{v}\begin{bmatrix}-\sin\nu\\ \cos\nu+\epsilon\\ 0\end{bmatrix} \tag{3.2}$$

However, often the True anomaly ν is not given, but instead the mean anomaly $M = \frac{2\pi}{T}\cdot t - \tau$, where T is the orbit time and τ is the time of perihelion passage in positive direction, or in other words, the time at which the true anomaly is zero[12]. The reason this quantity is used often, is that it changes linearly in time for systems that can be estimated as a two-body system. The conversion between the mean and true anomaly can be given by a Fourier series. In this research, we use the third order Fourier series estimation:

$$\nu = M + (2\epsilon - \epsilon^{\frac{3}{4}})\sin(M) + \frac{5}{4}\epsilon^2\sin(2M) + 13\epsilon^{\frac{3}{12}}\sin(3M) \tag{3.3}$$

3. Multiply the vectors above by the following rotation matrix[25], constructed from the Trojan Kepler orbits, to obtain the new vectors.

$$\mathcal{R} = \begin{bmatrix} \cos\Omega & -\sin\Omega & 0 \\ \sin\Omega & \cos\Omega & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos I & -\sin I \\ 0 & \sin I & \cos I \end{bmatrix} \begin{bmatrix} \cos\omega & -\sin\omega & 0 \\ \sin\omega & \cos\omega & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

Sometimes, the mean perihelion length $\tilde{\omega}$ is given instead of the argument of perihelion ω. Then we can calculate the latter by $\omega = \tilde{\omega} - \Omega$. 4. If we would have calculated the Cartesian coordinates of Jupiter, we would have used a rotation matrix as in equation 3.4, to transform Jupiter from the xy-plane to its actual position. However, we would like Jupiter to be in the xy-plane, thus this operation must be inverted. The position relative to the Trojans must remain the same, thus we have to apply the inverse operation to the Trojans as well. All in all, we thus have to multiply the Trojans position and velocity vectors with the inverse of matrix 3.4, but then with the Kepler elements substituted with Jupiter's elements ω_J (not the angular frequency here!), Ω_J, I_J. This yields the matrix:

$$\mathcal{R}_J^{-1} = \begin{bmatrix} \cos\omega_J & -\sin\omega_J & 0 \\ \sin\omega_J & \cos\omega_J & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos I_J & -\sin I_J \\ 0 & \sin I_J & \cos I_J \end{bmatrix}^{-1} \begin{bmatrix} \cos\Omega_J & -\sin\Omega_J & 0 \\ \sin\Omega_J & \cos\Omega_J & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1}$$

The individual matrices in this product can be recognized as rotation matrices, thus we could simply calculate the inverse by inverting the rotation operations. We thus have to replace all the angles by their opposites. This results in the matrix

$$\mathcal{R}_J^{-1} = \begin{bmatrix} \cos\omega_J & \sin\omega_J & 0 \\ -\sin\omega_J & \cos\omega_J & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos I_J & \sin I_J \\ 0 & -\sin I_J & \cos I_J \end{bmatrix} \begin{bmatrix} \cos\Omega_J & \sin\Omega_J & 0 \\ -\sin\Omega_J & \cos\Omega_J & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5. Ddefine the epoch of the Trojan as E_T and of Jupiter as E_J. Since the mean anomaly changes linearly with time, we can thus convert the original M_J to the one at the epoch of the Trojan as $M_J' = M_J + \frac{2\pi}{T}(E_T - E_J)$. After that, we use equation 3.3 to calculate Jupiter's true anomaly ν_J. To be as accurate as possible, we consider Jupiter's real, non-zero orbital eccentricity in contrast to the rest of the research.
6. Since we could calculate Jupiter's position vector relative to the position of the Sun using the matrix $\begin{bmatrix} \cos\nu_J \\ \sin\nu_J \\ 0 \end{bmatrix}$, we have to invert this operation. This could be achieved by multiplying the Trojans with the rotation matrix:

$$\begin{bmatrix} \cos\nu_J & \sin\nu_J & 0 \\ -\sin\nu_J & \cos\nu_J & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

The velocity vector of the Trojans must have the same position relative to the Trojans position after the rotation, thus it should be multiplied with this matrix as well.
7. We have to shift Jupiter to the left over a distance of μR, thus we subtract the vector $\mu R\hat{x}$ from the Trojans position vector.

Now we will describe the meaning of the two important variables the eccentricity and the inclination.

Eccentricity

The eccentricity is the measure that determines whether the orbit of an object is circular, elliptical, parabolic or hyperbolic, as shown in table 3.1. The last two belong to unbounded trajectories and the hyperbolic one is very important for the discussed Nice model[15]. Stable objects part of a two-body system have an elliptical orbit. However, when eccentricities are small, it is sometimes preferred to approximate the orbits circular, since often this simplifies the calculations. In this report, we will run simulations with both Jupiter having zero and non-zero eccentricity to investigate the influence of such a simpler model. For the latter case, this can be implemented by letting Jupiter's distance R_J oscillate in the rotating frame as will be described in the method section. The disadvantage of considering the eccentricity, is that it results in more difficult equations, making the code slower, and it causes extra oscillations for the Trojans, making it harder to interpreted the results. Furthermore, we introduce additional errors as we will have to make estimations for determining the position of Jupiter in the rotating frame. On the other hand, if Jupiter had an eccentricity during the migration, this would be more accurate than not considering the eccentricity.

Eccentricity	Orbital shape
0	circular
0-1	elliptical
1	parabolic
>1	hyperbolic

Table 3.1: Possible eccentricity values for a two-body system and its corresponding orbit shape.

Inclination

The second element of interest is the inclination, whose importance was already briefly discussed in the introduction. It is a measure that defines the angle between some reference plane and the plane in which the object is moving. In this report, we will consider the plane of Jupiter's orbit as reference plane when discussing the inclination of a Trojan and we can thus interpreted it as the quantity describing how skew the Trojans orbit is relative to Jupiter. Its definition is[28]

$$I = \text{arccos}\,\frac{L_z}{|\vec{L}|}$$

where $\vec{L}$ is the angular momentum of a Trojan and L_z the z-component of this angular momentum. In this project, we are interested in the effect of the migration on the distribution of inclinations, and therefore, we need to be able to actually calculate the inclinations. These transformations are discussed in subsection 3.4.3 since it is easiest to do this directly from the coordinate system that will be used in the model.

3.2. Three body system and Lagrange points

We can now extend the two body to a three body system by adding a small mass, which is called an asteroid. In a three body system, with two large masses and one small masses, there are five equilibrium (in the rotating frame) points for the position of the small mass under the mutual gravitational forces. The points are shown in figure 3.2.

The Lagrange points exist because in the rotating frame the gravitational forces on the small mass are exactly in balance with the centrifugal force. As has become clear from the introduction, the points of interest for this research are the L4 and L5 Lagrange point, which form an equilateral triangle with the Sun and Jupiter. We will firstly provide a proof that these points are indeed equilibrium points. For this proof, one does not have to assume that one of the masses is small. We can assume the masses form an equilateral triangle and then prove that the resulting force on any of the masses is zero. That is, assuming mass 1 at position $\vec{r_1} = \begin{bmatrix}0\\0\\0\end{bmatrix}$, mass 2 at $\vec{r_2} = \begin{bmatrix}R\\0\\0\end{bmatrix}$, mass 3 at $\vec{r_2} = \begin{bmatrix}\frac{1}{2}\\\frac{1}{2}\sqrt{3}\\0\end{bmatrix} R$ (be aware that we have chosen a slightly different coordinate system than will be used later in this report) one has to prove that R is constant and all objects orbit with the same frequency under their mutual gravitational force. This is equal to proving that assuming the particle is not accelerated in the rotating frame and has no velocity in this frame, there exists an angular frequency ω with its corresponding vector in the z-direction such that this assumptions hold. In other words, one needs to prove that Newtons second law for all three particles give a unique solution for ω. Call the positions vectors of the particles with respect to their centre of mass $\bar{r_1}$, $\bar{r_2}$ and $\bar{r_3}$. Then for all r_i Newtons second law for the rotating frame yield

$$\vec{0} = \vec{F_g} - m_i\vec{\omega} \times (\vec{\omega} \times \bar{r}_i) - 2m_i\vec{\omega} \times \vec{v} = 0 \Rightarrow \sum_{j\in\{1,2,3\}\setminus\{i\}} \frac{Gm_im_j}{|\vec{r_i} - \vec{r_j}|^3}(\vec{r_i} - \vec{r_j}) = -m_i\omega^2\bar{r}_i$$

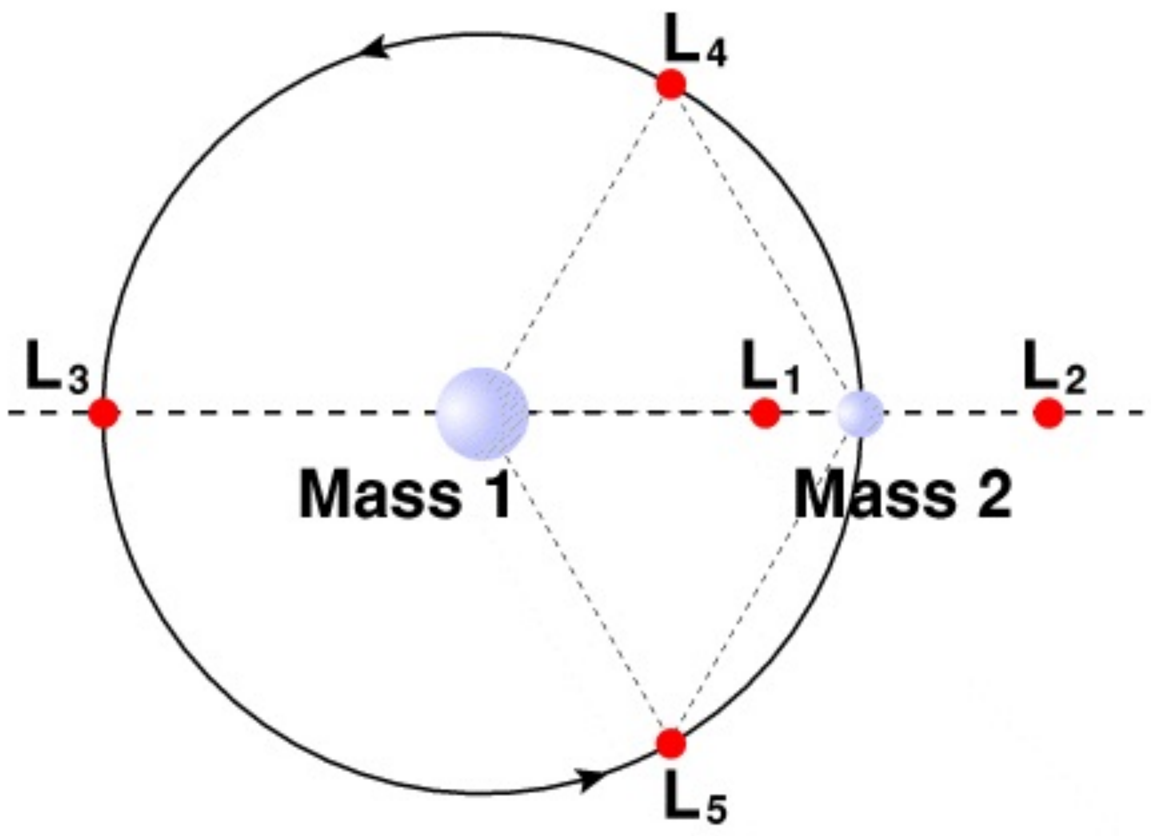


Figure 3.2: The five Lagrange points for a three body system. Source: [27]

Since $|\vec{r_i} - \vec{r_j}| = R$ for all $i \neq j$, we divide both sides by m_i and simplify the expression to

$$\sum_{j \in \{1,2,3\}} \frac{Gm_j}{R^3}(\vec{r_j} - \vec{r_i}) = -\omega^2 \bar{r}_i =$$

$$-\omega^2 \left(\vec{r_i} - \frac{1}{M} \sum_{j \in \{1,2,3\}} \vec{r_j} m_j \right) = -\omega^2 \left(\sum_{j \in \{1,2,3\}} \frac{m_j}{M} \vec{r_i} - \sum_{j \in \{1,2,3\}} \vec{r_j} \right)$$

It thus follows that $\omega = \sqrt{\frac{GM}{R^3}}$ Since the above expression is independent of the individual masses of the three particles, there indeed is a frequency such that the particles are in equilibrium in the frame rotating with this ω. Therefore the L4 and L5 points are indeed equilibrium points in the rotating frame. Notice that since the Trojan mass is small compared to the mass of the Sun and Jupiter, the angular frequency found corresponds with Jupiter's angular velocity around the Sun.

The reason these two Lagrange points are of interest is that they are, in contrast to the other Lagrange points, stable under the condition that one of the large masses is more than about 24 times large than the other large mass[27]. These conditions hold in the Sun-Jupiter system, which is why there could orbit so many asteroids around this orbit. The asteroids around these Lagrange points of Jupiter are called Trojans and make up about half of the total asteroids in our Solar System. After we have set up a coordinate system and derived the equations of motion, we will have a look at the Lagrange points once again to elaborate the theoretical pre knowledge about their symmetry and asymmetry.

3.3. Equations of motion

In the section, we will derive the equations of motion for the Trojans. The first step is to choose a coordinate system, which will be the cylindrical coordinates in the rotating frame of Jupiter. This system is chosen, since it gives more insight about the stability of the L4 and L5 points, as will be clear in section 3.5.2. Since Jupiter's orbital eccentricity is small, the orbit is considered circular for simplicity. The conversion from the Cartesian coordinates x, y and z to the cylindrical coordinates r, θ, z for the radial, angular and z components respectively is given by

$$\begin{aligned} r &= \sqrt{x^2+y^2} \\ \theta &= \text{atan2}\,(y,x) \\ z &= z \end{aligned} \tag{3.6}$$

and the unit directional vectors are given by

$$\begin{aligned} \hat{r} &= \cos\theta\hat{x} + \sin\theta\hat{y} \\ \hat{\theta} &= -\sin\theta\hat{x} + \cos\theta\hat{y} \\ \hat{z} &= \hat{z} \end{aligned} \tag{3.7}$$

We can also express the Cartesian position vector in terms of this coordinates by:

$$\vec{r} = r\cos\theta\hat{x} + r\sin\theta\hat{y} + z\hat{z} \tag{3.8}$$

With the coordinate system set up, we will derive the equations of motion using Hamiltonian mechanics. One therefore firstly has to calculate the potential and kinetic energy, the Lagrangian and some generalised momenta. If we would consider frictions, we would have mass dependent accelerations, but in outer space there is negligible friction. Also, the Trojan masses are much smaller than Jupiter and the Suns masses, such that they do not alter the orbit of those big masses. Therefore the Trojan masses could be divided out in every concerned force and have no influence on the final equations of motion. Therefore, we will choose the Trojan masses unit in the derivations for simplicity.

3.3.1. Potential

Using Cartesian coordinates one could firstly find a potential V. Since the potential is not dependent of the velocity of particles but only relative to the positions, we will use the Cartesian rotating frame around the common centre of mass of the Sun and Jupiter, which are then both positioned on the x-axis. Let $\vec{R_J}$ be the distance vector of Jupiter to this centre of mass, m_J, m_S the mass of Jupiter and the Sun respectively and $\vec{R_S}$ the distance vector of the Sun to the centre of mass. Using that the centre of mass is in the origin, we can express $\vec{R_S}$ in terms of $\vec{R_J}$. If we define $\mu = \frac{m_J}{m_S}$ in this report we then obtain:

$$0 = \frac{\vec{R_J}m_J + \vec{R_S}m_S}{m_J+m_S} \Rightarrow \vec{R_S} = -\vec{R_J}\frac{m_J}{m_S} = -\mu\vec{R_J}$$

We could find the potential due to the Sun and Jupiter by integrating Newton's law of gravity which yields

$$V = -\frac{Gm_S}{|\vec{r}-\vec{R_J}|} - \frac{Gm_J}{|\vec{r}-\vec{R_S}|}$$

Since Jupiter and the Sun are on the x-axis in the chosen reference frame, we could express $\vec{R}_J = R\hat{x}$ and $\vec{R_S} = -\mu R\hat{x}$ where R_J is defined as the distance from Jupiter to the centre of mass. Using equation 3.8, this can be rewritten as

$$V(r,\theta,z) = -\frac{Gm_S}{\sqrt{r^2+\mu^2R^2+2\mu rR\cos\theta+z^2}} - \frac{Gm_J}{\sqrt{r^2+R^2-2rR\cos\theta+z^2}} \tag{3.9}$$

3.3.2. Kinetic energy

Apart from the potential energy, which is determined by the positional elements, there also is a kinetic energy T which is determined by the velocity coordinates in the non-rotating frame. If $\vec{r}$ and $\vec{v}$ are the position and velocity vectors within the rotating frame with angular frequency ω_J, then the total velocity can be calculated by

$$\vec{v}_{\text{total}} = \vec{v} + \vec{\omega_J} \times \vec{r} \tag{3.10}$$

Although this formula is familiar among physicists when the angular velocity is constant, it is actually valid as well when the angular velocity is a function of time. This is an important property for our research, since we will consider migrations of Jupiter in which the angular velocity will not be constant. Therefore, we will prove formula 3.10 for the general case. For this, suppose that in the rotating frame an object has a position $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ in Cartesian coordinates and the frame has a rotation speed of $\omega(t) := \frac{d}{dt}\phi(t)$, where $\phi(t)$ is the argument of the frame relative to the inertial frame. The real position of the object in the non-rotating frame is then given by the vector $\mathcal{R}\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ where

$$\mathcal{R} = \begin{bmatrix} \cos(\phi(t)) & -\sin(\phi(t)) & 0 \\ \sin(\phi(t)) & \cos(\phi(t)) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

is a rotation matrix. Using the chain rule and $\omega(t) := \frac{d}{dt}\phi(t)$, we therefore find that the total velocity is given by

$$\vec{v}_{\text{total}} = -\omega_J(t)\begin{bmatrix} \sin(\phi(t)) & -\cos(\phi(t)) & 0 \\ \cos(\phi(t)) & \sin(\phi(t)) & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} + \mathcal{R}\dot{\begin{bmatrix} x \\ y \\ z \end{bmatrix}} = \vec{v} + \vec{\omega}(t) \times \vec{r}$$

Hence we conclude that equation 3.10 indeed holds, even when ω_J is time dependent. We can convert this equation to our cylindrical coordinates using the transformation in equation 3.8, which yields

$$\vec{v_{\text{total}}} = \dot{r}\hat{r} + r\dot{\theta}\hat{\theta} + \dot{z}\hat{z} + (r\hat{r} + z\hat{z}) \times \omega_J\hat{z} = \dot{r}\hat{r} + r(\dot{\theta} - \omega_J)\hat{\theta} + \dot{z}\hat{z}$$

Hence, the kinetic energy in our cylindrical coordinate system is given by

$$T = \frac{1}{2}\left(v^2 + \omega^2 r^2 + 2\vec{v}\cdot(\vec{\omega}\times\vec{r})\right) = \frac{1}{2}\left(v^2 + \omega^2 r^2 + 2(\dot{r}\hat{r} + r\dot{\theta}\hat{\theta})\cdot(\omega r\hat{\theta}) = v^2 + \omega^2 r^2 + 2\dot{\theta}r^2\omega\right) \tag{3.12}$$

Since the equation for the total velocity is valid for all time dependent $\omega_J(t)$, the kinetic energy from equation 3.12 has the same property.

3.3.3. Lagrangian

Now we have calculated the kinetic and potential energy, the Lagrangian $\mathcal{L}$ could be computed, which is given by the differences of these energies

$$\mathcal{L} = T - V = \frac{1}{2}m(v^2 + \omega_J^2 r^2 + 2\dot{\theta}r^2\omega_J) - V(r,\theta,z) \tag{3.13}$$

The Lagrangian produces a set of generalised momenta for the coordinates r, θ, z. They are given by

$$\begin{aligned} p &= \frac{\partial\mathcal{L}}{\partial r} = \dot{r} \\ l_z &= \frac{\partial\mathcal{L}}{\partial\theta} = r^2(\dot{\theta} + \omega_J) \\ p_z &= \frac{\partial\mathcal{L}}{\partial z} = \dot{z} \end{aligned} \tag{3.14}$$

These momenta, combined with the coordinates themselves, form a set of independent variables that fully describe the system. Therefore, we will use these momenta instead of the regular derivatives. Using these momenta, the Lagrangian can be rewritten as:

$$\begin{aligned} \mathcal{L} = \frac{p^2 + p_z^2 + r^2(\frac{l_z}{r^2} - \omega_J)^2 + \omega_J^2 r^2 + 2(\frac{l_z}{r^2} - \omega_J)r^2\omega_J}{2} - V(r,\theta,z) = \\ \frac{p^2 + p_z^2 + (\frac{l_z}{r} - \omega_J r)^2 + \omega_J^2 r^2 + 2(l_z\omega_J - \omega_J r^2)}{2} - V(x,y,z) = \\ \frac{p^2 + p_z^2 + \frac{l_z^2}{r^2} - 2\omega_J l_z + \omega_J^2 r^2 + \omega_J^2 r^2 + 2(l_z\omega_J - \omega_J^2 r^2)}{2} - V(r,\theta,z) = \\ \frac{p^2}{2} + \frac{p_z^2}{2} + \frac{l_z^2}{2r^2} - V(r,\theta,z) \end{aligned} \tag{3.15}$$

3.3.4. Hamiltonian

In general if there is a system with N degrees of freedom which can be fully described by N coordinates q_i, the Hamiltonian will generate a system of differential equations using the Lagrangian and the generalised momenta p_i corresponding to q_i. It is defined as[21]

$$\mathcal{H} = \sum_{i=1}^{N} p_i \dot{q}_i - \mathcal{L} \tag{3.16}$$

For our system with the coordinates from 3.6 and the generalised momenta from 3.14, this leads to a Hamiltonian of

$$\begin{aligned} \mathcal{H} = p\dot{r} + l_z\dot{\theta} + p_z\dot{z} - \mathcal{L} = p^2 + l_z(\frac{l_z}{r^2} - \omega_J) + p_z\dot{z} - \mathcal{L} = \\ \frac{p^2}{2} + \frac{p_z^2}{2} + \frac{l_z^2}{2r^2} - l_z\omega_J + V(r,\theta,z) \end{aligned} \tag{3.17}$$

Using this Hamiltonian, we can now finally compute the equations of motion as explained in[24]:

$$\begin{aligned} \dot{r} &= \frac{\partial \mathcal{H}}{\partial p} = p \\ \dot{\theta} &= \frac{\partial \mathcal{H}}{\partial l_z} = \frac{l_z}{r^2} - \omega_J \\ \dot{z} &= \frac{\partial \mathcal{H}}{\partial p_z} = p_z \\ \dot{p} &= -\frac{\partial \mathcal{H}}{\partial r} = \frac{l_z^2}{r^3} - \frac{\partial V}{\partial r} \\ \dot{l_z} &= -\frac{\partial \mathcal{H}}{\partial p} = -\frac{\partial V}{\partial \theta} \\ \dot{p_z} &= -\frac{\partial \mathcal{H}}{\partial p} = -\frac{\partial V}{\partial z} \end{aligned} \tag{3.18}$$

where the potential derivatives are given by:

$$\frac{\partial V}{\partial r} = G\left(\frac{m_S(r+\mu R\cos(\theta))}{(r^2+\mu^2R^2+2\mu rR\cos\theta+z^2)^{\frac{3}{2}}} + \frac{m_J(r-R\cos(\theta))}{(r^2+R^2-2rR\cos\theta+z^2)^{\frac{3}{2}}}\right)$$

$$\frac{\partial V}{\partial \theta} = G\left(\frac{m_S(-\mu Rr\sin(\theta))}{(r^2+\mu^2R^2+2\mu rR\cos\theta+z^2)^{\frac{3}{2}}} + \frac{m_J Rr\sin(\theta)}{(r^2+R^2-2rR\cos\theta+z^2)^{\frac{3}{2}}}\right)$$

$$\frac{\partial V}{\partial z} = Gz\left(\frac{m_S}{(r^2+\mu^2R^2+2\mu rR\cos\theta+z^2)^{\frac{3}{2}}} + \frac{m_J}{(r^2+R^2-2rR\cos\theta+z^2)^{\frac{3}{2}}}\right)$$

In the code for the numerical method, the equations of motions are rewritten mathematically to save computational time. All optimisations are summarised in section 6.1.1, but for the calculation of the potential derivatives the optimisation is very mathematical and therefore derived here. The idea is to define some variables and extract common base terms, to save computational time on 'heavy' functions like the fractional power. We therefore firstly defined $\rho = r\cos\theta, \psi = Rr\sin\theta$ which yields for the potential derivatives

$$\frac{\partial V}{\partial r} = \frac{Gm_S(r+\mu\rho)}{(r^2+\mu^2R^2+2\mu r\rho+z^2)^{\frac{3}{2}}} + \frac{Gm_J(r-\rho)}{(r^2+R^2-2r\rho+z^2)^{\frac{3}{2}}}$$

$$\frac{\partial V}{\partial \theta} = \frac{-Gm_S\mu\psi}{(r^2+\mu^2R^2+2\mu r\rho+z^2)^{\frac{3}{2}}} + \frac{Gm_J\psi}{(r^2+R^2-2r\rho+z^2)^{\frac{3}{2}}}$$

$$\frac{\partial V}{\partial z} = z\left(\frac{Gm_S}{(r^2+\mu^2R^2+2\mu r\rho+z^2)^{\frac{3}{2}}} + \frac{Gm_J}{(r^2+R^2-2r\rho+z^2)^{\frac{3}{2}}}\right)$$

Lastly, we extracted some common base terms, which ware defined as

$$B_1(r,r\rho,z) = \frac{Gm_S}{(r^2+\mu^2R^2+2\mu r\rho+z^2)^{\frac{3}{2}}}, B_2(r,r\rho.z) = \frac{Gm_J}{(r^2+R^2-2r\rho+z^2)^{\frac{3}{2}}}$$

This yields:

$$\frac{\partial V}{\partial r} = (r+\mu\rho)B_1 + (r-\rho)B_2$$

$$\frac{\partial V}{\partial \theta} = \psi(B_2 - \mu B_1)$$

$$\frac{\partial V}{\partial z} = z(B_1 + B_2)$$

In this way, the derivatives are no longer functions of θ but of ρ and ψ. In this way, we have to estimate less trigonometric functions, which is a relative time consuming calculation compared to a simple multiplication for example. Also, since we extracted a base term, we have to do many less operations per time step, and in particular we only have to calculate a fractional power once instead of three times, saving again on a heavy calculation.

3.4. Cartesian coordinates and generalised momenta

In the previous sections, the coordinate system was defined and the equations of motions where computed. However, as described in section 3.1, the standard way to describe a Trojans orbit is using Kepler elements and until now, we have only derived a map foam the Kepler elements to the Cartesian coordinates. The Cartesian coordinates thus have to be converted to the cylindrical coordinate system, but the map defined in equation 3.19 does not include the generalised momenta ans should therefore be extended in this chapter. Once the map is extended to all six coordinates in the cylindrical system, we are able to convert the Trojan data to our coordinate system using the method described in section 4.1. We will also convert the inclination from equation 3.24 to our coordinate system as we will need this formula at the end of our simulations. For this, we firstly have to derive the formula for the angular momentum in our coordinate system.

3.4.1. Full coordinate map

For the conversion from Cartesian coordinates to cylindrical ones, the elements in the z directions remain unchanged. To construct the full map, we thus only have to consider the x and y components of the position and velocity vector in Cartesian coordinates for which we will use a trick with complex numbers. Since the complex numbers are two dimensional, they can be seen as two dimensional vectors in the complex plane. Therefore we can represent the two dimension vector containing the x and y components of the three dimensional $\vec{r}$ vector as a complex number $\vec{r} := x + yi$. We can now use the rules for calculating with complex numbers.

By deriving the Cartesian position vector in 3.8 and comparing it with the equivalent definition above, the Cartesian velocity follows in terms of the cylindrical coordinates:

$$x + yi = re^{i\theta}$$

$$v_x + iv_y = \dot{r}e^{i\theta} + r\dot{\theta}ie^{i\theta} = pe^{i\theta} + (\frac{l_z}{r} - \omega_J r)ie^{i\theta} \tag{3.19}$$

From equation 3.19, one directly extracts $p_z = v_z$. We subtract these terms on both sided such that we lose the j terms and then multiply both sides with $e^{-i\theta}$ to obtain

$$(v_x + iv_y)e^{-i\theta} = \dot{r} + r\dot{\theta}i = p + (\frac{l_z}{r} - \omega_J r)i \tag{3.20}$$

where the generalised momenta from equation 3.14 are substituted for the derivatives. The conversions can now be found by comparing the real and imaginary parts:

$$p = \Re((v_x + v_y i)e^{-i\theta}) = v_x \cos(\theta) + v_y \sin(\theta), \quad l_z - \omega_J^2 = r\Im((v_x + v_y i)e^{-i\theta}) = v_x r\sin(\theta) + v_y r\cos(\theta))$$

Summarised, the full conversion is given by:

$$\begin{cases} r = \sqrt{x^2+y^2} \\ \theta = \text{atan2}\,(y,x) \\ z = z \\ p = v_x \cos(\theta) + v_y \sin(\theta) \\ l_z = v_x r \sin(\theta) + v_y r \cos(\theta)) - \omega_J^2 \\ p_z = v_z \end{cases} \tag{3.21}$$

3.4.2. Angular momentum

The angular momentum is not directly part of the research, but it is an important quantity that is used in the derivations of the inclinations and the symmetries. Therefore we will calculate the angular momentum here. When calculating the angular momentum L, one has to consider the velocity in the non-rotating frame. Therefore, one could not invert equation 3.21 directly, but firstly has to to add ω_J to the differential equation for the θ coordinate to obtain $\dot{\theta}' = \frac{l_z}{r^2}$ in equation 3.18. One does not have to adjust anything about the position vector and the other velocity coordinates since these could just be considered as a choice of axis, which in our case thus are the axes such that Jupiter and the Sun lie on the x-axis, moving in the x,y-plane. Now the vector from equation 3.8 and the first three differential equations from the equations of motion 3.18, can be used to obtain the following conversion needed for the specific calculation of the angular momentum:

$$\begin{cases} x = r\cos\theta \\ y = r\sin\theta \\ x = z \\ \dot{x} = \dot{r}\cos\theta - \dot{\theta}' r\sin\theta = p\cos\theta - \frac{l_z}{r}\sin\theta \\ \dot{y} = \dot{r}\sin\theta + \dot{\theta}' r\cos\theta = p\sin\theta + \frac{l_z}{r}\cos\theta \\ \dot{z} = p_z \end{cases} \tag{3.22}$$

Hence one arrives at

$$\begin{gathered} \vec{L} = \vec{r}\times\vec{p} = \begin{vmatrix} \vec{x} & \vec{y} & \vec{z} \\ r\cos\theta & r\sin\theta & z \\ p\cos\theta - \dfrac{l_z}{r}\sin\theta & p\sin\theta + \dfrac{l_z}{r}\cos\theta & p_z \end{vmatrix} = \\ \left(rp_z\sin\theta - z\left(p\sin\theta + \frac{l_z}{r}\cos\theta\right)\right)\widehat{x} + \left(z(p\cos\theta - \frac{l_z}{r}\sin\theta) - rp_z\cos\theta\right)\widehat{y} + \\ \left(r\cos\theta(p\sin\theta + \frac{l_z}{r}\cos\theta) - r\sin\theta(p\cos\theta - \frac{l_z}{r}\sin\theta)\right)\widehat{z} = \\ \left(rp_z\sin\theta - z\left(p\sin\theta + \frac{l_z}{r}\cos\theta\right)\right)\widehat{x} + \left(z(p\cos\theta - \frac{l_z}{r}\sin\theta) - rp_z\cos\theta\right)\widehat{y} + l_z\widehat{z} \end{gathered} \tag{3.23}$$

3.4.3. Inclination

Using the angular momentum from equation 3.23, one can express the inclination from equation 3.24 in the cylindrical coordinates. For this, we firstly calculate some properties about the angular momentum vector from equation 3.23:

$$\begin{gathered} L_x^2 + L_y^2 = r^2p_z^2\left(\sin^2\theta + \cos^2\theta\right) + z^2\left(p\sin\theta + \frac{l_z}{r}\cos\theta\right)^2 + z^2(p\cos\theta - \frac{l_z}{r}\sin\theta)^2 + \\ -2rzp_z\left(\sin\theta\left(p\sin\theta + \frac{l_z}{r}\cos\theta\right) + \cos\theta\left(p\cos\theta - \frac{l_z}{r}\sin\theta\right)\right) = \\ r^2p_z^2 + z^2p^2\left(\sin^2\theta + \cos^2\theta\right) + z^2\frac{l_z^2}{r^2}\left(\cos^2\theta + \sin^2\theta\right) + 2\frac{z^2 l_z p}{r}\left(\sin\theta\cos\theta - \cos\theta\sin\theta\right) = \end{gathered}$$

$$r^2p_z^2 + z^2p^2\left(\sin^2\theta + \cos^2\theta\right) + z^2\frac{l_z^2}{r^2}\left(\cos^2\theta + \sin^2\theta\right) + 2\frac{z^2 l_z p}{r}\left(\sin\theta\cos\theta - \cos\theta\sin\theta\right) =$$

$$-2rzp_z(p\left(\sin^2\theta + \cos^2\theta\right) + \frac{l_z}{r}\left(\sin\theta\cos\theta - \cos\theta\sin\theta\right)) =$$

$$r^2p_z^2 + z^2p^2 + z^2\frac{l_z^2}{r^2} - 2rzp_zp = (rp_z - zp)^2 + \frac{z^2l_z^2}{r^2}$$

We now realise that we will obtain less operations, and thus a faster code, if we use geometry to rewrite the inclination in terms of an arc tangent to obtain:

$$I = \arccos\frac{L_z}{|\vec{L}|} = \arctan\frac{\sqrt{L_x^2 + L_y^2}}{L_z} = \arctan\frac{\sqrt{(rp_z - zp)^2 + \frac{z^2}{r^2}l_z^2}}{l_z} = \arctan\sqrt{\left(\frac{rp_z - zp}{l_z}\right)^2 + \left(\frac{z}{r}\right)^2} \tag{3.24}$$

3.5. L4 and L5 points

Now all important quantities and coordinate transformations are set up, we can investigate the Lagrange points more closely. Since the stable L4 and L5 points are central in this research, we will first prove that they indeed exist as equilibrium points. After then, we will define, prove and discuss the symmetry of those points. Lastly, we will discuss the asymmetry for a migrating Jupiter.

3.5.1. Symmetry

Before we dive into a symmetry between the L4 and L5 point, we first note that there is also a simple symmetry within each point by mirroring a particle in the x, y-plane, thus inverting the z-directions. That is, in cylindrical coordinates:

$$r' = r, \theta' = \theta, z' = -z, p' = p, l_z' = l_z, p_z' = p_z \tag{3.25}$$

where the prime denotes the coordinates after mirroring the original Trojan. This symmetry is trivial since the equations of motion are symmetric in the z-direction even if Jupiter's eccentricity or the migration is taken into account. because all z dependent terms in the equations of motion 3.18 are of the form z^2, making the sign irrelevant. We will therefore refer to this as the trivial symmetry and only take into into account for some results. In this way, we are able to show it indeed does not have an effect in the numerical analysis as well, but we prevent unnecessary computational times by leaving out this irrelevant symmetry.

More interesting is that on a first glance the L4 and L5 point appear to be symmetric in the rotating frame. Before we are able to prove this symmetry does indeed exist, we have to define what we consider as a symmetry. We define this symmetry that if an orbit exist around L4, we can define a Trojan such that its orbit has exactly the same shape around L5. Thus, we must be able to mirror the orbit in the y-axis. Before we will prove mathematically that this is indeed possible, it will be thought of intuitively how the prove should look like. If the orbits are mirrored in the x-axis, every possible position vector on the orbit r must be mirrored in the y-axis. But if we mirror an arbitrary vector $\vec{r}$ at some moment in time, it is assured that any moment in time later the position vectors still is mirrored position vector if and only if they change in the same way. In other words, there slopes or derivatives should be related. The derivatives are given by the velocity vector and thus we expect these to be mirrored in the x-axis as well. Since the slope is 'two-directional', we also have the option to flip the velocity after wards. In the letter case, the mirrored Trojan thus moves the other way around. The latter is now assumed, since we know all Trojans move anti clockwise. This intuition will help us to find a proposed solution, from which we could later prove it is indeed valid. It also tells us that the velocity is an important measure in this proof.

To execute the mathematically, we firstly start with some definitions. Without loss of generality start at time $t = 0$. Since we have a physical problem, we must have unique solutions of the orbits as functions of time. Therefore, we can parameterise every variable by the time. Call the position vector in the original frame $\vec{r}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}$ with corresponding velocity vector $\vec{v}(t) = \begin{bmatrix} v_x(t) \\ v_y(t) \\ v_z(t) \end{bmatrix}$. After mirroring the position in the x-axis, we call the position vector $\vec{r'}(t)$ and velocity vector $\vec{v}'(t)$. In the rotating frames, the cylindrical coordinates are called $r(t), \theta(t), z(t), p(t), l_z(t), p_z(t)$ for the original particle and $r'(t), \theta'(t), z'(t), p'(t), l'_z(t), p'_z(t)$ for its mirror. For simplicity, if we look at time $t = 0$, the t dependence is not written down thus for example $r(\vec{0}) = r(\vec{t}) = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$

We have to prove that after mirroring the Trojans stay mirrored at all time. However, as explained in the first two paragraphs, they go around the orbits in different directions. In other words, if initially $\vec{r'} = \begin{bmatrix} x \\ -y \\ z \end{bmatrix}$, we expect that the solution is given by the symmetric orbit:

$$\forall t \in \mathbb{R}, r'(\vec{t}) = \begin{bmatrix} x(-t) \\ -y(-t) \\ z(-t) \end{bmatrix} \tag{3.26}$$

Now we know that physical systems must have a unique solution, thus it is sufficient to just assume that the orbit in equation 3.26 exist and than prove that it is a solution to the system. The uniqueness then implies that it must be the orbit after mirroring and thus that the orbit is indeed symmetric with the original one. We could prove the solution is a solution, by showing that it satisfies the equations of motion.

Therefore, we now start by assuming equation 3.26. Since we already know the initial conditions for the position, our intuition suggest that the velocity now determines the orbit. We therefore start by computing it by deriving $\vec{r'}(t) = \begin{bmatrix} x(-t) \\ -y(-t) \\ z(-t) \end{bmatrix}$. It follows that

$$\vec{v}'(t) = \begin{bmatrix} -v_x(-t) \\ v_y(-t) \\ -v_z(-t) \end{bmatrix} \tag{3.27}$$

Plugging in $t = 0$ yields three other initial conditions (which agree with the intuitions from the first two paragraphs) for the velocity vector $\vec{v}'$ after mirroring. Combined with the $\vec{r'}(t)$ function, this can be converted to six independent equations for the coordinates of the cylindrical system. Therefore, they describe the entire system and thus form a unique solution if they all six obey the equations of motion. Furthermore, it provides initial conditions for the generalised momenta, which are needed when we want to simulate a mirrored Trojan. Since the equations of motions are in the cylindrical system, it is easier to now rewrite the found vectors in cylindrical coordinates. For the positional elements. this is trivial: $\begin{bmatrix} r'(t), \theta'(t), z'(t) \end{bmatrix} = \begin{bmatrix} r(-t), \theta(-t), z(-t) \end{bmatrix}$. For the general momenta, we observe from the vectors in 3.26 and 3.27 that the angular momentum components $L'_x(t)$ and $L'_z(t)$ are time mirrors of the original Trojans L_x and L_z as well. Thus from 3.23 it must follow that:

$$\begin{cases} L'_z(t) = L_z(-t) \Rightarrow l'_z(t) = l_z(-t) \\ L'_x(t) = L_x(t) \Rightarrow r'(t)p'_z(t)\sin(\theta'(t)) - z'(t)\left(p'(t)\sin(\theta'(t)) + \dfrac{l'_z(t)}{r'(t)}\cos(\theta'(t))\right) = \\ \qquad r(-t)p_z(-t)\sin(\theta(-t)) - z(-t)\left(p(-t)\sin(\theta(-t)) + \dfrac{l_z(-t)}{r(-t)}\cos(\theta(-t))\right) \end{cases} \tag{3.28}$$

If we now plug in $r'(t) = r(-t), \theta'(t) = -\theta(-t), z'(t) = z(-t), l'_z(t) = l_z(-t)$ in the bottom equation, this yields

$$\begin{aligned} -r(-t)p'_z(t)\sin(\theta(-t)) - z(-t)\left(-p'(t)\sin(\theta(-t)) + \frac{l_z(-t)}{r(-t)}\cos(\theta(-t))\right) = \\ r(-t)p_z(-t)\sin\left(\theta(-t)\right) - z\left(p(t)\sin(\theta(-t)) + \frac{l_z(-t)}{r(-t)}\cos(\theta(-t))\right) \end{aligned} \tag{3.29}$$

Since this must hold for all combinations of coordinates for the non-mirrored particle, it follows that the only solution is

$$r'(t) = r(-t), \theta'(t) = -\theta(-t), z'(t) = z(-t), p'(t) = -p(-t), l_z'(t) = l_z(-t), p_z'(t) = -p_z(-t) \tag{3.30}$$

Since this is the unique solution to 3.29, we have found that this set solutions agrees with the sets of solutions 3.26 in the Cartesian system, where the initial conditions are found by plugging in $t = 0$:

$$r' = r, \theta' = -\theta, z' = z, p' = -p, l_z' = l_z, p_z' = -p_z \tag{3.31}$$

As explained, we are thus are only left to prove that equation 3.30 is a solution to the differential equation with these initial conditions. This is done by showing that it satisfies the equations of motion 3.18. Indeed,

$$\begin{aligned}
\dot{r}'(t) &= -\dot{r}(-t) = -p(-t) = p'(t) \\
\dot{\theta}'(t) &= \dot{\theta}(-t) = \frac{l_z(-t)}{r^2(-t)} - \omega_J = \frac{l_z'(t)}{r'^2(t)} - \omega_J \\
\dot{z}'(t) &= -\dot{z}'(-t) = -p_z(-t) = -p_z(t) \\
\dot{p}'(t) = \dot{p}(-t) &= \frac{l_z^2(-t)}{r^3(-t)} - \frac{\partial V}{\partial r}(-t) = -\frac{l_z'^2(t)}{r'^3(t)} - \frac{\partial V}{\partial r'}(t) \\
\dot{l_z}'(t) &= -\dot{l_z}(-t) = -\frac{\partial V}{\partial \theta}(-t) = \frac{\partial V}{\partial \theta'}(t) \\
\dot{p_z}'(t) &= \dot{p_z}(-t) = -\frac{\partial V}{\partial z}(-t) = -\frac{\partial V}{\partial z'}(t)
\end{aligned} \tag{3.32}$$

Therefore we conclude that the proposed solution indeed gives an equivalent orbit, albeit the time is reversed. From this we conclude that the L4 and L5 point are indeed symmetric and we can mirror a Trojan by using the map

$$r' = r, \theta' = -\theta, z' = z, p' = -p, l_z' = l_z, p_z' = -p_z$$

This symmetry will later be exploited in the method to be able to create a system with just as many L4 Trojans and L5 Trojans from any data set.

It has to be noted that the symmetry is broken in the actual system because of for example gravitational permutations of Saturn and by Jupiter's actual non-zero eccentricity. However, this leads to an asymmetry of about 10% over a time of 4,5 billion years[3], which is a $4.5 \cdot 10^3$ times longer than the maximal simulated time in this project. Therefore, it is acceptable to neglect these effects on the time scales used in this project.

To give some more insight before we compare this result with the migrating case, we will show in figure 3.3 the plot for the stability regions of both points Li et al. found using Sicardy and Dubois theory[20]. This plot shows that the L4 and L5 have symmetric stability regions, which is necessary if the orbits are symmetric. 3.3:

3.5.2. Asymmetry L4 and L5

As described in the introduction, this symmetry no longer exist for the migrating Jupiter. If the symmetry form equation 3.26 would still exist, the equations of motion must still be satisfied for this case. However, if we now compute $\dot{\theta}$, we must consider $\omega_J(t)$ as a function of time, since $\omega_J = \sqrt{\frac{Gm_S}{R^3(t)}}$ and R is now a function of time.

$$\dot{\theta}'(t) = \dot{\theta}(-t) = \frac{l_z(-t)}{r^2(-t)} - \omega_J(-t)$$

But since $R(-t) \neq R(t)$ and both the square root and the third power are an injective function, we find that it cannot hold that $\omega_J(-t) = \omega_J(t)$. Therefore, equation 3.30 is not a solution to the equations of

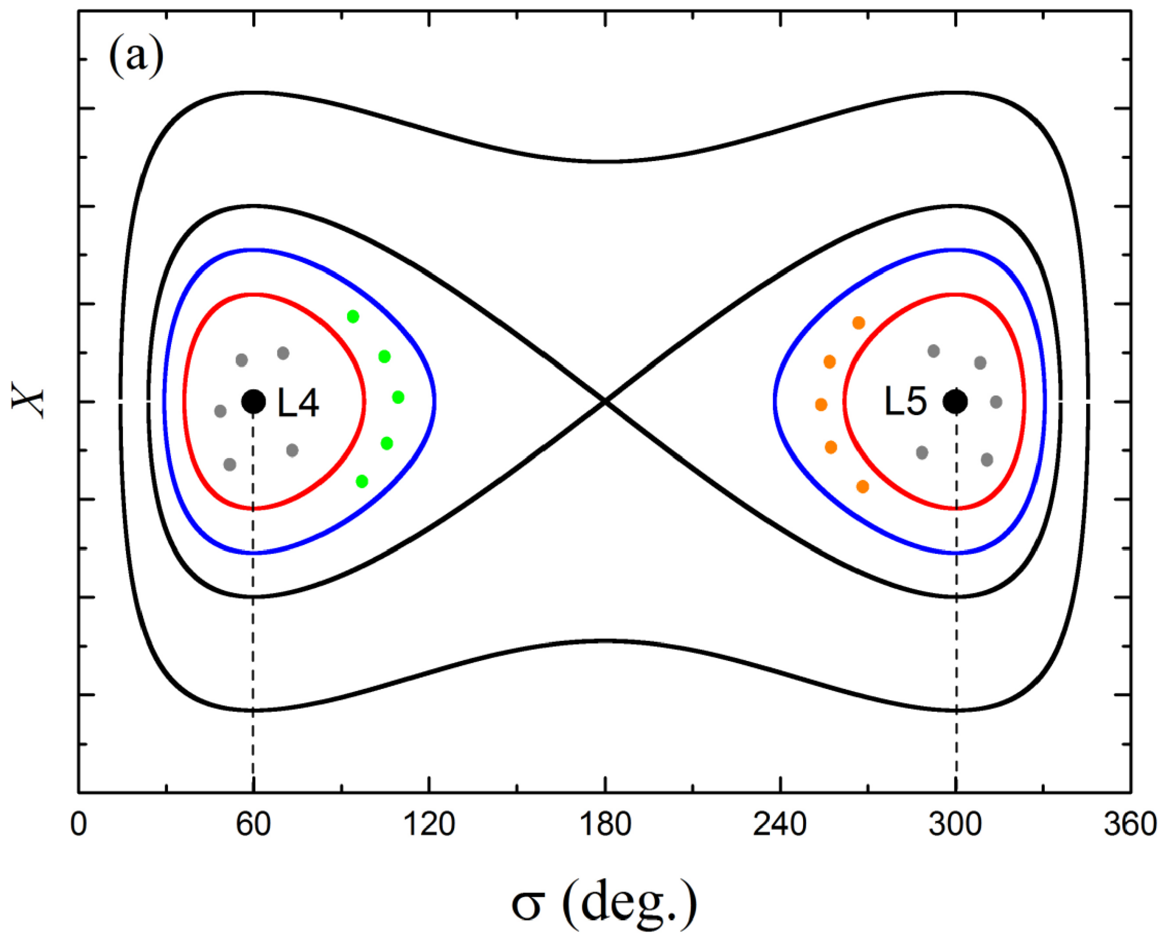


Figure 3.3: The stability regions of an L4 and an L5 Trojan as found by Li et al.[2]. X is proportional to the distance between a Trojan and Jupiter and σ the angle in the rotating frame. The blue lines are the stability boundaries and the red line denotes the bundary for very stable Trojans. The dots represent some test particles used by Li et al.

motion in this case and the symmetric orbits no longer exist.

Li et al. again used Sicardy and Dobois theory[20] to plot the stability regions for the L4 and L5 point during the migration of Jupiter. These plots are shown in figure 3.4. We deduce that the stability region has decreased for the L4 point, while it increases around the L5 point. The entire proof can be written in [20], but we can make the part for large angles θ intuitive using our coordinate system. Suppose that we have an L4 Trojan at the border of this region with a large angle θ. Then, in the normal case, the forces pushing the Trojan in to the region and out of the region must cancel each other out. The equation of motion for $\dot{\theta}$ thus gives $\frac{l_z}{r^2} = \omega_J$. But now, if we migrate Jupiter, ω_J decreases, thus the force that pulls the Trojan into the stable region decreases, resulting in the Trojan being pushed out. For the L5 Trojan, on the other hand, this is equivalent to the force pushing the Trojan out of the stable region being decreased. Hence, we expect the boundary with balance $r^2 = \omega_J$ to be further removed from the L5 point. However, these derivations only hold for all moments in time when r and l_z remain the same for both particles at all time.

Li et al. described that one might expect intuitively that the shrunken L4 suitability region leads to the outer L4 Trojans being left unstable during the migration. To test this, they picked an L4 Trojan at the boundary of the L4 region in the normal case (with $\theta > 60^\circ$) and a similar particle with this angle mirrored. For this, they defined the resonance angle as the maximum angular deviation from the Lagrange point. This will be an important quantity for our research as well and be denoted by the symbol $\Delta\theta$. The results Li et al found for the test particles and their resonant angles are shown in figure 3.5.2. They concluded

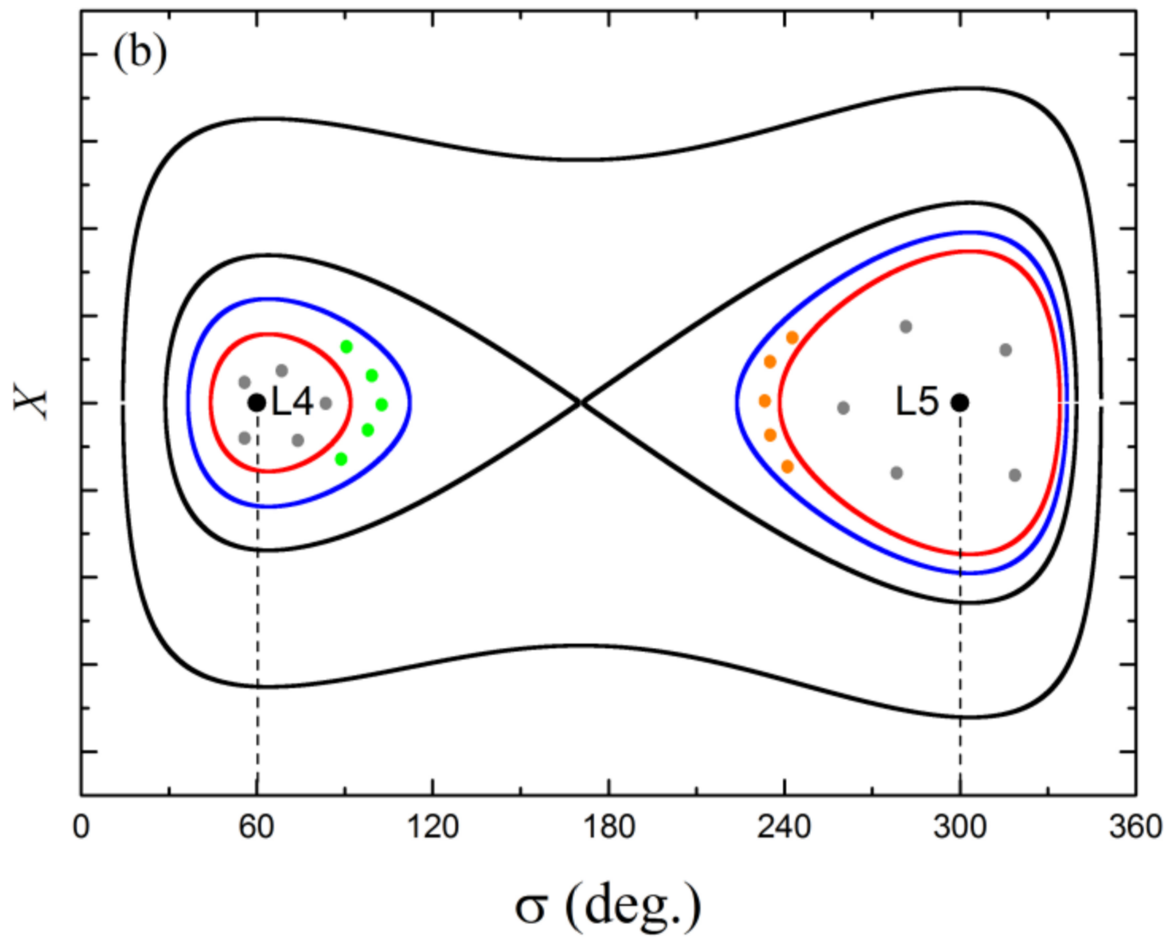


Figure 3.4: The stability regions of an L4 and an L5 Trojan as found by Li et al.[2]. X is proportional to the distance between a Trojan and Jupiter and σ the angle in the rotating frame. The blue lines are the stability boundaries and the particles between the blue and red lines have high angular and radial deviations. One observes that the stability region of the L4 point is smaller than that of the L5 point. The dots represent some test particles used by Li et al, that are altered due to the migration.

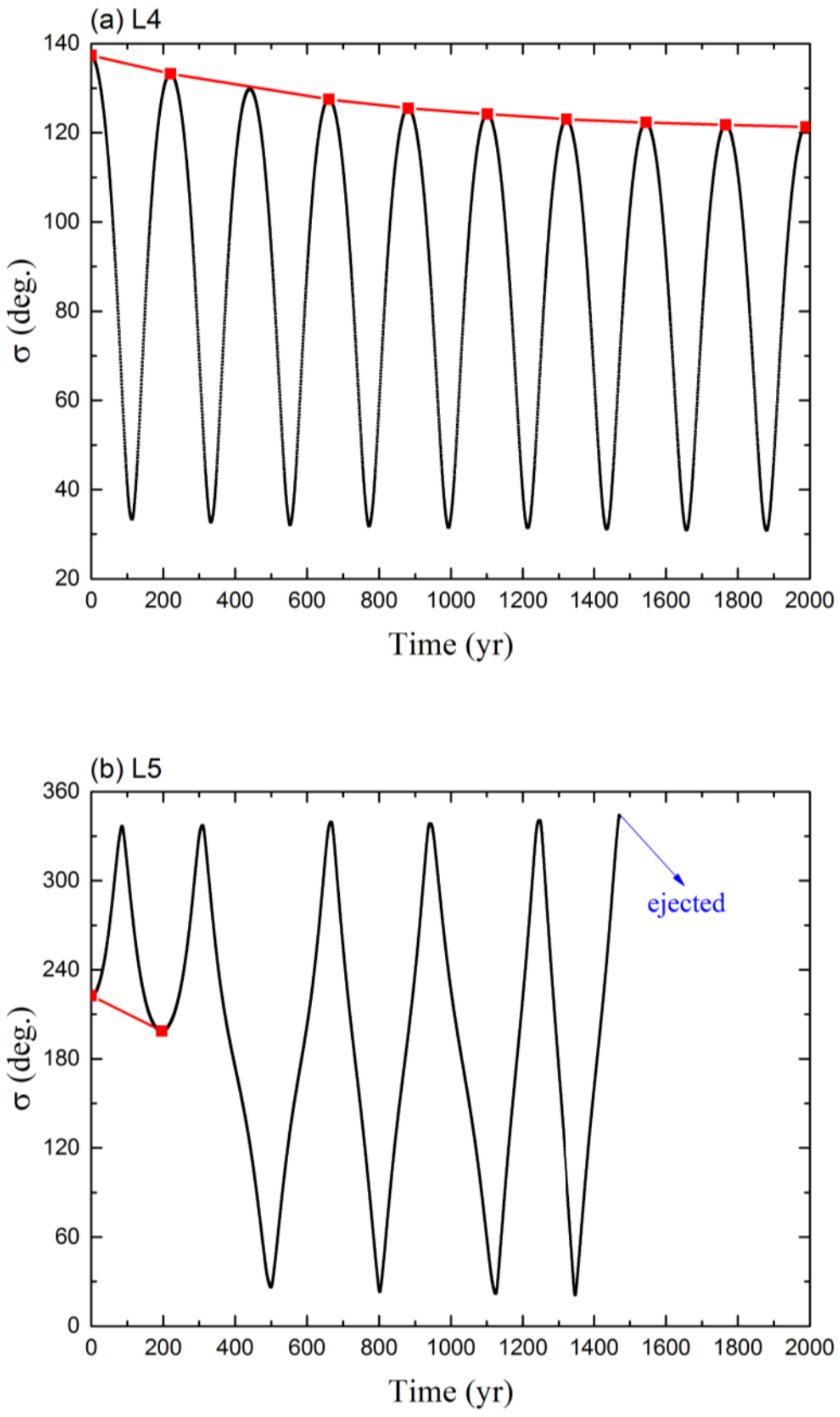


Figure 3.5: Evolution a Trojan resonant angle with initial resonant angle 137° during a migration of Jupiter. Both the plot for a L4 and a L5 Trojan are shown. σ is the angle θ in our report. Source: [2]

that in this case, the resonant angle for the L4 Trojan in fact decreased, while that of the L5 Trojan increased and eventually got ejected. They suggested that during the migrations, it might actually be the case that the shrinking L4 regions pushes the Trojans inwards, leaving more stable Trojan at the end of the migration, while the increasing L5 region pushes Trojans outwards, potentially pushing them into unstable regions.

The fact that the shrunken stability leads to more stable L4 Trojans leaves the question where the loop hole is. In fact, the stability regions plotted in 3.3 and 3.4, are only domains in position space. It thus only accounts for potential energies and an initial kinetic energy. However, the migration can be described by a force on the system over a certain distance. In other words, there is extra work performed on the Trojan and the sum of the kinetic and potential energies is not be conserved. In fact we see from our equations of motion that all derivative functions $\dot{p}$, $\dot{l_z}$ and $\dot{p_z}$ alters compared to the non-migrating case, because the potentials are now time dependent. We can see this as some extra force altering the Trojans velocities. These forces might cause the Trojan being pushed to or from the stable regions during the migration. It is therefore possible that the migration pushes an L4 Trojan towards the stable region, while its L5 mirrors are pushed out of this region.

Nonetheless, it is difficult to derive whether the shrinking stability regions always push L4 Trojans inwards or not. Li et al. tested it for a small number of Trojans with certain initial conditions, but did not consider for example the inclination in most simulations. For the inclinations it is more difficult to see what happens, but since the the distribution of the L4 and L5 Trojans are altered in the θ directions by the migration in a different manner, the potential energy alters as well leading to different velocities in the z-directions. Therefore, we hypothesize the migration leads to an observable difference in inclinations.

4 Method

In order to answer how the Trojan distributions are effected by the migration, we need a model to describe the migration of Jupiter and we need a code that is able to simulate the model. We decided to create our own code since this allows us to use our coordinate system and to optimise the code for this specific research. In order to execute the code, we will need to pick an appropriate numerical method. However, before we look into the model and method in detail we will first discuss the initial data. The initial Trojans are not created, but extracted from real Trojan data such that we obtain a large data set of initial conditions. Therefore, we have to describe how we can implement this initial data correctly in our coordinate system.

4.1. Initial data

The initial Trojan data was downloaded from NASA's small object database[14], which contained 12.555 Trojans. In this way, we have many Trojans with different initial conditions to start our simulations from. The NASA data is given in Kepler elements, thus we have to convert them to our coordinate system using the conversions described in subsection 3.1.1. Then, we use formula 3.31 to obtain as many L4 and L5 Trojans and to make sure that these copies are symmetric, such that the data set is increased to a set of 25.110 Trojans with just as many L4 and L5 Trojans. Also, we have ensured that each Trojan has a copy in the L4 and L5 point and vice versa. We will refer to this as the symmetric copy or mirror, although this symmetry is broken once the migration starts. In section 3.5, we have also written about the trivial symmetry. We generated the Trojans described by this symmetry using formula 3.25, but we will not use this copies in all simulations to save computational time.

Apart from the Trojan data, we need some data of Jupiter to be able to execute the conversion above. The data used is given in table 4.1.

Quantity	Corresponding symbol or equation	value
Mass	m_J	$1,898.13 \cdot 10^{24}$ kg
Semi-major axis	a_J	5.20336301 AU
Eccentricity	ϵ	0.04839266
Inclination	I	$1.30530°$
Longitude of ascending node	Ω_J	$100.55615°$
Longitude of perihelion	$\omega_J + \Omega_J$	$14.75385°$
Mean Longitude	$M_J + \Omega_J + \omega_J$	$34.40438°$
Epoch	E_J	J2000 (2451545.0 days)

Table 4.1: Current data of Jupiter needed to calculate positions of Trojans[13]

It is worth noting that as a first attempt, it was assumed that Jupiter had a small influence on the Trojans in the time elapsed between the different epochs. This simplifies some of the calculations by using the numerical method and allows all simulations to start from the same moment in time, but of course excluding Jupiter introduces other errors. In our attempt, it resulted to about 36% of the Trojans being unstable after a simulation of 10000 years without migration, compared to less than a percent using the current method. The alternative method is however described in the appendix C.

4.2. Numerical method

For numerically integration of Hamiltonian systems, one usually uses a symplectic integrator to maintain the symplectic property as much as possible. The exact meaning of the symplectic property is beyond the scope of this project, but a standard implicit symplectic integrator has a bounded energy error[8] which is important since the Hamiltonian is conserved physically when there is no implicit time dependence in its formula. Apart from the symplectic property, a high order method is preferred, since one wants to use bigger time steps in order to be able to simulate many years in a short amount of time. The Yoshida integrator (A.1) was therefore considered, but our non-migrating system was not exactly of the required form. A Richardson estimation estimated the error to be of a second order instead of the desired fourth order. However, Luo et. al [8], improved the integrator to a fourth order method, by duplicating the system each time step, where in one system the position is constant and in the other the velocity, and averaging out the results after each time step. For time independent separable Hamiltonians, that is the Hamiltonian can be written as the sum of the time independent kinetic energy and the potential energy, this integrator coincides with the original Yoshida integrator[8] (OYI). This improved method will therefore be called the Midpoint Yoshida Integrator (MYI) in this report. They claim the method is symplectic-like, thus yielding a bounded Hamiltonian error and claim it is mostly useful for the long term evolution of problems with non-separable Hamiltonians[8], like ours. The method is as follows:
Firstly, one copies the canonical coordinates and momenta vectors $\vec{r}$ and $\vec{p}$, call these variables $\tilde{r}$, $\tilde{p}$. Then one sets a new Hamiltonian $\mathcal{H}(\vec{r},\vec{p},\tilde{r},\tilde{p}) = \mathcal{H}_1(\vec{r},\tilde{p}) + \mathcal{H}_2(\tilde{r},\vec{p})$, where $\mathcal{H}_1$ and $\mathcal{H}_2$ are copies of the original Hamiltonian. Then it follows that:

$$\dot{\vec{r}} = \vec{\nabla}_p\mathcal{H} = \vec{\nabla}_p\mathcal{H}_2(\tilde{r},\vec{p}), \qquad \dot{\tilde{p}} = -\vec{\nabla}_{\tilde{r}}\mathcal{H} = -\vec{\nabla}_{\tilde{r}}\mathcal{H}_2(\tilde{r},\vec{p})$$

Since H_2 is independent of $\vec{r}$ and $\tilde{p}$, these equations, merged in one vector $[\vec{r},\tilde{p}]$ can be solved and we obtain the approximation for a time step h

$$[\vec{r},\tilde{p}] \approx [\vec{r},\tilde{p}] + h[\vec{\nabla}\mathcal{H}_2(\tilde{r},\vec{p}), \vec{\nabla}\mathcal{H}_2(\tilde{r},\vec{p})]$$

which Junhie Luo et al. call the operation $\mathbf{H_1}(h)$ (bold notation is used here for the vectors instead of arrays to improve readability in combination with tilde symbol). Similarly, the operation $\mathbf{H_2}(h)$ approximates the solution for $[\tilde{r},\vec{p}]$. Now set the operation

$$\mathbf{A_2}(h) = \mathbf{H_2}(h/2)\mathbf{H_1}(h)\mathbf{H_2}(h/2)$$

Then, the approximation with time step dt for $[\vec{r},\vec{p}]$ is obtained by applying the operation $\mathbf{A_2}(d_1dt)\mathbf{A_2}(d_2dt)\mathbf{A_2}(d_1dt)$ to the system $(\vec{r},\vec{p},\tilde{r},\tilde{p})$ and then averaging the results for $[\vec{r},\tilde{p}]$ and $[\tilde{r},\vec{p}]$. d_1 and d_2 are here the same constants as in A.1. Junjie et al. claim this methods gives a symplectic-like integrator, which should suppress the unbounded energy changing behaviour of the OYI. The integrators are compared in section 6.1. To determine a correct time step, one has to calculate the stability region

of the numerical method and the eigenvalues of the Jacobian of the system. Using the stability analysis from appendix A.2, we have plotted the stability region of the numerical method and found a sufficient bound of $dt \leq \frac{2.5}{|\lambda|}$ provided $\Re(\lambda) \leq 0$ as shown in figure 4.1 The eigenvalues could be determined algebraically as described in appendix B.3 so we do not have to use any rude estimations. This is of great advantage, as we would like as big time steps as possible in order to be able to run simulations of a million of years. Combing the eigenvalues with the stability condition above one finds a bound for the time step $dt \leq 4.74$ y. However, if the Trojan is further removed from the equilibrium point, one might need a smaller time step. Also, during the migration there are no equilibrium points in the chosen inertial frame, potentially requiring a smaller time step. Therefore we take a much smaller time step of 1/3 years. The slow migration rate suggests this should be small enough. The errors could be estimated for testing whether a chosen time step is sufficient or not, but this is left for further research on the numerical method.

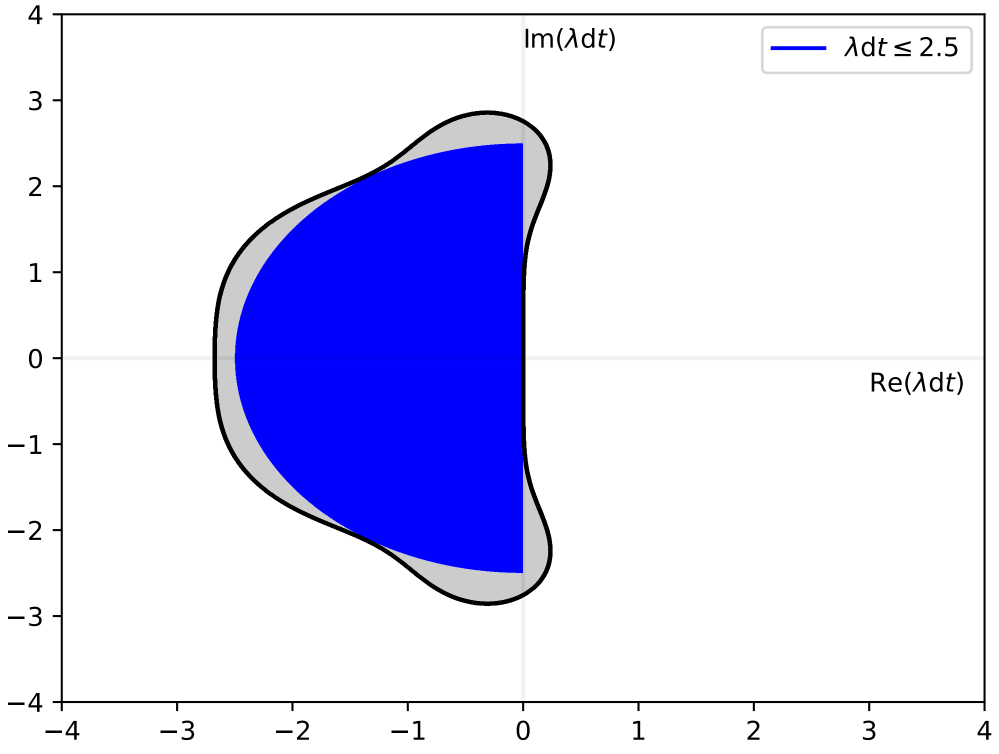


Figure 4.1: Stability region for the midpoint Yoshida integrator with sufficient stability condition

4.2.1. Migration implementation

According to Malhorta[9], the migration of Jupiter can be estimated as

$$R(t) = R_{\text{initial}} + \left(1 - \exp\left(-\frac{t}{\tau}\right)\right)\Delta R \tag{4.1}$$

Here we define $\tau = \frac{10}{3}T$ where T is the time at which Jupiter has travelled about 96% of the total migration distance, see figure 4.2. The total migration duration is then comparable to a constant migration over a time T. Furthermore, our definition is now consistent with the migration used by Li et al.[2], which allows us to use their results as estimates of values for parameters. Although Li et al. described the migration using a force on Jupiter in the direction of its velocity[2], we do not simulate Jupiter and can simply substitute formula 4.1 together with a formula for the angular velocity into the equations of motion. This follows from the fact that when R_J and ω_J are functions of the time, formulas 3.12 and 3.9 are still valid as explained in these sections. Therefore the equations of motion which were derived using these energies are still valid. The difference is that the Hamiltonian is no longer conserved, but this is not a requirement for the equations of motion. For the case when Jupiter's orbit is a circle, the formula for the angular velocity is given by $\omega_J = \sqrt{\frac{Gm_S}{R^3}}$. The initial value chosen for R is Jupiter's current semi-axis from table4.1 and we have a simulation time of one million years. We expect that it takes some time for the effect to become visible.

Since R_J and ω_J are known functions, we have to choose a time at which they are evaluated during each time step. We have chosen to pick the value it has at the mean time of the interval.

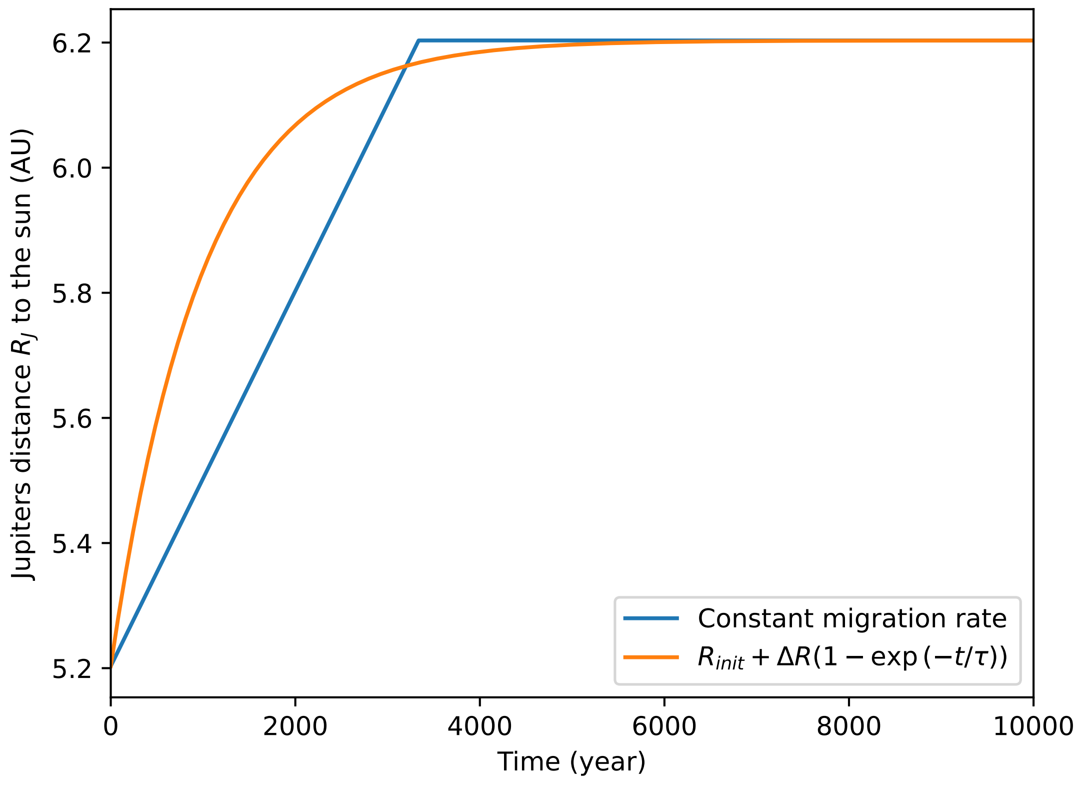


Figure 4.2: The exponential function used to describe Jupiter's radial distance to the sun during an outward migration of 1 AU. If we compare the graph with a constant migration of 1 AU over a time T, we can define $\tau = \frac{10}{3}T$. From the plot follows that in that case Jupiter has travelled about 96% of its total migration distance with the exponential description and therefore the duration of both migrations are comparable. For this plot $\tau = 1000$ is used.

4.3. Simulations and code implementation

After the initial data is set up using section 4.1, we first ran simulations of a hundred thousand years to determine the Trojan types on the long term without migration. Also, we run small simulations of ten thousand years to determine the maximum and minimum angular deviations from the Lagrange point before the migration. This has to be executed once. After that, several simulations can be run with different parameters for the migration. An important note is that saving the orbits is not possible. Even when the data is compressed, a simulation of a hundred thousand years would lead to several gigabytes of data for our large data set. One therefore has to calculate the quantities needed directly after a simulation has ended. In our case, this are the final inclination and the type of Trojan. In words, the used scheme is as follows:

1. Load the converted Trojan data into one vector with the six initial conditions for our coordinate system.
2. Make symmetric copies of these Trojans.
3. Update the time to $dt/2$ where dt is the time step.
4. Calculate the new value of R_J using formula 4.1.
5. Calculate the new angular velocity ω_J using the new R_J.
6. Perform one step of the midpoint Yoshida integrator.
7. Add dt to the time.
8. Repeat 4-7 until the total simulation time is reached.
9. Calculate the inclination at the last moment of the simulation.
10. Determine the Trojan type.

The simulations themselves (step 2 up and till 7) are executed in a C code. This is very fast, especially after we implemented several optimisations. The code is shown in appendix E.1. To handle the initial data and code easily, we created a Python library with several functions and classes as shown in appendix E.2. These can be called from Python function, as in the example code from appendix E.5. In this example we used multi threading to run several simulations simultaneously. This is of great importance to allow the required long simulations for the large numbers of Trojans. In the discussion section 6.1.1, we have elaborated more details on our code performance and efficiency. Step 9 is in fact inaccurate, since the inclination is not constant. However, if the inclination is a slowly changing variable after the migration, this is a valid and time efficient estimation. This is taken as an assumption in this report.

Jupiter's orbital eccentricity

Although our initial research did not include Jupiter's orbital eccentricity, the effect is investigated and elaborated in the discussion. However, the implementation in the model is not trivial and therefore elaborated here. In the code, Jupiter's orbital eccentricity will only be considered in the migrating case and will still be ignored in the non-migrating case for time saving and simplicity, since the effect is already known to be small on our timescale. We will however derive the equations for the non-migrating case, since we had the requirement that Jupiter always is on a Kepler orbit during the migration.

When Jupiter's orbit has an eccentricity this can be seen as oscillations of Jupiter in the rotating frame. We have already discussed in the main section that a time dependent R_J does not change the equation of motion. However, we choose as a condition that Jupiter remains on the x-axis, since otherwise the formula for the potential derived in subsection 3.9 is invalid. This could be solved by making the angular velocity ω of the rotating frame time dependent in such a manner, that Jupiter is always on the x-axis in this frame. This is allowed in our problem, since we have already proven that the equations of motion are valid for time dependent angular velocities. We thus only have to find such an ω. For the derivation of what the angular velocity should be, we consider the rotation matrix of equation 3.11 again. We will ignore the z directions, since they are zero in our frame. Now, we need to find a function for $\phi(t)$ in this rotation matrix such that the two dimensional positional vector is given by $\begin{bmatrix} R \\ 0 \end{bmatrix}$ in the rotating frame.We have to find an $\omega = \frac{d}{dt}\phi(t)$ such that the following conditions hold

$$\begin{cases} R\begin{bmatrix}\cos(\phi) & -\sin(\phi)\\ \sin(\phi) & \cos(\phi)\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = R\begin{bmatrix}\cos(\nu)\\ \sin(\nu)\end{bmatrix} \\ \bar{v}\begin{bmatrix}-\sin(\nu)\\ \cos(\nu)+\epsilon\end{bmatrix} = -\omega\begin{bmatrix}\sin(\phi) & \cos(\phi)\\ -\cos(\phi) & \sin(\phi)\end{bmatrix}\begin{bmatrix}R\\0\end{bmatrix} + \begin{bmatrix}\cos(\phi) & -\sin(\phi)\\ \sin(\phi) & \cos(\phi)\end{bmatrix}\begin{bmatrix}\dot{R}\\0\end{bmatrix} \\ R = a\frac{1-\epsilon^2}{1+\epsilon\cos(\nu)}\end{cases} \tag{4.2}$$

From this it follows that $\phi = \nu$ and $\dot{R} = a\epsilon\sin(\nu)\omega\frac{1-\epsilon^2}{(1+\epsilon\cos(\nu))^2} = \frac{\epsilon R\omega\sin(\nu)}{1+\epsilon\cos(\nu)}$. Combining this results with system of equations 4.2, implies that we have to find an ω such that the following set of equations hols:

$$\begin{cases} -\bar{v}\sin(\nu) = -\omega R\sin(\nu) + \frac{\epsilon R\omega\cos(\nu)\sin(\nu)}{1+\epsilon\cos(\nu)} \\ \bar{v}(\cos(\nu)+\epsilon) = \omega R\cos(\nu) + \frac{\epsilon R\omega\sin^2(\nu)}{1+\epsilon\cos(\nu)}\end{cases}$$

We now multiply the top equation with $\frac{\cos(\nu)}{\sin(\nu)}$ and add it to the bottom. If this yields a unique solution, it directly implies that the original system of two equations for one variable was mathematically consistent. We obtain

$$\bar{v}\epsilon = \frac{\epsilon R\omega(\sin^2(\nu)+\cos^2(\nu))}{1+\epsilon\cos(\nu)} = \frac{\epsilon R\omega}{1+\epsilon\cos(\nu)}$$

which finally evaluates to

$$\omega = \sqrt{\frac{Gm_S(1+\epsilon\cos(\nu))^2}{aR^2(1-\epsilon^2)}} = \sqrt{\frac{Gm_S(1+\epsilon\cos(\nu))}{R^3}}$$

The angular velocity is no longer constant and does no longer represent the mean motion of Jupiter. The only criterion, about the angular velocity of the rotating frame and Jupiter's distance function, in order to keep the found equations of motion valid is that Jupiter is on the horizontal axis which is now the case. We now have to pick an arbitrary initial value for the true anomaly, which we choose to have value 0 at time $dt/2$, where dt is the time step used in the numerical method. Note that since it holds that $\omega = \frac{d\nu}{dt}$, we can estimate the new true anomaly at a time step by adding ωdt to the old one. The full scheme to implement the eccentricity in a numerical method is therefore given by:

1. Start with calculating $\omega_J = \sqrt{\frac{Gm_S}{R^3}}$ at time zero.
2. Estimate the mean anomaly at $t = \frac{1}{2}dt$ by $m_J \approx \frac{1}{2}\omega_J dt$.
3. Calculate Jupiter's distance R using formula 4.1 at the same time as used there.
4. Calculate Jupiter's true anomaly using equation 3.3.
5. Calculate the new angular frequency using $\omega_J = \sqrt{\frac{Gm_S}{R^3}}$.
6. Update R to correct for the eccentricity by using equation 3.1 with Jupiter's eccentricity and true anomaly and the calculated R form step 4 plugged in for a.
7. Perform one iteration step over the equations of motion.
8. Update the mean anomaly by adding the approximation $\omega_J dt$ to it.
9. Repeat steps 3-9 until the desired simulation time is reached.

5 Results

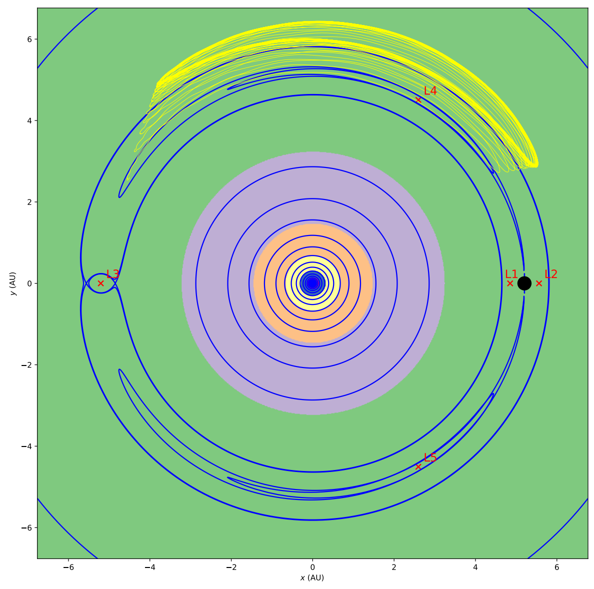
After the methods described in the method chapter are executed, we were able to generate some graphs and tables with the results. In this chapter, we will provide results that are directly related to our original method and research question. Some more results that were created to investigate our model and several different effects are given in the discussion chapter in the section where they are discussed. Apart from these results, there are some additional results which are not discussed. Some of these where only created for the first few Trojans or contained short simulation times to obtain some insight. Others where accidentally created (not wrong, but not with the investigated values). However, in all cases these are valid results nonetheless and therefore they are given in appendix D.

5.1. Test Trojan

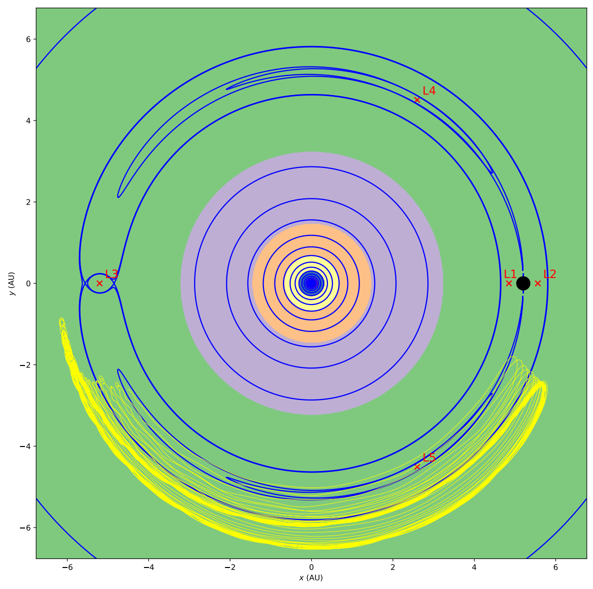
Before we proceeded to simulate many results, we firstly reproduced the results using a test particle with a high resonant initial angle to check whether our results where consistent with Li et al[2]. Because they did not provide the full initial conditions and we did not consider eccentricities, it is not possible to recreate the exact same graphs, but we can compare whether the results have the same properties. We choose our test particles with zero initial velocity in the rotating frame with a high initial angle. The used initial conditions are:

$$\begin{aligned} r &= R \\ \theta &= \pm 150^\circ \\ z &= 0 \\ p &= 0 \\ l_z &= \omega_J R^2 \\ p_z &= 0 \end{aligned} \tag{5.1}$$

We simulated these particles over 10000 years with a time step of 1/3 years. We firstly plot the orbits in the horizontal plane. This led to different plots for the L4 and L5 point shown in figures 5.1. In these figures one observes that the yellow orbit is in both cases more dense further away from the centre. The orbit actually consists of many lines printed over each other, thus indicating that the Trojan is in this region for a long time. Since the migration is much shorter than the total simulation time, this thus means that the Trojan has moved outwards during the migration, which is as expected. However, what differs between the plots, is that the L4 Trojan appears to have a smaller range in angles, while the L5 orbit widens. For some more details on this effect and to be able to compare it with figure 3.5, we now plot the angles over time in figure 5.2. The plot shows that the maximum angle for the L4 Trojan rapidly decreases and then remains constant after the migration. However, for the L5 Trojans, the resonant angle is increased and this effect develops long after the migration. This indicates that it could take many years before the L5 Trojan is kicked out of orbit, which is consistent with the findings of Li et al.[2]. On the other hand, the fast stabilisation of the maximum angle for the L4 Trojan, suggest a long term stability of the L4 Trojan. This hypothesis is supported by continuing the simulations to a total duration of a hundred thousand years. The resulting orbits are plotted in figure 5.3. The test particle gives some

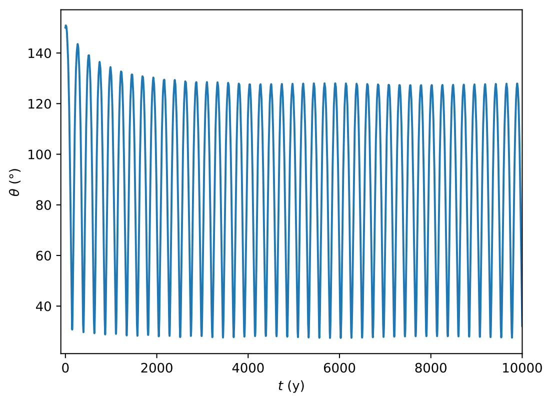


(a) L4 Trojan

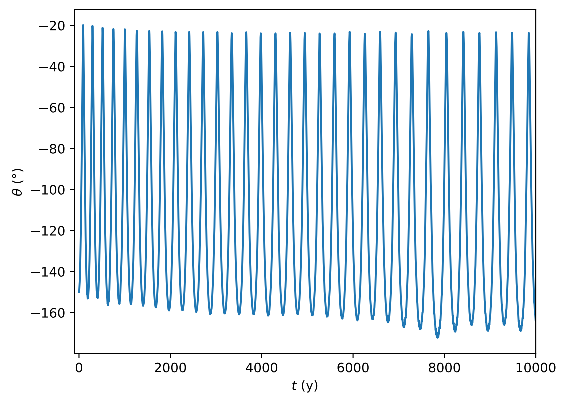


(b) L5 Trojan

Figure 5.1: Orbit of a test Trojan initially placed near the Lagrange point with angles with respect to Jupiter of $\pm 150°$. Jupiter is then migrated outward over 1 AU. From the figure one deduces that the Trojans move outward as well but the orbit of the L4 Trojan narrows, while the orbit of the L5 Trojan broadens. The total simulated time is 10000 years. The position of the Lagrange points and Jupiter, which is represented by the black dot, are determined at the beginning of the migration.



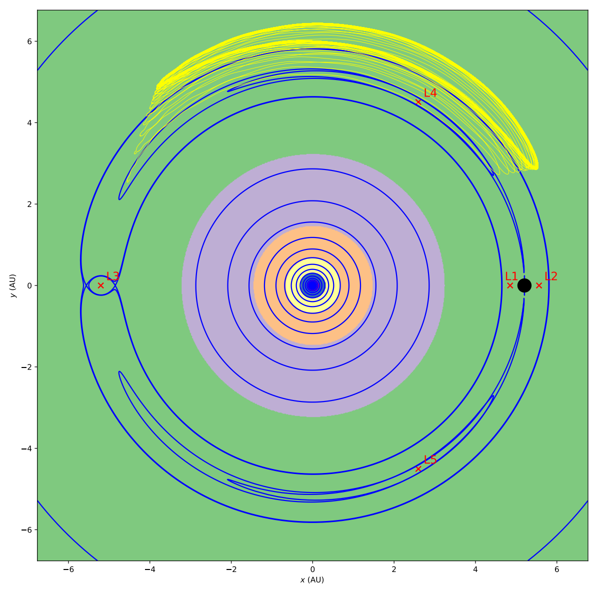
(a) L4 Trojan



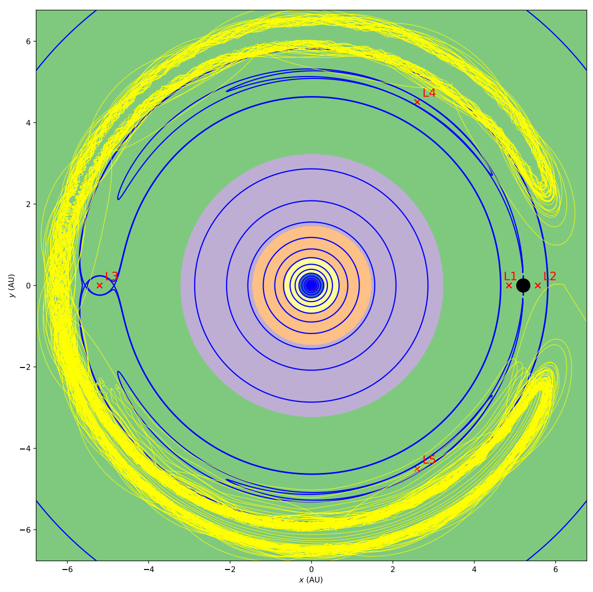
(b) L5 Trojan

Figure 5.2: Angles of a test Trojan initially placed near the Lagrange point with angles with respect to Jupiter of $\pm 150°$ during the migration of Jupiter. Jupiter is then migrated outward over 1 AU. The resonant angle for the L4 Trojan decreases rapidly, while for the L5 Trojan it increases for a much longer time and more irregularly.

first insight in the behaviour of the model and the migration duration, but both are elaborated further in the discussion.



(a) L4 Trojan



(b) L5 Trojan

Figure 5.3: Orbit of a test Trojan initially placed near the Lagrange point with angles with respect to Jupiter of $\pm 150°$. Jupiter is then migrated outward over 1 AU. From the figure one deduces that the Trojans move outward as well but the orbit of the L4 Trojan narrows and remain stables while the L5 Trojan is eventually kicked out, denoted by the line on the right leaving the figure. The total simulated time is 100000 years.

5.2. Result tables

Using the method described in the methods section 4, tables 5.1, 5.4 and 5.3 were created, which contain the original Trojan types and the types after a migration of 1 AU using the values 500, 1000 and 1500 for τ. For the case with $\tau = 1000$, we also simulated a migration of 2 AU, for which the results are shown in table5.2. The used values for these variables are a small selection of the values investigated in the research of Li et al[2]. As explained in the method section, we did nonetheless use several of their investigated values to get more reliable results. The tables belonging to the simulations that did take Jupiter's eccentricity into account are given in subsection 5.2.1.

	L4'	L5'	Horse shoe'	Unstable'	Total
L4	11915	0	119	154	12188
L5	0	12141	30	16	12187
Horse shoe	11	25	37	33	106
Unstable	33	67	73	456	629
Total'	11959	12233	259	659	25110

Table 5.1: Trojans of each type after a migration of 1 AU of Jupiter from its current position over a time of one million years using a time step of 1/3 years. The migration rate is described equation 4.1 with $\tau = 1000$ y. The initial Trojans are 12555 actual Trojans from [14] with its symmetric copies to obtain as many L4 and L5 Trojans initially. The prime denotes the types after the migration.

	L4'	L5'	Horse shoe'	Unstable'	Total
L4	9448	1	1359	1380	12188
L5	0	12119	28	40	12187
Horse shoe	5	24	40	37	106
Unstable	22	96	93	418	629
Total'	9475	12240	1520	1875	25110

Table 5.2: Trojans of each type after a migration of 2 AU of Jupiter from its current position over a time of one million years using a time step of 1/3 years. The migration rate is described equation 4.1 with $\tau = 1000$ y. The initial Trojans are 12555 actual Trojans from [14] with its symmetric copies to obtain as many L4 and L5 Trojans initially. The prime denotes the types after the migration.

	L4'	L5'	Horse shoe'	Unstable'	Total
L4	12101	0	41	46	12188
L5	0	12160	18	9	12187
Horse shoe	13	21	44	28	106
Unstable	35	57	74	463	629
Total'	12149	12238	177	536	25110

Table 5.3: Trojans of each type after a migration of 1 AU of Jupiter from its current position over a time of one million years using a time step of 1/3 years. The migration rate is described equation 6.2 with $\tau = 1500$ y. The initial Trojans are 12555 actual Trojans from [14] with its symmetric copies to obtain as many L4 and L5 Trojans initially. The prime denotes the types after the migration.

	L4'	L5'	Horse shoe'	Unstable'	Total
L4	10045	0	10	2133	12188
L5	0	12097	0	90	12187
Horse shoe	8	21	1	76	106
Unstable	18	65	0	546	629
Total'	10071	12183	11	2845	25110

Table 5.4: Trojans of each type after a migration of 1 AU of Jupiter from its current position over a time of one million years using a time step of 1/3 years. The migration rate is described equation 4.1 with $\tau = 500$ y. The initial Trojans are 12555 actual Trojans from [14] with its symmetric copies to obtain as many L4 and L5 Trojans initially. The prime denotes the types after the migration.

5.2.1. Eccentricity

In the initial model, the eccentricity of Jupiter's orbit was neglected. In order to be able to test the model, we ran the simulations of the migration over 1 AU again for $\tau = 1000$ and $\tau = 500$, but now with taking Jupiter's orbital eccentricity into account during the entire simulation. The results will be compared with the principle results form section 5.2 in the discussion section, but here we will show the results for each Trojan type in tables 5.5 and 5.6.

	L4'	L5'	Horse shoe'	Unstable'	Total
L4	11692	0	1	495	12188
L5	0	12082	0	105	12187
Horse shoe	5	16	0	85	106
Unstable	8	47	0	574	629
Total'	11705	12145	1	1259	25110

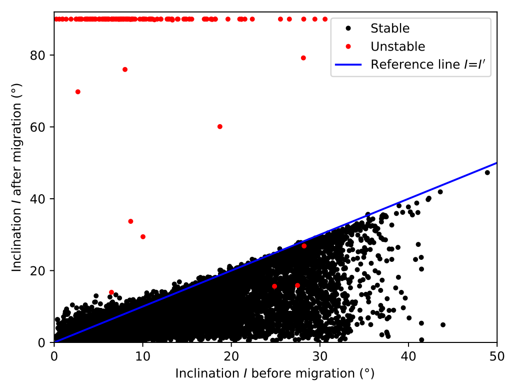
Table 5.5: Trojans of each type after a migration of 1 AU of Jupiter from its current position over a time of one million years using a time step of 1/3 years. The migration rate is described equation 4.1 with $\tau = 1000$ y. The initial Trojans are 12555 actual Trojans from [14] with its symmetric copies to obtain as many L4 and L5 Trojans initially. The prime denotes the types after the migration. During the entire simulation, Jupiter had its current eccentricity of 0.04839266.

	L4'	L5'	Horse shoe'	Unstable'	Total
L4	9904	0	2	2282	12188
L5	0	12057	0	130	12187
Horse shoe	7	20	0	79	102
Unstable	13	67	0	549	629
Total'	9924	12144	2	3040	25110

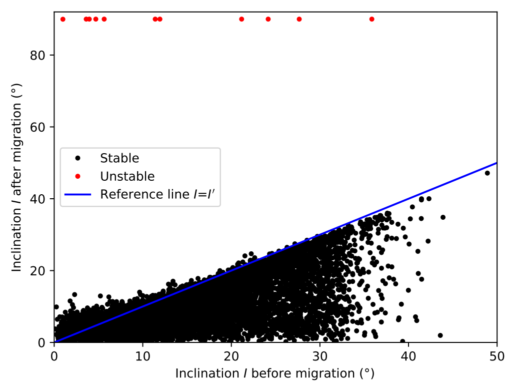
Table 5.6: Trojans of each type after a migration of 1 AU of Jupiter from its current position over a time of one million years using a time step of 1/3 years. The migration rate is described equation 4.1 with $\tau = 500$ y. The initial Trojans are 12555 actual Trojans from [14] with its symmetric copies to obtain as many L4 and L5 Trojans initially. The prime denotes the types after the migration. During the entire simulation, Jupiter had its current eccentricity of 0.04839266.

5.2.2. Inclinations

From the simulations, graphs were created using the inclinations before and after the simulation for the original L4 and L5 Trojans with the migration of 1 AU using $\tau = 1000$. For this, we considered the case without Jupiter's eccentricity in figure 5.4 and the case with Jupiter's eccentricity in figure 5.6. Also, a close up of the stable Trojans with low initial inclinations is shown for both graphs in figure 5.5 and 5.7 respectively.

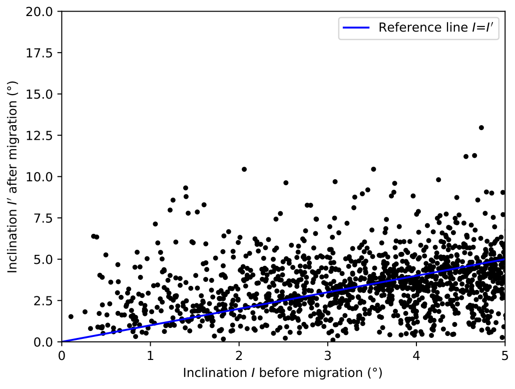


(a) L4

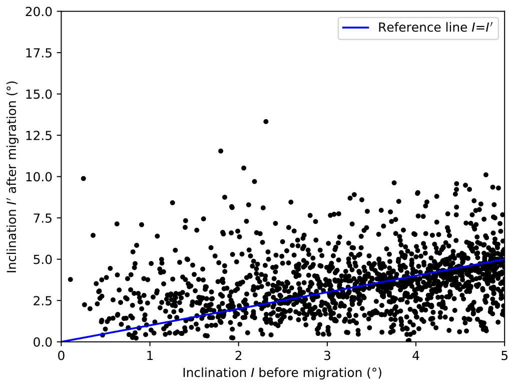


(b) L5

Figure 5.4: Inclinations before and after a migration of Jupiter of the Trojans from table 5.1. The type of a Trojan is determined before the migration and the stability is determined after the migration.

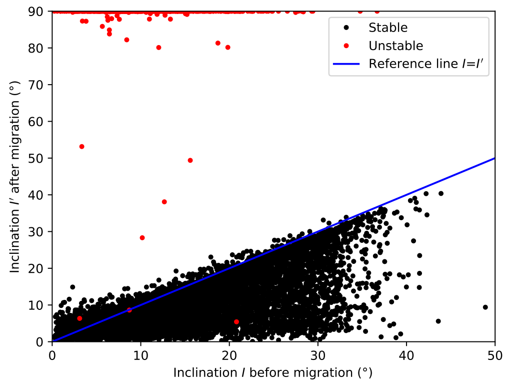


(a) L4

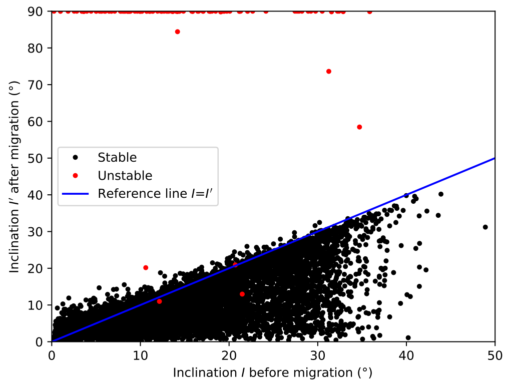


(b) L5

Figure 5.5: Zoomed in on Trojans that had low inclinations before the migration in figure 5.4. Only the Trojans that were stable after the migration are considered

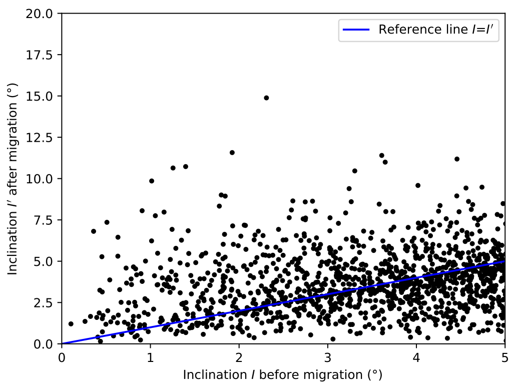


(a) L4

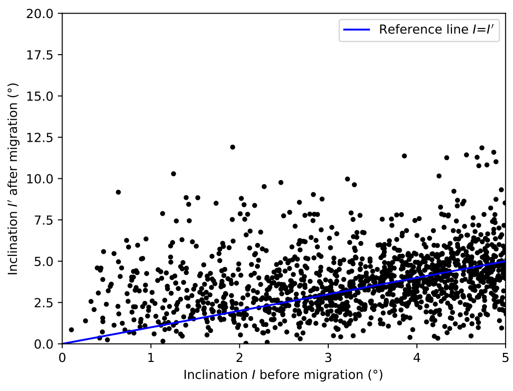


(b) L5

Figure 5.6: Inclinations before and after a migration of Jupiter of the Trojans from table 5.5. The type of a Trojan is determined before the migration and the stability is determined after the migration.



(a) L4



(b) L5

Figure 5.7: Zoomed in on Trojans that had low inclinations before the migration in figure 5.6. Only the Trojans that were stable after the migration are considered

6 Discussion

Before we are able to draw any conclusions about the results found, it is important to consider potential flaws in the method. The first part of the model that will be tested is the numerical method. Although it is difficult and time consuming to properly test it for all results, we will verify whether it behaves as expected for Trojans near the equilibrium. Then we will test the model itself by investigating the effect on different migration implementations and the Jupiter's orbital eccentricity. Also the migration duration will be varied as well as the effect of several initial conditions by comparing the data with generated Trojan data. Finally, we will investigate whether the migration could explain some of the properties of the current inclinations as described in the introduction. Although we will do some suggestions for further research within these sections, we will end with further research recommendations on parts that we did not investigate in this discussion.

6.1. Integrator comparison

A good integrator for our Hamiltonian problem should at least obey two properties: the order of the method should equal the theoretical order near the mathematical equilibrium and for the non-migrating non-eccentric case the Hamiltonian should be more or less conserved, since it is physically conserved for this system by the conservation of energy. Typically, a numerical method will not conserve energies at all times, but if the deviations are relatively small and approach zero on average, the method is appropriate.

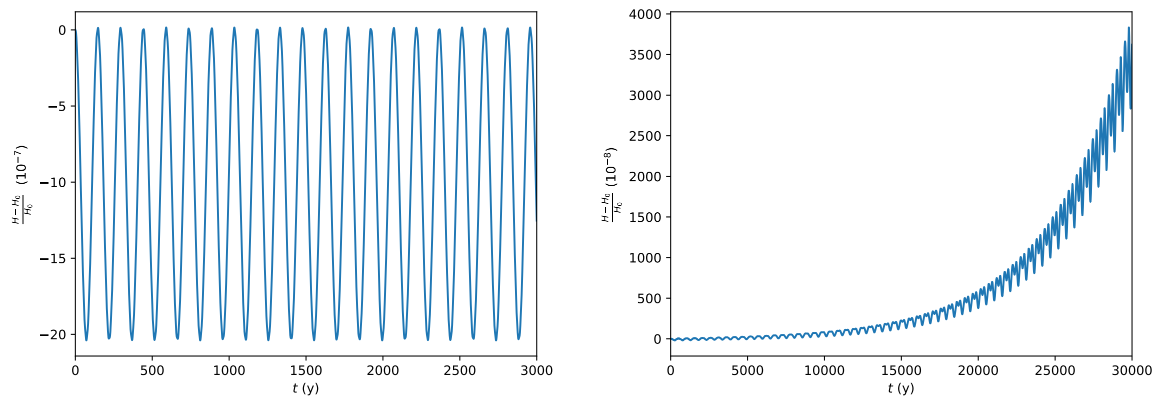
The two criteria are tested by comparing the two different Yoshida integrators, namely the original one described in appendix A.1 and the midpoint Yoshida integrator we used, for the orbit of a Trojan very close to the Lagrange point. We have tested the simplest case where there is no migration and Jupiter has zero orbital eccentricity. This choice is made because if we find a method being inappropriate for this simple near-equilibrium case, it is unlikely that it will be appropriate for the more complex case.

To test whether the methods obeys the first criterion of the order, we used a Richardson estimation. If a numerical solution of a one dimensional problem is given by a w_{dt}^{N}, where N is the number of iterations at the moment the solution is determined and dt is the used time step, the Richardson estimation is given by[26]

$$2^p \approx \frac{w_{N/2}^{2dt} - w_{N/4}^{4dt}}{w_N^{dt} - w_{N/2}^{2dt}} \tag{6.1}$$

where p is the order. Now since we have six equations of motions, our solution thus is a vector containing the approximations for the six variables after iteration N. Equation 6.1 is still valid, but now one must choose a variable out of the vector to calculate the values. We estimated the order with θ as variable, since this variable was certain to vary a lot during the simulation (if we consider variables that change too little, the division of the near zero numbers may actually lead to a high numerical error itself which results in a false error estimation). using $dt = 1/10$ -which is much less than the limit for the MYI derived in appendix A.2- and $N = 30000$. it led to an estimated order of 1 for the original Yoshida integrator, while the midpoint Yohsida integrator had the expected order of 4. We conclude that the non separability of the Hamiltonian indeed does not preserve the desired first order of the OYI. While it might get the order of the OYI as expected for smaller time steps, it is not of our interest to investigate this effect as a smaller time step would make the simulations to slow, making the method still inappropriate.

To test whether both methods do satisfy the second criterion that it conserves the Hamiltonian, we plotted the Hamiltonian compared with the initial energy over a time of 30000 years using a time step of 1/10 years. Since the behaviour of the MYI did not change over this time, its graph appeared to be a block, thus we zoomed in on the first 3000 years to obtain some insight in the behaviour. The results are shown in figure 6.1 The behaviour of the MYI is as expected. The Hamiltonian does change, but the



(a) Original Yoshida Integrator **(b)** Midpoint Yoshida Integrator

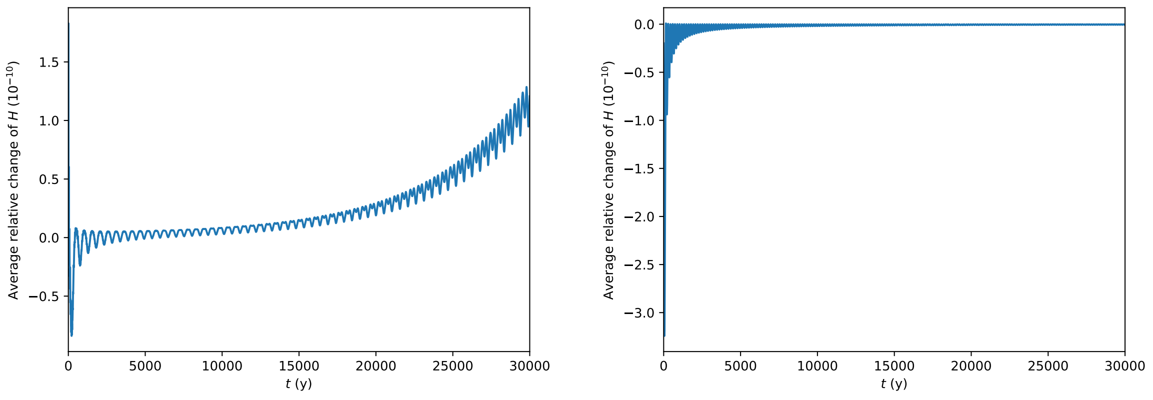
Figure 6.1: Hamiltonian of a Trojan near a Lagrange point using the original and an improved midpoint Yoshida integor over 30000 years with a time step of 1/10 years. The behaviour of the midpoint Yoshida integrator did not change over time, making the graph appear like a block due to the consistent rapid oscillations. Therefore, this graph is zoomed in to the first 3000 years to give insight in the oscillation. For the original Yoshida integrator, the effect is not constant but changes over time. Therefore this graph is not zoomed in.

amplitude of this change is small (more than 9 orders of magnitude smaller than the original energy) and the change is oscillating around a constant value, giving an on average conserved energy. Therefore, this is indeed a symplectic integrator for our problem.

To visualize the behaviour of both integrators on the same time scale of 30000 years, we have plotted the average relative change of the Hamiltonian at each time step in figure 6.2. For a stable method, we expect this average change to converge to zero, which seems to hold for the midpoint Yoshida integrator. However, for the OYI, we see that it is an increasing function, meaning that the Hamiltonian is changing faster and faster, leading to a highly unstable method for our problem. When the results are extended on much longer time scales till one million years, the found characteristics did not shown any visible change for the MYI. Also decreasing the time step to 1/3 years did not create any visible difference. Since the method appears to allow small time steps, is stable for stable Trojans and conserves the symplectic property on the desired simulation duration, we conclude that the MYI is an appropriate method to simulate near equilibrium Trojans. However, for the OYI the errors kept increasing on the longer time scale, resulting in massive energy changes and unstable Trojans. We therefore conclude that this integrator is not a valid choice for our problem. A side note on our method is that we have not proven its behaviour for the migrating case. A possible test for this behaviour is to use Richardson estimations to check whether the errors are small enough. The disadvantage is that it costs a lot of computation time to estimate the errors of many results and therefore we chose to leave it for further research. Nonetheless, the code for such an error estimation is provided in the library from appendix E.2.

6.1.1. Code implementation

Apart from the method itself, it is important how the method is implemented in a code. In order to make our code very useful for further research, we developed it with ideas in mind: firstly, it should be



(a) Original Yoshida Integrator **(b)** Midpoint Yoshida Integrator

Figure 6.2: Hamiltonian of a Trojan near a Lagrange point using a regular Yoshida integrator with 100000 time steps of 1/30 years. The Hamiltonian is increasing, which is seen in the normal plot on the left as well as in the right relative plot where the average appears to converge to a nonzero horizontal line, implying a constant Hamiltonian increment, indicating unstable long term behaviour.

sufficiently fast in order to run many long simulations on a regular computer. Secondly, its basics should be easily readable. As described in section 4, this is done by writing a very optimised C code for the simulations themselves, but handling the data in Python using a library with some commented standard functions. The most noticeable optimisations used are:

1. Multi threading to run multiple simulations simultaneously.
2. Finding as base therm in the potentials and calculating them once.
3. Writing out powers like $x * x$ instead of using the build in power function.
4. Defining an additional; function for the power of $3/2$ by x*sqrt*(x) since the square root function is much faster than the power function in C.
5. Merging constants and defining them as one global variable, such as the variable Gm_S for the product $G \cdot m_S$.
6. Calculating extensive variables, for example because they contain a trigonometric function, that are used multiple times in a step once and given them as function parameters, such as $r\cos(\theta)$.
7. Giving the compiler several instructions about the type of functions, such that it could be optimised.

For the last step, one should also use the flag '-O3' when compiling the code. This is the case for the automatic compilation in the example code in appendix E.5. Although the final performances are dependent on among other things the machine used, we will provide estimates of our observed performances to give the reader an idea of the order of magnitude of the expected run times. For our system, the optimisations finally resulted in a code that was more than 1000 times as fast as the original Python attempt. With the current code, we could simulate 3 million time steps in about 1.4-1.6 seconds on our system for the non-migrating case and about 1.5-1.8 seconds for the migrating case when ignoring the eccentricity. Using our time step of 1/3, this corresponds to the desired 1 million simulated years. When run in parallel, it took about 6 hours to simulate the 25110 migrating Trojans for a million years. The optimisations together with the Python library and comments make our code greatly appropriate for further research. The performance could optionally be improved by using a faster computer or several computers simultaneously, to simulate even more Trojans or longer time spans.

6.2. Migration implementation

Every physical model is a simplification of the real world and therefore introduces errors. It is therefore important to estimate how accurate the used model is. In the next section it will be discussed whether the model satisfies our expectations, but we firstly tested how different implementations of the model effected the results on migrations. We did not bother to test our model for the non-migrating case, since the orbits of Trojans in this case are well known and our plots in for example figure 2.5 already satisfy our expectations. The two investigated effects on the migrations are a different description of Jupiter's distance R_J and implementing the Jupiter's orbital eccentricity in the model.

6.2.1. Distance of Jupiter

The Distance function 4.1 used to estimate the migration in this project was used since it consisted with the choice of Li et al. and was to our knowledge the only function substantiated (by Malhorta[9]). However, the simplest model is to use a constant migration rate. We have already met the function describing this migration in figure 4.2. To test this function, we choose a comparable migration speed. We firstly tested this model on a time scale of 100000 years. The results are shown in table 6.1 It follows

L4	L5	Horse shoe	Unstable
14823	1	505	9466
3	14846	533	9402
9	3	4	32
4	17	0	376

Table 6.1: Trojans of each type after a migration of 1 AU of Jupiter from its current position over a time of one million years using a time step of 1/3 years. The migration rate is $1.5 \cdot 10^{-4}$ AU y^{-1} over a distance of 0.5 AU. The initial Trojans are 12555 actual Trojans from [14] with its symmetric copies in both the horizontal plane and their copies in the other Lagrange point to obtain as many L4 and L5 Trojans initially. The prime denotes the types after the migration.

that there are much more unstable Trojans after the migration when compared to table 5.1. As we already derived in the method section, the total migration duration is comparable for both simulations. Since the total migration distance simulated for the constant migration rate is shorter than the 1 AU used for the results from table 5.1, we would have expected more stable Trojans instead if the two functions had comparable properties. From the plots in figure 4.2 follows that the key differences are that the migration of the exponential function behave like a steeper horizontal function which smoothens of at the end. Apparently, the abrupt stop forms a shock effect resulting in much more unstable Trojans. Since the smoothing appears to have a huge effect, this makes one wondering what would happen if the start is smooth as well. Therefore, we also tried to generate some simulations using the function

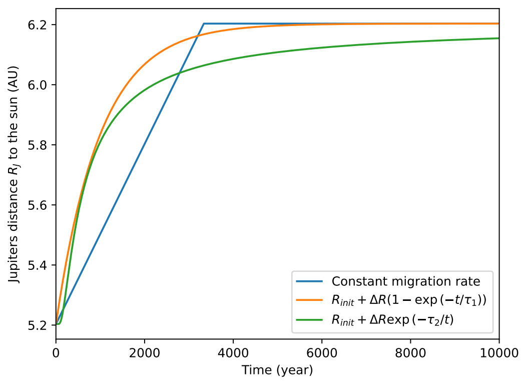
$$R = R_{\text{initial}} + \exp\left(\frac{-\tau}{t}\right) \tag{6.2}$$

The function is plotted in figure 6.3 using two different values for the parameter τ. As seen in the figure, the function starts as a near horizontal line and then rapidly rises to a steep function as the other exponential function and it also stops smoothly. However, we have to make a choice between a similar steepness to the other exponential figure or a similar total migration time by choosing different values for τ. This makes it hard to compare two individual results of the two functions together, but we can compare the overall results. Using a slow migration of $\tau = 1000$ years, we obtained the following results in table 6.2. For a migration with $\tau = 500$, the result is shown in table 6.3 for the case Jupiter's orbit is circular and in table 6.4 for the case when Jupiter has a nonzero orbital eccentricity. We conclude that without considering Jupiter's orbital eccentricity, it is possible to obtain more L4 than L5 Trojans using this function, bit the margins are small when the migration is fast. From the three different functions

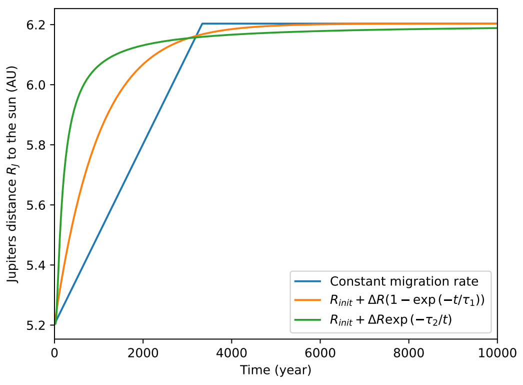
	L4'	L5'	Horse shoe'	Unstable'	Total
L4	12103	0	0	85	12188
L5	0	12082	0	105	12187
Horse shoe	11	8	0	87	106
Unstable	9	12	0	608	629
Total'	12123	12102	0	885	25110

(a) $\tau = 1000$ y

Table 6.2: Trojans of each type after a migration of 1 AU of Jupiter from its current position over a time of one million years using a time step of 1/3 years. The migration rate is described equation 6.2 with $\tau = 1000$ y. The initial Trojans are 12555 actual Trojans from [14] with its symmetric copies to obtain as many L4 and L5 Trojans initially. The prime denotes the types after the migration.



(a) Similar steepness ($\tau_2 = 500$ y)



(b) Similar migration duration ($\tau_2 = 150$ y)

Figure 6.3: Graphs for several descriptions of Jupiter's radial distance to the Sun during an outward migration of 1 AU from its current position. Here the total migration time of the constant migration is 3333 years and $\tau_1 = 1000$ years to obtain a similar total migration time. For the smoother exponential function, we can either pick $\tau_2 = 500$ years such that its steepness is comparable with the other exponential function or $\tau_2 = 150$ years such that its total migration time is comparable.

	L4'	L5'	Horse shoe'	Unstable'	Total
L4	12069	0	2	117	12188
L5	0	12059	1	127	12187
Horse shoe	13	8	1	84	106
Unstable	16	13	0	600	629
Total'	12098	12080	4	928	25110

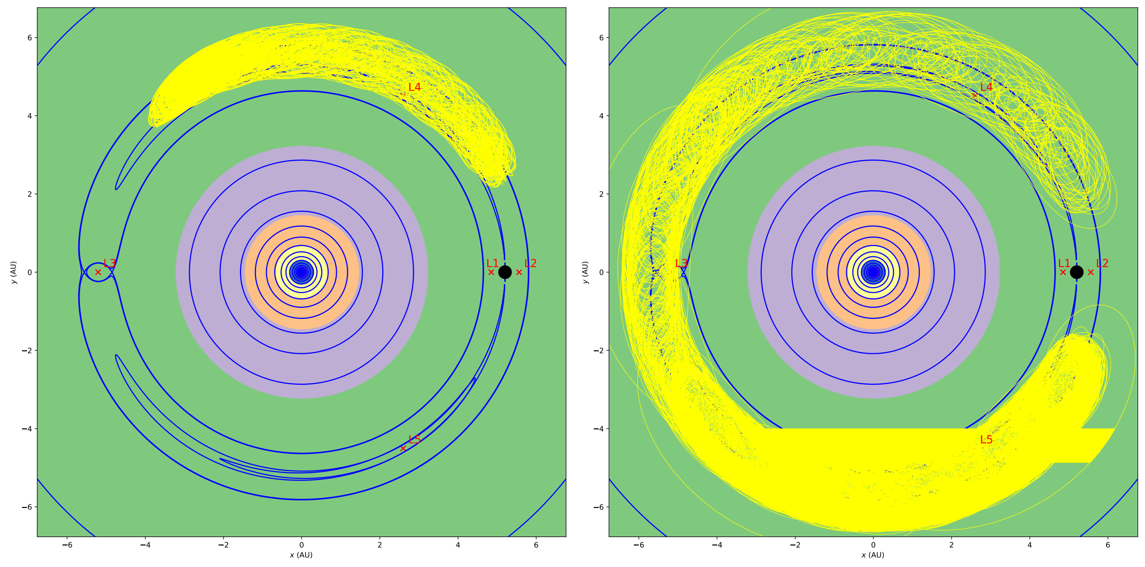
Table 6.3: Trojans of each type after a migration of 1 AU of Jupiter from its current position over a time of one million years using a time step of 1/3 years. The migration rate is described equation 6.2 with $\tau = 500$ y. The initial Trojans are 12555 actual Trojans from [14] with its symmetric copies to obtain as many L4 and L5 Trojans initially. The prime denotes the types after the migration.

	L4'	L5'	Horse shoe'	Unstable'	Total
L4	12022	0	0	166	12188
L5	0	12048	0	139	12187
Horse shoe	13	9	0	84	106
Unstable	20	13	0	596	629
Total'	12055	12070	0	1085	25110

Table 6.4: Trojans of each type after a migration of 1 AU of Jupiter from its current position over a time of one million years using a time step of 1/3 years. The migration rate is described equation 6.2 with $\tau = 500$ y and Jupiter's orbital eccentricity is taken into account. The initial Trojans are 12555 actual Trojans from [14] with its symmetric copies to obtain as many L4 and L5 Trojans initially. The prime denotes the types after the migration.

considered, it is clear that the way in which the migration is implemented is of major importance for our results. In particular, the three different functions that could be adopted to have comparable migration speeds or duration indicate that not only the migration speed itself is important, but also how fast it reaches and slows down the migration. With the smoother start, it appears that for our initial conditions there could be more L4 than L5 Trojans at the end of the simulations, but the differences are small.

Although one could imagine that the first function with a constant migration rate is not physical as we expect a differentiable function, there are still many differentiable functions to choose from. For example, we could choose a function of the form exp $-\tau/t^2$ to obtain an even smoother start of migration, but arrive at a faster migration stop. For the best estimate of the distance function, it is best to use data from simulations with the Nice model, as described in the further research section 6.6.



(a) L4 Trojan **(b)** L5 Trojan

Figure 6.4: Orbit of a test Trojan initially placed at the Lagrange point with angles with respect to Jupiter of $\pm 130°$. Jupiter is migrated outward over 0.5 AU and its current eccentricity is used during the entire migration. From the figure one deduces that the Trojans move outward as well but the orbit of the L4 Trojan narrows and remain stables while the L5 Trojan is eventually kicked out after a long time in a Horse shoe orbit. The total simulation run is 70000 years, but for the L4 only the first 10000 years are plotted. This choice is made because otherwise the individual lines would not be distinguishable from the yellow blob. Nonetheless, the orbit has the same shape over the longer time span.

6.2.2. Jupiter's orbital eccentricity

To simplify the model, Jupiter's eccentricity is neglected. However, Li et al[2] already showed that the Jupiter's orbital eccentricity has an effect on the L4:L5 ratio, although they did not consider the case without eccentricity. We have already met some results when considering the eccentricity in 5.2.1 and when considering the alternative exponential migration function. We conclude that more Trojans have become unstable due to the migration for both Lagrange points. The interesting part is that when there are more L5 Trojans than L4 Trojans after a migration such as in 5.2, the ratio L5:L4 increases when adding the eccentricity to the same simulations. Also, when we look at the test Trojan from section 5.1, it is unstable for both the L4 and the L5 point, but when we decrease the angle to 130 degrees, we see that once again it happens that the L4 Trojan is stable, while the L5 Trojan is unstable as seen in figure6.4. This did not happen without eccentricity for this angle. On the other hand, if we did use the initial angle of 130 degree without eccentricity, both Trojans are stable. Furthermore, the orbits of both Trojans have obtained larger deviations from Jupiter compared to 5.1. All in all we conclude that the eccentricity changes the stability region for both the L4 and the L5 point and since the ratio's change as well, the change is asymmetric. Nonetheless, it is known that the eccentricity already creates an initial asymmetry[3], so to be able to support this conclusion more strongly, it is best to have a data set with more unstable Trojans after the migration such as Li had. Therefore, we will generate some Trojans with comparable initial conditions to investigate the effect of the eccentricity further. Furthermore, this data will be used to compare what the influence is of the initial conditions we chose. These parts will both be discussed in the next section.

6.3. Generated Trojans

Since our initial attempts did not entirely match our expectations, we tried to test our model to initial conditions comparable to those used by Li et al. If the model works as expected on these initial conditions, this yields information about what the possible initial conditions are to obtain more L4 than L5 Trojans. Li made several attempts with different initial conditions and migrations, but we have tested fewer cases, since it is not our research to reproduce all their results, but only to check whether our model and integrator give likewise results. For this test, we generated a thousand L4 Trojans with random eccentricities in a range of 0 and 0.3. All Trojans had the Lagrange distance as semi-major axis as in Li et al. The initial deviation from Jupiter are chosen randomly as well, by picking the initial true anomaly of each Trojan randomly. Using a minimum value of this true anomaly, the Trojans are forced to at least obtain this high angle and are therefore forced in an orbit with a high resonant angle. We only considered inclinations of 0 and 0.01 $^\circ$ and the other Kepler elements where 0. The symmetry described in 3.5 was used to obtain as many L5 Trojans as L4 Trojans with compatible initial conditions. All simulations where run for a simulated time of a million years with time step of $1/3$ years and the input variables with the corresponding results are given in table 6.5.

From this table, it appears that it is indeed possible to obtain more L4 than L5 Trojans. To obtain a

Inclination ($^\circ$)	Jupiter eccentricity	ΔR (AU)	Initial true anomaly ($^\circ$)	L4	L5	L4:L5
0	0	1	30-140	586	5	1.09
0	0	1	30-140	577	531	1.09
0	0.04839266	1	80-140	650	502	1.29
0.01	0.04839266	0.5	90-140	649	550	1.18
0.01	0.04839266	1	90-140	667	480	1.39

Table 6.5: Number of 1000 L4 and 1000 L5 Trojans after a million years of simulation with a migrating Jupiter. The Trojans initially had randomly eccentricities between 0 and 0.3 and started at the current Lagrange point, as is Jupiter. The initial angle is relative to Jupiter and chosen randomly as well. ΔR is the total traveled migration distance of Jupiter and τ is a quantity that determines the migration speed, where a bigger τ means a bigger migration speed. The L4 Trojans are created randomly using the described conditions, from which the L5 Trojans are mirrors in angle to ensure to have an equal data set in both points. The velocities were mirrored such that in the non-migrating case without eccentricities, the copied Trojans would give the same orbits as the original, but mirrored.

more statistical accurate result we firstly reproduced the last simulation with ten times as many Trojans. This resulted in 7220 stable L4 Trojans and 5285 Stable L5 Trojans resulting in a ratio L4:L5 of 1.37. As the uncertainty in a variable can generally be estimated by $\sqrt{N}$ where N is the number of Trojans in that point, we can estimate the uncertainty in the ratio by $\frac{\sqrt{7220}}{7220} + \frac{\sqrt{5285}}{5285} \approx 0.03$. We conclude that the ratio L4:L5 is given by 1.37 ± 0.03 and thus that our simulations indeed lead to more L4 Trojans than L5 Trojans.

One of the remarkable things from table 6.5, is that Jupiter's orbital eccentricity has a major influence on the final results. To compare the results better, we generated once again 10000 L4 Trojans with its 10000 L5 copies and now ran the simulation with and without eccentricity on this same data set. This resulted in table 6.6. We conclude that the eccentricity has a great influence on these generated Trojans as well. However, a difference with the results for the actual Trojans, is that this eccentricity increases the L4:L5 ratio, while for the actual Trojans from for example table 5.1 it decreases the L4:L5 ratio. In conclusion, Jupiter's orbital eccentricity emphasizes the ratio arisen by the migration. Nonetheless, the bottom result form table 6.6 suggests that even without an eccentricity population differences are possible, but at least for our results the difference is marginal.

Inclination ($^\circ$)	Jupiter eccentricity	ΔR (AU)	Initial true anomaly ($^\circ$)	L4	L5	L4:L5
0.01	0	1	90-140	6902	4965	1.39$\pm$0.03
0.01	0.04839266	1	90-140	8150	7806	1.05$\pm$0.02

Table 6.6: Number of 10000 L4 and 10000 L5 Trojans after a million years of simulation with a migrating Jupiter. The Trojans initially had randomly generated eccentricities between 0 and 0.3 and started at the current Lagrange point. The initial angle is relative to Jupiter and chosen randomly as well. ΔR is the total traveled migration distance of Jupiter and τ is a quantity that determines the migration speed, where a bigger τ means a bigger migration speed. The L4 Trojans are created randomly using the described conditions, from which the L5 Trojans are mirrors in angle to ensure to have an equal data set in both points. The velocities were mirrored such that in the non-migrating case without eccentricities, the copied Trojans would give the same orbits as the original, but mirrored.

The generated Trojan data has given us some insight in the effect of the eccentricity, but it leaves us puzzling what parameters in the generated data and the real Trojan data lead to this differences in the L4:L5 ratio. We have hypothesised several quantities to cause this effect, which are given below using some graphs. All graphs where generated using $\tau = 1000$ y, a total migration of $\Delta R = 1$ AU and simulated during one million years. Jupiter's orbit was approximated circular in the simulation.

The first attempt was to make a plot of the resonance angle $\Delta\theta$ in the non-migrating case versus the maximal change in r which we called the maximal horizontal displacement δr. These quantities are determined by running the simulations for the non-migrating case , without considering the Jupiter's orbital eccentricity, 10000 years using a time step of 1/30 years. To obtain a readable plot, we have taken a close look into the symmetric Trojan pairs; only Trojans which are stable after the simulation while its mirror is not are plotted. The result is shown in figure 6.5. From this figure, it appears that it

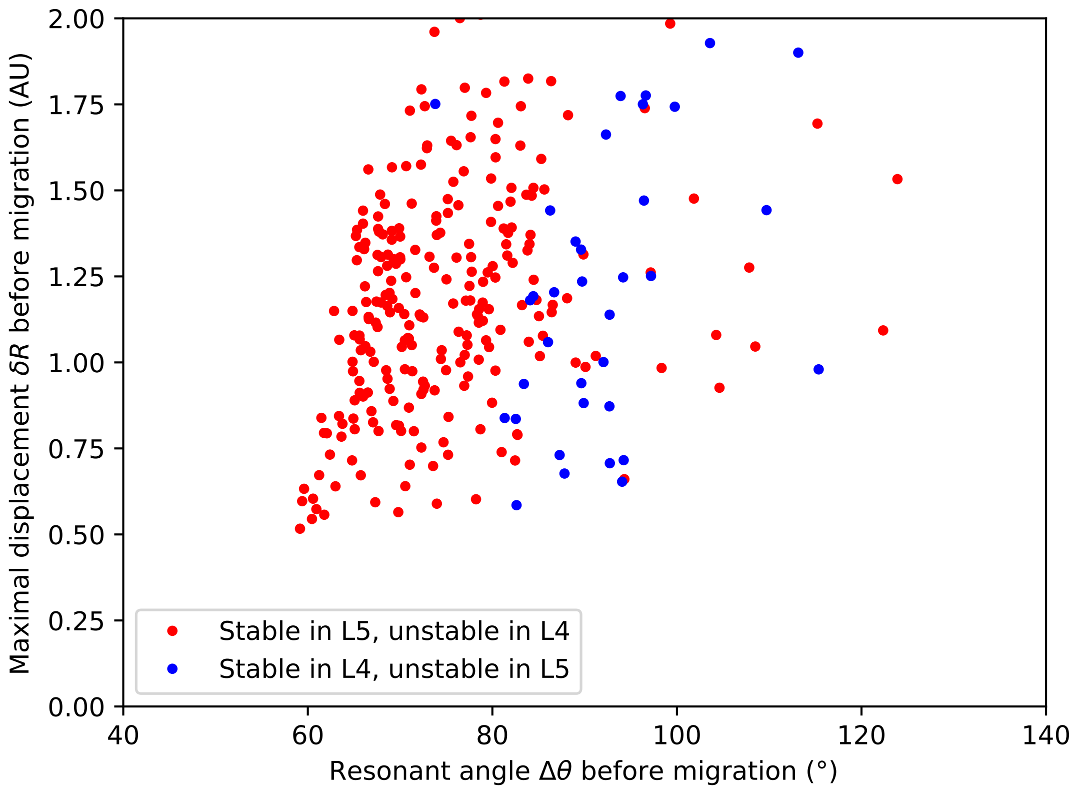
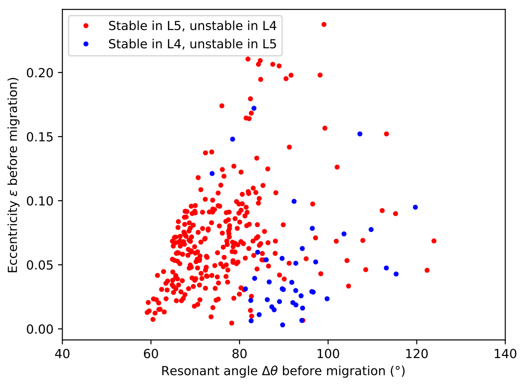


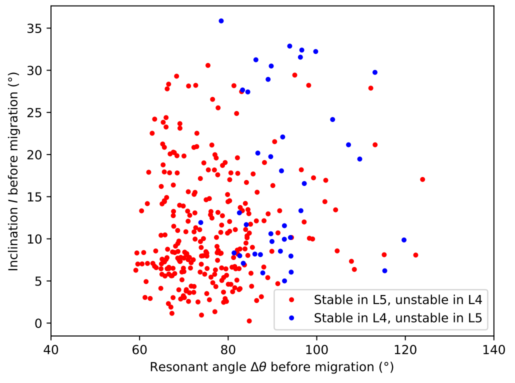
Figure 6.5: Plots of the initial resonant angle $\Delta\theta$ of some Trojans versus the initial maximum horizontal displacement δR. The stability is determined after a migration of Jupiter over 1 AU described by equation 4.1 with parameter $\tau = 1000$ years. The total simulation time is one million years. The Trojans considered in both cases are the same and are actual Trojans from the NASA database[14] and its mirrored counterparts. For this plot. Trojans that led to unstable Trojans for both the original Trojan and a copy or to stable Trojans in both cases, are omitted to emphasize the different between the original Trojans and the mirrors.

mostly happens that an L4 Trojan is stable and an L5 Trojan is unstable, when the resonance angle is very high. On the other hand, if the resonant angle is not very high, it is still possible that L4 Trojans become unstable, it happens much more often that an L4 Trojan becomes unstable, while an L5 Trojan becomes stable. Since their are much more Trojans with low resonance angles under the actual Trojans while 6.5 forces relative high resonant angles, this might explain the differences found between our research and that of Li. If the outward migration of Jupiter indeed has caused the asymmetry, this suggest that initially the Trojans had very high resonant angles which decreased over time due to other reasons. The latter could be caused by collisions of Trojans[10]. The data suggests that in the case with very low resonant angles, there are no unstable Trojans at all after a migration.

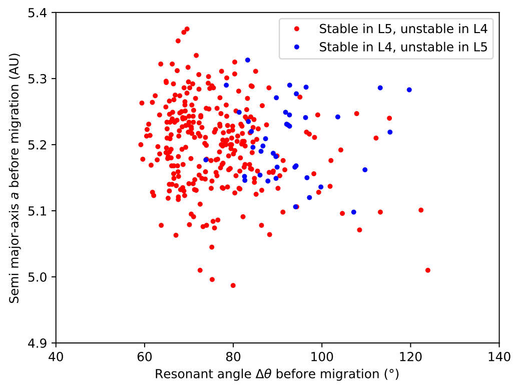
Sometimes it happens, as shown in figure 6.5, that a L5 Trojan may be stable with an unstable L4 copy after a migration with high resonant angles. This suggest that another variable, apart from the resonant angle, is also responsible for the effect that a L4 Trojan is stable after the migration while its L5 mirror is not. Therefore we plotted the resonance angle against four other variables of the initial Trojans: the eccentricity ϵ, the inclination I, the initial semi-major axis a and the Hamiltonian H. The latter is relevant since it represents the total energy for the non-migrating case. The results are shown in figure 6.6. From this figure it appears that the initial inclinations has a small influence on the asymmetric population



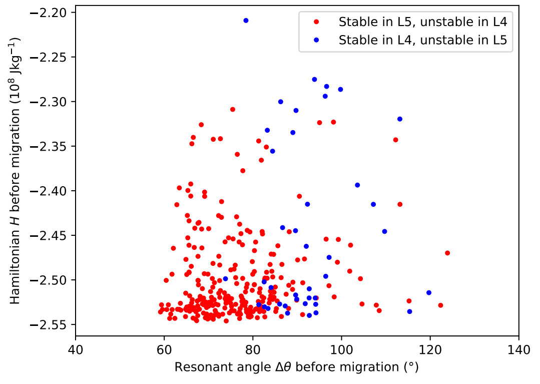
(a) Eccentricity



(b) Inclination



(c) Semi-major axis



(d) Hamiltonian

Figure 6.6: Plots of the initial resonant angle $\Delta\theta$ of some Trojans versus several different other initial quantities. The stability is determined after a migration of Jupiter over 1 AU described by equation 4.1 with parameter $\tau = 1000$ years. The total simulation time is one million years. The Trojans considered in both cases are the same and are actual Trojans from the NASA database[14] and its mirrored counterparts. For this plot. Trojans that led to unstable Trojans for both the original Trojan and a copy or to stable Trojans in both cases, are omitted to emphasize the different between the original Trojans and the copies.

numbers and that higher initial energies lead to more stable L4 Trojans after the migration. The last plot we make is the plot where the minimum and maximum angles are plotted in absolute values in figure 6.7. From this figure, we conclude that especially low minimum angles are important to create stable L4 Trojans with unstable L5 Trojans.

From the attempts, we conclude that it is possible that Jupiter's outward migration leads to more stable L4 Trojans than L5 Trojans, but only given a restricted set of initial conditions containing large resonant angles. This leaves the question what this implies for our Solar System. The data suggests that if the outward migration is indeed responsible for the asymmetry, the original Trojans had large resonance angles. Due to other effects, the Trojans then must have obtained lower resonant angles to obtain the distribution of the current Trojans. Since higher resonance angles correspond to higher energies, this could have occurred due to Trojan collisions, as investigated by Marzari et al.[10]. This is also suggested by Li. However, the important difference with their resource is that our results prove that such a Trojan distribution is a necessary conditions for obtaining the current asymmetry. Therefore, if the Nice model is indeed correct, our results yields insight in the Trojan distribution of the early Solar System.

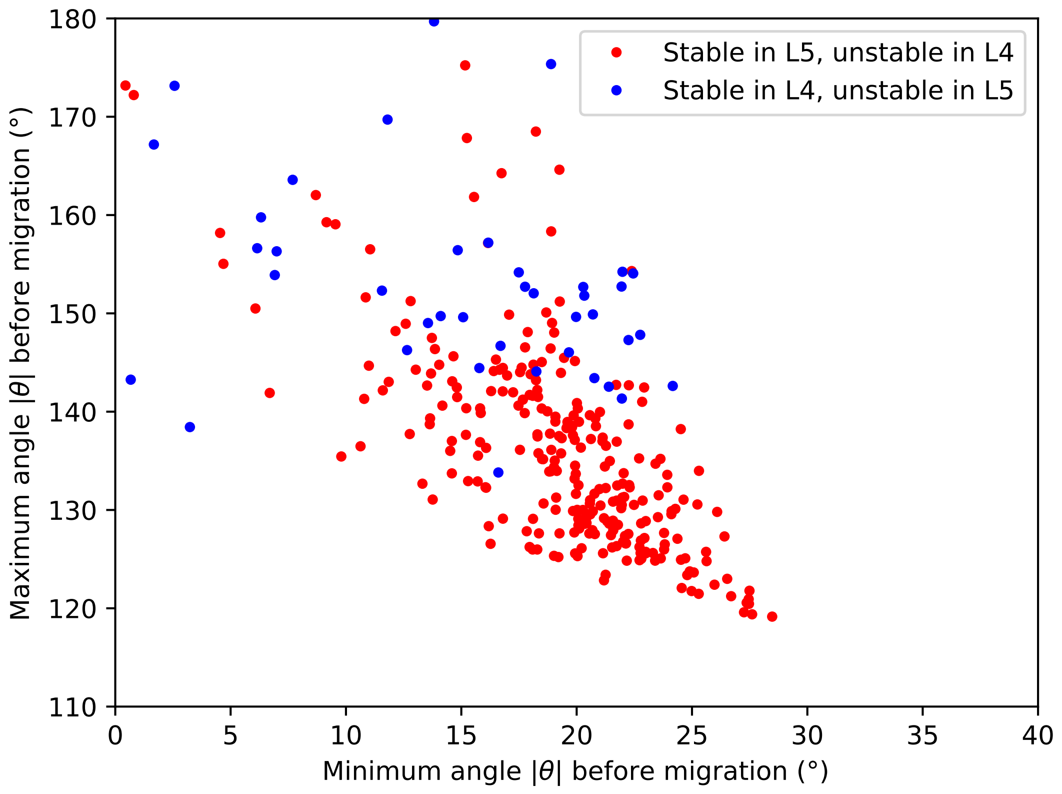


Figure 6.7: Plots of the initial minimum and maximum angle θ of some Trojans relative to Jupiter. The stability is determined after a migration of Jupiter over 1 AU prescribed by an exponential function with parameter $\tau = 1000$ years and the current Jupiter's orbital eccentricity is used. The total simulation time is one million years. The Trojans considered in both cases are the same and are actual Trojans from the NASA database[14] or symmetric copies of those. For this plot. Trojans that led to unstable Trojans for both the original Trojan and a copy or to stable Trojans in both cases, are omitted to emphasize the different between the original Trojans and the copies.

6.4. Inclination

Although we have discussed the influence of the initial inclinations in the section above, we have not yet discussed whether the migration could lead to a different final inclination distribution for the L4 and L5 Trojans and whether the current observed high inclinations angles could also have occurred due to the migration. For this, we first have a look at figure 5.4. The first thing that stands out is that there does not is a clear difference between the final inclinations for the L4 and the L5 Trojans, apart from the fact that more L4 Trojans are lost. The second aspect that stands out, is that most Trojans have reduced their inclination angles, which would imply that the high inclinations are not caused by the outward migration. However, if we zoom in on the stable Trojans with initial inclination less than 5 degrees, as shown in figure 5.5, we see that in this range more Trojans have increased their inclinations. Therefore since the early Trojans had nearly co planar orbits, it is likely that the Trojans have risen inclinations due to the migration, but this could not explain the high inclinations we find today. In fact, figure 5.4 imply that it is also unlikely that the high inclinations where existing before the migration, since these inclinations would decrease due to the migration. Therefore, the most likely explanation is that the outward migration of Jupiter has caused a first increase in inclination for the nearly coplanar initial Trojans, but that this effect is strengthened by a different phenomenon that occurred after the migration. If one has a look at figures 5.6 and 5.7, one would not make different conclusions for the eccentric case.

Apart from the increase or decrease in inclinations, there appears to be a minor difference in the inclination distribution for the L4 and the L5 Trojans as can be seen in the subplots in figures 5.5. For the L5 Trojans, it namely seems that there are higher increases in inclinations possible than for the L4 Trojans. A remarkable aspect is that this is exactly complementary to the eccentric case in figure 5.7, where there are higher increases possible for the L4 Trojans than for the L5 Trojans. All in all, we conclude that the eccentricities have an effect on the inclination distribution and that the migration could cause an asymmetric inclination distribution for the L4 and L5 Trojans, but the effects are marginal for our simulations and therefore hard to separate from numerical and statistical errors. Furthermore, the exact differences are strongly dependent on Jupiters orbital eccentricity.

6.5. Total simulated time

It is desired to minimize the runtime of the simulation, so that many simulation can be done in the time given for the research. Li et al. showed that the simulated time of one million years is sufficient to cover the influence of the migration, because when they extended the simulations to a billion years, there only is a ten percent increase in the number of L4 and L5 Trojans, which they explain without the migration by the research of Romina et al.[3]. It has to be noted that the researches are not entirely comparable as Li did not for example investigate the effect of Saturn on this long term, while Romina et al.[3] did. Furthermore, we use a slightly different model, so our results do not have to be one to be compatible. Therefore, we have investigated this effect partly. Since we had many Trojans to be simulated under many conditions using a laptop, we first tried to run the simulations using a 100000 years. However, when we compared this simulation with the longer simulation of one million years, it turned out that the ratio L4:L5 was still increasing. Therefore, the short term simulations give insight in the migration, but the full effect of the migration might be missed on the long term. Therefore, we also implemented a one million year simulation.

Although Li et al. found that a one million year simulation is sufficient for their research, it might not be the case for our research, since our initial conditions are different. It is possible that a longer simulation indeed does not yield a different result for the Trojans with large resonant angles used by them, but does yield a different result for the initial conditions we used. The shorter simulated time could also be an alternative explanation of why the current Trojans have smaller resonant angles than in the resulting Trojans of Li's research: the small resonant angles where pre-existent but only after a very long time the effect of the migration lead to an asymmetry in this small resonant angle Trojans in contrast to the Trojans with high resonant angles. Therefore, we investigated if we could exclude this effect by selecting a thousand random Trojans and simulate them five times longer than the previous case. The simulation was done without Jupiter's orbital eccentricity and of course compared to the simulation without this eccentricity. The results are shown in table 6.7. From the table we conclude that there is indeed a difference when we simulate longer. Although the effect appears to be smaller than compared

L4’	L4’	L5’	HS’	Unstable’	Total
L4	936	0	10	12	958
L5	0	954	3	2	959
HS	1	6	4	3	14
Unstable	2	3	8	56	69
Total	939	963	25	73	2000

(b) One million years

L4’	L4’	L5’	HS’	Unstable’	Total
L4	924	0	0	34	958
L5	0	950	0	9	959
HS	1	3	0	10	14
Unstable	1	3	0	65	69
Total	926	956	0	118	2000

(c) Five million years

Table 6.7: Trojans of each type after a migration of 1 AU of Jupiter from its current position over two different simulation times using a time step of 1/3 years. The migration rate is described equation 4.1 with $\tau = 1000$ y. The initial Trojans are 1000 randomly selected actual Trojans from [14] with its symmetric copies to obtain as many L4 and L5 Trojans initially. The prime denotes the types after the migration and HS stands for Horse shoe.

to the increase from 100000 years to one million years, we see that the L4:L5 ratio is still changing. Since Jupiter’s orbital eccentricity is not considered in this simulation, the effect can only be due to the migration or numerical errors. It appears that the estimation for the simulation time of one million years as used in Li et al., might not be sufficient for our set of Trojans. In a further research, the best simulation time could be determined more carefully.

6.6. Further research

In general, research on the Trojans is a relevant topic because it yields much information about the early development of the Solar System. There are also practical applications for the knowledge about the Trojans orbits. So could we potentially use the knowledge about stable Trojans to put a satellite in the Earth’s L4 and L5 Lagrange points. this could be advantageous because such a satellite has always the same position relative to the Earth and the Sun, but it does not need any fuel for stabilisation. Also, we could potentially could our knowledge about unstable Trojans and the Trojan distributions to identify potential dangers to Earth, since it is known that it is possible that some Trojans could collide with the Earths orbit when becoming unstable[22]. Although the applications are broad, we will focus in our further research recommendations related more closely to the migration of Jupiter, in line with the specific focus of this project.

Apart from the suggestions made earlier in this chapter, we have three other suggestions for further research, namely merging with the Nice model, considering inward migrations and finally considering the gravitational attractions of the gas giant on the Trojans themselves. These three suggestions are elaborated in the next three paragraphs.

Merging with Nice model From our results, it becomes clear that the Jupiter’s orbital eccentricity plays a crucial role for the stability of Trojans during the migration. Li et al suggested that the Jupiter’s orbital eccentricity might have been higher at the time of migration[2] than Jupiter’s current eccentricity. Although Li et al.[2] have investigated this effect for different eccentricity on part of their results, they used a constant eccentricity like we did for both the migration and the non-migration part. However, we know that gravitational encounters by the other planets do not only lead to a migration of a planet, but alters the orbital eccentricity of the planet as well. Also, the research of Li et al. did not include the effect of a combined change in eccentricity, migration rate and migration distance, while they might be closely related due to the Nice model. We therefore suggest that in further research, simulations of the Nice model are used as a data set for Jupiter’s time dependent position instead of the modeled function for this distance and constant eccentricities. These simulations could both help in the research of finding the best initial conditions for our Solar System for the Nice model, as supporting the theory about the L4 and L5 asymmetry. As described in section 6.1.1, our code is of great advantage for such a research over the code used by Li.

Inward migrations In the Nice model, there where several different types of migrations of Jupiter, both inward and outward. Li et al. predicted that there was one more outward migration than inward and that the effect of one outward migration was not cancelled out by an inwards one. This ’extra’ migration was hypothesized to cause the different Trojan populations. However, these cancellations might not be a good assumption, since it is very well possible that the inward migrations lead some Trojans to obtain high resonant angles. Our data suggests that this is the only possible position where we can obtain stable L4 Trojans after migration, with unstable L5 copy. To investigate this result, once again our model could be used and it could even be combined with data from Nice model simulations as described above.

Gravitational attractions by close encounters Li et al. suggested the model could be improved by taking into account the effect of Saturn, Uranus and Neptune. They referred to two papers of Marzari et al. that indicates that direct perturbations by Saturn indeed increases the L4:L5 ratio, but also mention that this is on a timescale that is one or two orders of magnitude bigger than our their and our simulations. There are other influences of Jupiter that might be more present, for which they refer to Freistetter. Nonetheless, they mention that it is very hard to investigate the perturbations, since it is very uncertain how the Solar System was exactly in its early stage and they mention this could lead to unreliable results due to arbitrarily designed models[2].

However, they did not consider the effect of the fifth gas giant. As described in 2.1, Jupiter's migration is caused by the gravitational attraction of this giant at close encounters. However, since the Trojans have all more or less the same distance to the Sun as Jupiter, it is very well possible that the close encounters of this gas giant also had close encounters with the Trojans, resulting in similar accelerations as on Jupiter. We hypothesize that these influences are much more present than the influence of Saturn, since the effect of the gas giant on Jupiter was also bigger than that of Saturn according to the Nice model. Therefore, we suggest that in further research the close encounters of the fifth gas giant are not only considered when determining Jupiter's migration, but also considered as an additional force to the Trojans. Several models could be used, but as a simple approximation to implement it in the current model, we suggest that in our model the Trojans obtain the same acceleration as Jupiter. Using our current description of the evolution of Jupiter's semi-major axis $R_J = R_{\text{initial}} + (1 - \text{exp}(-t/\tau))\Delta R$, this lead to an acceleration a_j of[9]

$$a_J = \frac{\hat{v}}{\tau}\left(\sqrt{\frac{Gm_S}{R_J}} - \sqrt{\frac{Gm_S}{R_J + \Delta R_J}}\right) \text{exp}\left(-\frac{t}{\tau}\right)$$

This acceleration then has to be converted to our cylindrical system and then be added to the equation of motion. Of course this is a rough estimate of the acceleration, since the Trojans are not at the same position as Jupiter and probably better estimates can be found. This is left for further research.

7 Conclusion

Ever since their discovery, the Trojans have raised many questions among astrophysicists. One of the mysteries about the Trojans is its asymmetric distribution in the L4 and the L5 points. The ratio between the number of L4 and L5 Trojans is estimated to be about 1.6, which can not be explained by simulations to its current orbits. In this research, it was investigated whether an outward migration of Jupiter, comparable to one prescribed by the Nice model, could cause its current distribution. For this a recent research of Li et al. is used as a basis. In this research they found the ratio of 1.6 for a restricted initial distribution of Trojans. We extended the research by using the actual Trojans as initial distribution to obtain a broader data set. To obtain as many L4 as L5 Trojans initially, we doubled the data set by using pairs of symmetric copies. In this manner, we have investigated what initial conditions are essential for the L4:L5 ratio and we investigated whether the current high inclinations could be explained by the outward migration, since theories predict the initial Trojan orbots were nearly coplanar. We used Hamiltonian mechanics in the rotating frame of Jupiter for our model and we used a recently found improved Yoshida integrator to obtain a symplectic fourth order integrator for the non-separable Hamiltonian. Firstly, Jupiter's orbital eccentricity was ignored. We found that in fact there were more stable L5 Trojans than L4 Trojans after the migration, which is contrary to our expectation. When the initial inclinations were smaller than 5 degrees, for most Trojans the inclination was indeed increased. However, it did not increase to the current inclinations which reach values to about 40 degrees and Trojans that had high initial inclinations in fact had their inclinations decreased. It follows that the migration could have caused the higher inclinations, but the most extreme inclinations must have occurred after the migration.

We found that the L4:L5 ratio could only explain today's high value if the Trojans initially had high resonant angles. Furthermore, we found that Jupiters orbital eccentricity has a major influence on the results and leads to an increase of any asymmetry in the populations due to the migration. Furthermore, we investigated different model functions for the migration and found that both the migration speed, total migration duration and the type of function describing the migration alters the results. Since it is difficult to find a proper function for the evolution of Jupiter's major semi-axis and orbital eccentricity during the migration, we suggest to use simulations of the Nice model for these two quantities. Our model is of great use for this, since it allows to directly load these values instead of having to describe them as a force on Jupiter. Another effect that could be investigated during further research, is to not only let Jupiter be attracted during close encounters with the fifth gas giant, but to let the Trojans experience this force as well.

References

[1] URL: https://www.overleaf.com/project/5f0d576145959f000121398a (visited on 06/06/2023).

[2] Li et al. “Asymmetry in the number of L4 and L5 Jupiter Trojans driven by jumping Jupite”. In: *Physica Scripta* 669.68 (2023). DOI: 10.10510004-6361202244443. URL: https://doi.org/10.1051/0004-6361/202244443.

[3] Romina P. Di Sisto, Ximena S. Ramos, and Cristián Beaugé. “Giga-year evolution of Jupiter Trojans and the asymmetry problem”. In: *Icarus* 243 (2014), pp. 287–295. ISSN: 0019-1035. DOI: https://doi.org/10.1016/j.icarus.2014.09.002. URL: https://www.sciencedirect.com/science/article/pii/S0019103514004643.

[4] National Ocean Service US government. *What is GPS*. URL: https://oceanservice.noaa.gov/facts/gps.html (visited on 04/21/2023).

[5] Kate Howels. *What was the Chelyabinsk meteor event?* 2023. URL: https://www.planetary.org/articles/what-was-the-chelyabinsk-meteor-event (visited on 04/21/2023).

[6] NASA’s Goddard Space Flight Center Conceptual Image Lab. *How Were the Trojan Asteroids Discovered and Named?* Author belongs to the image used. URL: https://www.nasa.gov/feature/goddard/2021/how-were-the-trojan-asteroids-discovered-and-named (visited on 08/13/2023).

[7] *Leapfrog integration*. URL: https://en.wikipedia.org/wiki/Leapfrog_integration.

[8] Junjie Luo et al. “EXPLICIT SYMPLECTIC-LIKE INTEGRATORS WITH MIDPOINT PERMUTATIONS FOR SPINNING COMPACT BINARIES”. In: *Physica Scripta* 834.1 (Jan. 2017), pp. 1–5. DOI: 10.38471538-4357/834/1/64. URL: https://dx.doi.org/10.3847/1538-4357/834/1/64.

[9] Renu Malhotra. “The Origin of Pluto’s Orbit: Implications for the Solar System Beyond Neptune”. In: *aj* 110 (July 1995), p. 420. DOI: 10.1086/117532. arXiv: astro-ph/9504036 [astro-ph].

[10] F Marzari and H Scholl. “The growth of Jupiter and Saturn and the capture of Trojans”. In: *Astronomy and Astrophysics, v. 339, p. 278-285 (1998)* 339 (1998), pp. 278–285.

[11] F. Marzari et al. “Origin and Evolution of Trojan Asteroids”. In: *Asteroids III* (), pp. 725–739.

[12] Carl D. Murray and Stanley F. *Solar System Dynamics*. Cambridge university press, 2009. DOI: 10.1017/CBO9781139174817.003. URL: https://doi.org/10.1017/CBO9781139174817.003.

[13] NASA. *Jupiter Fact Sheet*. URL: https://nssdc.gsfc.nasa.gov/planetary/factsheet/jupiterfact.html (visited on 04/21/2023).

[14] NASA. *Small-Body Database*. URL: https://ssd.jpl.nasa.gov/tools/sbdb_query.html (visited on 04/21/2023).

[15] David Nesvorny. *Young Solar System’s fifth giant planet?* Sept. 2011. URL: https://arxiv.org/abs/1109.2949.

[16] D. Nesvorný, D. Vokrouhlický, and A. Morbidelli. “CAPTURE OF TROJANS BY JUMPING JUPITER”. In: (2013). URL: https://arxiv.org/pdf/1303.2900.pdf.

[17] Seth B. Nicholson. “The Trojan Asteroids”. In: *Astronomical Society of the Pacific Leaflets* 8.381 (1962), p. 239.

[18] The Planets. *What Are The Trojan Asteroids?* URL: https://theplanets.org/what-are-the-trojan-asteroids (visited on 07/12/2023).

[19] E.M. Shoemaker, C.S. Shoemaker, and R.F. Wolfe. “Trojan asteroids - Populations, dynamical structure and origin of the L4 and L5 swarms”. In: (Mar. 1989), p. 487.

[20] Bruno Sicardy and Véronique Dubois. “Co-Orbital Motion with Slowly Varying Parameters”. In: *Celestial Mechanics and Dynamical Astronomy* 86.4 (Aug. 2003), pp. 321–350.

[21] John R. Taylor. *Classical Mechanics*. 9th ed. printing 2020. Californy: University Sience Books, 2005.

[22] Nola Taylor Tillman. *Dangerous Asteroids May Be Lurking in Jupiter’s Shadow*. URL: `https://www.space.com/hidden-jupiter-asteroids-threaten-earth.html` (visited on 08/19/2023).

[23] Github user. *equipotentials.py*. URL: `https://github.com/zingale/astro_animations/blob/main/binary_exoplanets/equipotentials/equipotentials.pys` (visited on 04/12/2023).

[24] Visser. “A&A proofs: manuscript”. In: (2023).

[25] P. M. Visser. “Collision detection for N-body Kepler systems”. In: (Sept. 2022). DOI: `10.1051/0004-6361/202243754`. URL: `https://doi.org/10.1051%5C%2F0004-6361%5C%2F202243754`.

[26] C. Vuik et al. *Numerical Methods For Ordinary Differential Equations*. 2nd ed. Delft Academic Press, 2016.

[27] Wikipedia. *Lagrange point*. URL: `https://simple.wikipedia.org/wiki/Lagrange_point` (visited on 06/21/2023).

[28] Wikipedia. *Orbital inclination*. URL: `https://en.wikipedia.org/wiki/Orbital_inclination` (visited on 07/13/2023).

A Numerical methods

A.1. Original Yoshida integrator

The Yoshida integrator is a fourth order method to solve differential equations of the from $\ddot{x} = A(x)$[7], given by[7]

$$\begin{aligned}
x_i^1 &= x_i + c_1\, v_i\, \Delta t,\\
v_i^1 &= v_i + d_1\, a(x_i^1)\, \Delta t,\\
x_i^2 &= x_i^1 + c_2\, v_i^1\, \Delta t,\\
v_i^2 &= v_i^1 + d_2\, a(x_i^2)\, \Delta t,\\
x_i^3 &= x_i^2 + c_2\, v_i^2\, \Delta t,\\
v_i^3 &= v_i^2 + d_1\, a(x_i^3)\, \Delta t,\\
x_{i+1} \equiv x_i^4 &= x_i^3 + c_1\, v_i^3\, \Delta t,\\
v_{i+1} \equiv v_i^4 &= v_i^3
\end{aligned} \tag{A.1}$$

where $w_0 = -\frac{\sqrt[3]{2}}{2-\sqrt[3]{2}}$, $w_1 = \frac{1}{2-\sqrt[3]{2}}$,$c_1 = \frac{w_1}{2}$,$c_2 = \frac{w_0+w_1}{2}$,$d_1 = w_1$,$d_2 = w_0$. In this project, the equations of motion are not of the form $\ddot{x} = A(x)$ and therefore this method might not be fourth order, symplectic and/or stable. In section 6.1 it is shown that this common integrator is indeed not applicable to these equations of motion.

A.2. Stability analysis midpoint Yoshida integrator

In section 6.1, the performance of the midpoint Yoshida integrator is tested near the equilibrium. From these analysis, it appeared that the method is symplectic, stable and fourth order. The latter two properties are here proven analytically for a near-stable Trojan. This analysis is important to determine an upper bound for the time step used when numerically integrating the equations of motion for a near equilibrium Trojan. Due to $H_2(h/2)$ we map:

$$[\vec{r},\tilde{p}] \to [\vec{r},\tilde{p}] + \frac{h}{2}\mathbf{J}[\tilde{r},\vec{p}]$$

$$H_1(h)H_2(h/2) : [\vec{r},\tilde{p}] \to [\vec{r},\tilde{p}]+\frac{h}{2}\mathbf{J}[\tilde{r},\vec{p}], [\tilde{r},\vec{p}] \to [\tilde{r},\vec{p}]+h\mathbf{J}([\vec{r},\tilde{p}]+\frac{h}{2}\mathbf{J}[\tilde{r},\vec{p}]) = [\tilde{r},\vec{p}]+h\mathbf{J}[\vec{r},\tilde{p}]+\frac{h^2}{2}\mathbf{J}^2[\tilde{r},\vec{p}]$$

$$H_2(h/2)H_1(h)H_2(h/2) : [\vec{r},\tilde{p}] \to [\vec{r},\tilde{p}] + \frac{h}{2}\mathbf{J}[\tilde{r},\vec{p}] + \frac{h}{2}\mathbf{J}([\tilde{r},\vec{p}] + h\mathbf{J}[\vec{r},\tilde{p}] + \frac{h^2}{2}\mathbf{J}^2[\tilde{r},\vec{p}]) =$$

$$[\vec{r},\tilde{p}] + \frac{h}{2}\mathbf{J}[\tilde{r},\vec{p}] + \frac{h}{2}\mathbf{J}[\tilde{r},\vec{p}] + \frac{h^2}{2}\mathbf{J}^2[\vec{r},\tilde{p}] + \frac{h^3}{4}\mathbf{J}^3[\tilde{r},\vec{p}]$$

$$A_2(d_1dt) : [\vec{r},\tilde{p}] \to (I + \frac{(d_1dt)^2}{2}\mathbf{J}^2)[\vec{r},\tilde{p}] + (d_1dt\mathbf{J} + \frac{(d_1dt)^3}{4}\mathbf{J}^3)[\tilde{r},\vec{p}],$$

$$[\tilde{r}, \vec{p}] \to (I + \frac{(d_1 dt)^2}{2}\mathbf{J}^2)[\tilde{r}, \vec{p}] + d_1 dt\mathbf{J}[\vec{r}, \tilde{p}]$$

For simplicity, we define $w := d_1 + d_2$

$$A_2(d_2 dt)A_2(d_1 dt) : [\vec{r}, \tilde{p}] \to (I + \frac{(d_2 dt)^2}{2}\mathbf{J}^2)((I + \frac{(d_1 dt)^2}{2}\mathbf{J}^2)[\vec{r}, \tilde{p}] + (d_1 dt\mathbf{J} + \frac{(d_1 dt)^3}{4}\mathbf{J}^3)[\tilde{r}, \vec{p}])$$

$$+(d_2 dt\mathbf{J} + \frac{(d_2 dt)^3}{4}\mathbf{J}^3)((I + \frac{(d_1 dt)^2}{2}\mathbf{J}^2)[\tilde{r}, \vec{p}] + d_1 dt\mathbf{J}[\vec{r}, \tilde{p}]) =$$

$$(I^2 + \frac{d_1^2 + d_2^2}{2}dt^2\mathbf{J}^2 + \frac{d_1^2 d_2^2 dt^4}{4}\mathbf{J}^4 + \frac{d_1 d_2^3 dt^4}{4}\mathbf{J}^4 + d_1 d_2 dt^2\mathbf{J}^2)[\vec{r}, \tilde{p}]+$$

$$(d_1 dt\mathbf{J} + \frac{d_1 d_2^2 dt^3}{2}\mathbf{J}^3 + \frac{d_1^3 dt^3}{4}\mathbf{J}^3 + \frac{d_1^3 d_2^2}{8}dt^5\mathbf{J}^5 + d_2 dt\mathbf{J} + \frac{d_2^3 dt^3}{4}\mathbf{J}^3 + \frac{d_1^2 d_2 dt^3}{2}\mathbf{J}^3 + \frac{d_1^2 d_2^3 dt^5}{8}\mathbf{J}^5)[\tilde{r}, \vec{p}] =$$

$$(I + \frac{w^2}{2}dt^2\mathbf{J}^2 + \frac{d_1 d_2^2 w}{4}dt^4\mathbf{J}^4)[\vec{r}, \tilde{p}] + (wdt\mathbf{J} + \frac{d_1^3 + d_2^3 + 2d_1 d_2 w}{4}dt^3\mathbf{J}^3+$$

$$\frac{d_1^2 d_2^2 w}{8}dt^5\mathbf{J}^5)[\tilde{r}, \vec{p}]$$

$$[\tilde{r}, \vec{p}] \to (I + \frac{(d_2 dt)^2}{2}\mathbf{J}^2)((I + \frac{(d_1 dt)^2}{2}\mathbf{J}^2)[\tilde{r}, \vec{p}] + d_1 dt\mathbf{J}[\vec{r}, \tilde{p}]) + d_2 dt\mathbf{J}((I + \frac{(d_1 dt)^2}{2}\mathbf{J}^2)[\vec{r}, \tilde{p}]+$$

$$(d_1 dt\mathbf{J} + \frac{(d_1 dt)^3}{4}\mathbf{J}^3)[\tilde{r}, \vec{p}]) =$$

$$(I^2 + \frac{d_1^2 + d_2^2}{2}dt^2\mathbf{J}^2 + \frac{d_1^2 d_2^2}{4}dt^4\mathbf{J}^4)[\tilde{r}, \vec{p}] + (d_1 dt\mathbf{J} + \frac{d_1 d_2^2}{2}dt^3\mathbf{J}^3)[\vec{r}, \tilde{p}]+$$

$$(d_2 dt\mathbf{J} + \frac{d_2 d_1^2}{2}dt^3\mathbf{J}^3)[\vec{r}, \tilde{p}] + (d_1 d_2 dt^2\mathbf{J}^2 + \frac{d_1^3 d_2 dt^4}{4}\mathbf{J}^4)[\tilde{r}, \vec{p}] =$$

$$(I + \frac{w^2 dt^2}{2}\mathbf{J}^{\mathbf{2}} + \frac{d_1^2 d_2 w}{4}dt^4\mathbf{J}^4)[\tilde{r}, \vec{p}] + wdt\mathbf{J} + \frac{d_1 d_2 w}{2}dt^3\mathbf{J}^3)[\vec{r}, \tilde{p}]$$

$A_2(d_1 dt)A_2(d_2 dt)A_2(d_1 dt)$:

$$[\vec{r}, \tilde{p}] \to (I{+}\frac{(d_1 dt)^2}{2}\mathbf{J}^2)((I{+}\frac{w^2}{2}dt^2\mathbf{J}^2{+}\frac{wd_1 d_2^2}{4}dt^4\mathbf{J}^4)[\vec{r}, \tilde{p}]{+}(wdt\mathbf{J}{+}\frac{d_1^3 + d_2^3 + 2d_1 d_2 w}{4}dt^3\mathbf{J}^3{+}\frac{d_1^2 d_2^2 w}{8}dt^5\mathbf{J}^5)[\tilde{r}, \vec{p}])+$$

$$(d_1 dt\mathbf{J} + \frac{(d_1 dt)^3}{4}\mathbf{J}^3)((I + \frac{w^2}{2}dt^2\mathbf{J}^{\mathbf{2}} + \frac{wd_1^2 d_2}{4}dt^4\mathbf{J}^4)[\tilde{r}, \vec{p}] + (wdt\mathbf{J} + \frac{wd_1 d_2}{2}dt^3\mathbf{J}^3)[\vec{r}, \tilde{p}]) =$$

$$(I^2 + \frac{d_1^2 + w^2}{2}dt^2\mathbf{J}^2 + \frac{wd_1 d_2^2 + w^2 d_1^2}{4}dt^4\mathbf{J}^4 + \frac{wd_1^3 d_2^2}{8}dt^6\mathbf{J}^6 + wd_1 dt^2\mathbf{J}^2 + \frac{wd_1^2 d_2}{2}dt^4\mathbf{J}^4+$$

$$\frac{wd_1^3}{4}dt^4\mathbf{J}^4 + \frac{wd_1^4 d_2}{8}dt^6\mathbf{J}^6)[\vec{r}, \tilde{p}]+$$

$$(wdt\mathbf{J} + \frac{d_1^3 + d_2^3 + 2wd_1 d_2 + 2wd_1^2}{4}dt^3\mathbf{J}^3 + d_1^2\frac{d_1^3 + d_2^3 + 2wd_1 d_2 + wd_2^2}{8}dt^5\mathbf{J}^5 + \frac{wd_1^4 d_2^2}{16}dt^7\mathbf{J}^7$$

$$+d_1 dt\mathbf{J} + \frac{d_1^3 + 2w^2 d_1}{4}dt^3\mathbf{J}^3 + \frac{2wd_1^3 d_2 + w^2 d_1^3}{8}dt^5\mathbf{J}^5 + \frac{wd_1^5 d_2}{16}dt^7\mathbf{J}^7)[\tilde{r}, \vec{p}] =$$

$$(I + \frac{1}{2}dt^2\mathbf{J}^2 + \frac{w^2 d_1}{4}dt^4\mathbf{J}^4 + \frac{w^2 d_1^3 d_2}{8}dt^6\mathbf{J}^6)[\vec{r}, \tilde{p}]+$$

$$(dt\mathbf{J} + \frac{1 + 2wd_1(d_1 - 1)}{4}dt^3\mathbf{J}^3 + \frac{w^2 d_1^2(d_2 + 1)}{8}dt^5\mathbf{J}^5 + \frac{w^2 d_1^4 d_2}{16}dt^7\mathbf{J}^7)[\tilde{r}, \vec{p}]$$

$$[\tilde{r},\vec{p}] \rightarrow (I+\frac{(d_1dt)^2}{2}\mathbf{J}^2)((I+\frac{w^2dt^2}{2}\mathbf{J^2}+\frac{d_1^2d_2w}{4}dt^4\mathbf{J}^4)[\tilde{r},\vec{p}]+(wdt\mathbf{J}+\frac{d_1d_2w}{2}dt^3\mathbf{J}^3)[\vec{r},\tilde{p}])+$$
$$d_1dt\mathbf{J}((I+\frac{w^2}{2}dt^2\mathbf{J}^2+\frac{d_1d_2^2w}{4}dt^4\mathbf{J}^4)[\vec{r},\tilde{p}]+(wdt\mathbf{J}+\frac{d_1^3+d_2^3+2d_1d_2w}{4}dt^3\mathbf{J}^3+$$
$$\frac{d_1^2d_2^2w}{8}dt^5\mathbf{J}^5)[\tilde{r},\vec{p}])=$$
$$(I^2+\frac{d_1^2+w^2}{2}dt^2\mathbf{J}^2+\frac{w^2d_1^2+wd_1^2d_2}{4}dt^4\mathbf{J}^4+$$
$$\frac{wd_1^4d_2}{8}dt^6\mathbf{J}^6+wd_1dt^2\mathbf{J}^2+d_1\frac{d_1^3+d_2^3+2wd_1d_2}{4}dt^4\mathbf{J}^4+\frac{d_1^3d_2^2w}{8}dt^6\mathbf{J}^6)[\tilde{r},\vec{p}]+$$
$$(wdt\mathbf{J}+\frac{wd_1^2+wd_1d_2}{2}dt^3\mathbf{J}^3+\frac{wd_1^3d_2}{4}dt^5\mathbf{J}^5+d_1dt\mathbf{J}+$$
$$\frac{w^2d_1}{2}dt^3\mathbf{J}^3+\frac{wd_1^2d_2^2}{4}dt^5\mathbf{J}^5)[\vec{r},\tilde{p}]=$$
$$(I+\frac{1}{2}dt^2\mathbf{J}^2+\frac{w^2d_1}{4}dt^4\mathbf{J}^4+\frac{w^2d_1^3d_2}{8}dt^6\mathbf{J}^6)[\tilde{r},\vec{p}]+$$
$$(dt\mathbf{J}+w^2d_1dt^3\mathbf{J}^3+\frac{w^2d_1^2d_2}{4}dt^5\mathbf{J}^5)[\vec{r},\tilde{p}]$$

Now averaging the results and using the fact that originaly $[\vec{r},\tilde{p}]=[\tilde{r},\vec{p}]$ yields:

$$[\vec{r},\vec{p}] \rightarrow (I/2+\frac{1}{4}dt^2\mathbf{J}^2+\frac{w^2d_1}{8}dt^4\mathbf{J}^4+\frac{w^2d_1^3d_2}{16}dt^6\mathbf{J}^6)+$$
$$dt\mathbf{J}/2+\frac{1+2wd_1(d_1-1)}{8}dt^3\mathbf{J}^3+\frac{w^2d_1^2(d_2+1)}{16}dt^5\mathbf{J}^5+\frac{w^2d_1^4d_2}{32}dt^7\mathbf{J}^7+I/2+\frac{1}{4}dt^2\mathbf{J}^2+\frac{w^2d_1}{8}dt^4\mathbf{J}^4+\frac{w^2d_1^3d_2}{16}dt^6\mathbf{J}^6+$$
$$dt\mathbf{J}/2+\frac{w^2d_1}{4}dt^3\mathbf{J}^3+\frac{w^2d_1^2d_2}{8}dt^5\mathbf{J}^5)[\vec{r},\vec{p}])=$$
$$(I+dt\mathbf{J}+\frac{1}{2}dt^2\mathbf{J}^2+\frac{1+2w^2d_1}{8}dt^3\mathbf{J}^3+\frac{w^2d_1}{4}dt^4\mathbf{J}^4+$$
$$\frac{w^2d_1^3d_2}{8}dt^5dt^5\mathbf{J}^5+\frac{wd_1^3(w+d_2)}{16}dt^6\mathbf{J}^6+\frac{w^2d_1^4d_2}{32}dt^7\mathbf{J}^7)[\vec{r},\vec{p}]=$$

$$(I+dt\mathbf{J}+\frac{1}{2}dt^2\mathbf{J}^2+\frac{1}{6}dt^3\mathbf{J}^3+\frac{1}{24}dt^4\mathbf{J}^4+\frac{1-2\sqrt[3]{2}}{48(2-\sqrt[3]{2})^2}dt^5\mathbf{J}^5-\frac{\sqrt[3]{2}}{48(2-\sqrt[3]{2})^3}dt^6\mathbf{J}^6-\frac{\sqrt[3]{2}}{192(2-\sqrt[3]{2})^4}dt^7\mathbf{J}^7)[\vec{r},\vec{p}] \tag{A.2}$$

Comparing this with the Taylor series, one indeed notices that the method is a fourth order method.

Using, Python the stability region could be plotted in figure A.1. Drawing a half circle in the same image, as shown in A.2, one obtains that for $\Re(\lambda)\leq 0$, $dt \leq \frac{2.5}{|\lambda|}$ is a sufficient condition for stability.

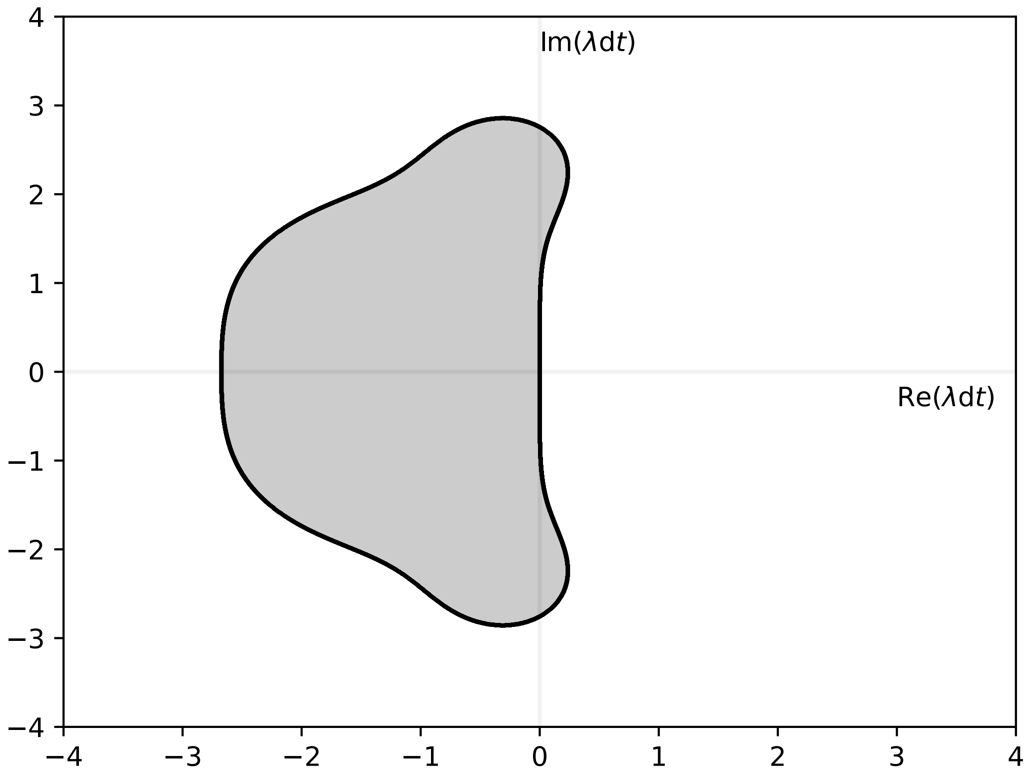


Figure A.1: Stability region for the midpoint Yoshida integrator

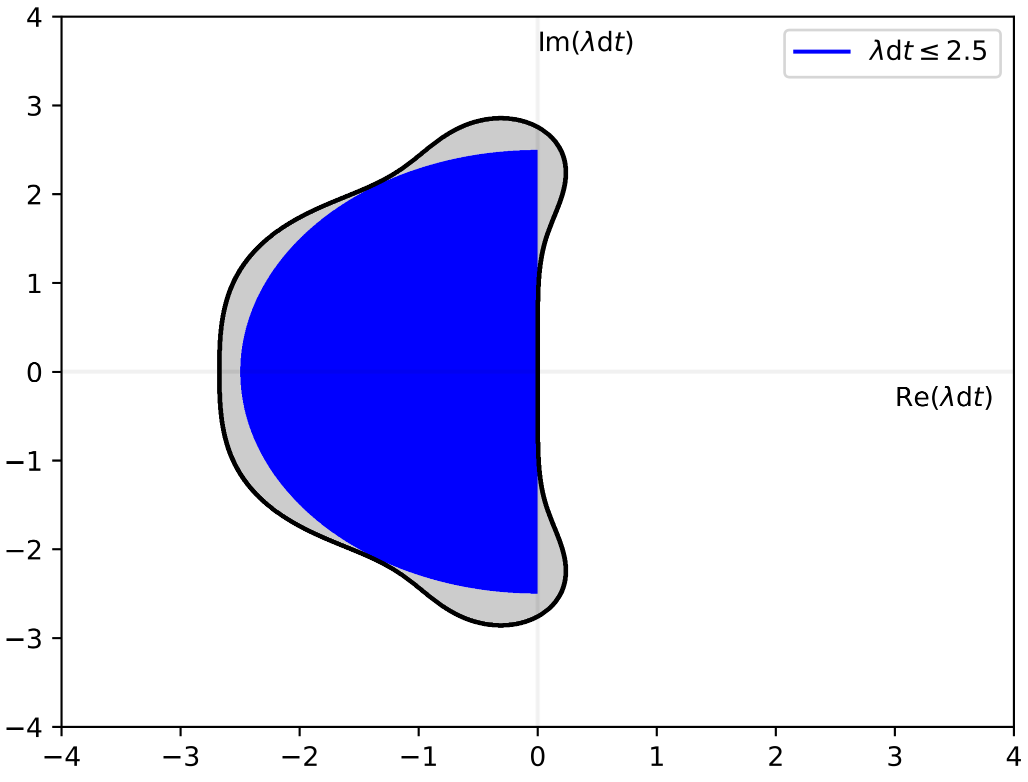


Figure A.2: Stability region for the midpoint Yoshida integrator with sufficient stability condition

B
Jacobian

The Jacobian of the system of differential equations 3.18 is given by:

$$J = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ -\frac{2l_z}{r^3} & 0 & 0 & 0 & \frac{1}{r^2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -\frac{3l_z^2}{r^4} - \frac{\partial^2 V}{\partial r^2} & -\frac{\partial^2 V}{\partial r \partial \theta} & -\frac{\partial^2 V}{\partial r \partial z} & 0 & \frac{2l_z}{r^3} & 0 \\ -\frac{\partial^2 V}{\partial \theta \partial r} & -\frac{\partial^2 V}{\partial \theta^2} & -\frac{\partial^2 V}{\partial \theta \partial z} & 0 & 0 & 0 \\ -\frac{\partial^2 V}{\partial z \partial r} & -\frac{\partial^2 V}{\partial z \partial \theta} & -\frac{\partial^2 V}{\partial z^2} & 0 & 0 & 0 \end{bmatrix}$$

Using our initial conditions, this yields:

$$J \approx \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ -5.52161612 \cdot 10^{-32} & 0 & 0 & 0 & 1.65032152 \cdot 10^{-24} & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 5.58920164 \cdot 10^{-16} & -3.78785157 \cdot 10^{-7} & 4.19854017 \cdot 10^{-19} & 0 & 5.52161612 \cdot 10^{-32} & 0 \\ -3.78785157 \cdot 10^{-7} & 8.60581879 \cdot 10^{4} & -2.56484810 \cdot 10^{-10} & 0 & 0 & 0 \\ -2.79950087 \cdot 10^{-16} & -2.56484810 \cdot 10^{-10} & -2.79950087 \cdot 10^{-16} & 0 & 0 & 0 \end{bmatrix}$$

Usually, one estimates the eigenvalues, for example Gresgorin's theorem, because of its simplicity. Now, it is easily observed that Gresgorin's theorem bounds:

$$|\lambda| \leq 8.606 \cdot 10^4 \tag{B.1}$$

Therefore, one obtains a stable time step:

$$dt \leq \frac{2.5}{8.606 \cdot 10^4} \approx 2.9 \cdot 10^{-5} \tag{B.2}$$

This is far lower than preferred, thus the theorem doesn't help in this problem. Therefore, one has to calculate the eigenvalues.

$$0 = |J - \lambda I| = \begin{vmatrix} -\lambda & 0 & 0 & 1 & 0 & 0 \\ -\frac{2l_z}{r^3} & -\lambda & 0 & 0 & \frac{1}{r^2} & 0 \\ 0 & 0 & -\lambda & 0 & 0 & 1 \\ -\frac{3l_z^2}{r^4} - \frac{\partial^2 V}{\partial r^2} & -\frac{\partial^2 V}{\partial r \partial \theta} & -\frac{\partial^2 V}{\partial r \partial z} & -\lambda & \frac{2l_z}{r^3} & 0 \\ -\frac{\partial^2 V}{\partial \theta \partial r} & -\frac{\partial^2 V}{\partial \theta^2} & -\frac{\partial^2 V}{\partial \theta \partial z} & 0 & -\lambda & 0 \\ -\frac{\partial^2 V}{\partial z \partial r} & -\frac{\partial^2 V}{\partial z \partial \theta} & -\frac{\partial^2 V}{\partial z^2} & 0 & 0 & -\lambda \end{vmatrix}$$

$$\begin{vmatrix} -\lambda & 0 & 0 & 1 & 0 & 0 \\ -\frac{2l_z}{r^3} & -\lambda & 0 & 0 & \frac{1}{r^2} & 0 \\ 0 & 0 & -\lambda & 0 & 0 & 1 \\ -\frac{3l_z^2}{r^4} - \frac{\partial^2 V}{\partial r^2} & -\frac{\partial^2 V}{\partial r \partial \theta} & -\frac{\partial^2 V}{\partial r \partial z} & -\lambda & \frac{2l_z}{r^3} & 0 \\ -\frac{\partial^2 V}{\partial \theta \partial r} & -\frac{\partial^2 V}{\partial \theta^2} & -\frac{\partial^2 V}{\partial \theta \partial z} & 0 & -\lambda & 0 \\ -\frac{\partial^2 V}{\partial z \partial r} & -\frac{\partial^2 V}{\partial z \partial \theta} & -\frac{\partial^2 V}{\partial z^2} - \lambda^2 & 0 & 0 & 0 \end{vmatrix} =$$

$$\begin{vmatrix} -\lambda & 0 & 0 & 1 & 0 & 0 \\ -\frac{2l_z}{r^3} & -\lambda & 0 & 0 & \frac{1}{r^2} & 0 \\ 0 & 0 & -\lambda & 0 & 0 & 1 \\ -\frac{3l_z^2}{r^4} - \frac{\partial^2 V}{\partial r^2} & -\frac{\partial^2 V}{\partial r \partial \theta} & -\frac{\partial^2 V}{\partial r \partial z} & -\lambda & \frac{2l_z}{r^3} & 0 \\ -\frac{\partial^2 V}{\partial \theta \partial r} & -\frac{\partial^2 V}{\partial \theta^2} & -\frac{\partial^2 V}{\partial \theta \partial z} & 0 & -\lambda & 0 \\ -\frac{\partial^2 V}{\partial z \partial r} & -\frac{\partial^2 V}{\partial z \partial \theta} & -\frac{\partial^2 V}{\partial z^2} - \lambda^2 & 0 & 0 & 0 \end{vmatrix} =$$

$$-\begin{vmatrix} -\lambda & 0 & 0 & 1 & 0 \\ -\frac{2l_z}{r^3} & -\lambda & 0 & 0 & \frac{1}{r^2} \\ -\frac{3l_z^2}{r^4} - \frac{\partial^2 V}{\partial r^2} & -\frac{\partial^2 V}{\partial r \partial \theta} & -\frac{\partial^2 V}{\partial r \partial z} & -\lambda & \frac{2l_z}{r^3} \\ -\frac{\partial^2 V}{\partial \theta \partial r} & -\frac{\partial^2 V}{\partial \theta^2} & -\frac{\partial^2 V}{\partial \theta \partial z} & 0 & -\lambda \\ -\frac{\partial^2 V}{\partial z \partial r} & -\frac{\partial^2 V}{\partial z \partial \theta} & -\frac{\partial^2 V}{\partial z^2} - \lambda^2 & 0 & 0 \end{vmatrix} =$$

$$-\begin{vmatrix} -\lambda & 0 & 0 & 1 & 0 \\ -\frac{2l_z}{r^3} & -\lambda & 0 & 0 & \frac{1}{r^2} \\ -\frac{3l_z^2}{r^4} - \frac{\partial^2 V}{\partial r^2} - \lambda^2 & -\frac{\partial^2 V}{\partial r \partial \theta} & -\frac{\partial^2 V}{\partial r \partial z} & 0 & \frac{2l_z}{r^3} \\ -\frac{\partial^2 V}{\partial \theta \partial r} & -\frac{\partial^2 V}{\partial \theta^2} & -\frac{\partial^2 V}{\partial \theta \partial z} & 0 & -\lambda \\ -\frac{\partial^2 V}{\partial z \partial r} & -\frac{\partial^2 V}{\partial z \partial \theta} & -\frac{\partial^2 V}{\partial z^2} - \lambda^2 & 0 & 0 \end{vmatrix} =$$

$$\begin{vmatrix} -\frac{2l_z}{r^3} & -\lambda & 0 & \frac{1}{r^2} \\ -\frac{3l_z^2}{r^4} - \frac{\partial^2 V}{\partial r^2} - \lambda^2 & -\frac{\partial^2 V}{\partial r \partial \theta} & -\frac{\partial^2 V}{\partial r \partial z} & \frac{2l_z}{r^3} \\ -\frac{\partial^2 V}{\partial \theta \partial r} & -\frac{\partial^2 V}{\partial \theta^2} & -\frac{\partial^2 V}{\partial \theta \partial z} & -\lambda \\ -\frac{\partial^2 V}{\partial z \partial r} & -\frac{\partial^2 V}{\partial z \partial \theta} & -\frac{\partial^2 V}{\partial z^2} - \lambda^2 & 0 \end{vmatrix} =$$

$$\begin{vmatrix} -\frac{2l_z}{r^3} & -\lambda & 0 & \frac{1}{r^2} \\ \frac{l_z^2}{r^4} - \frac{\partial^2 V}{\partial r^2} - \lambda^2 & -\frac{\partial^2 V}{\partial r \partial \theta} + \frac{2l_z}{r}\lambda & -\frac{\partial^2 V}{\partial r \partial z} & 0 \\ -\frac{\partial^2 V}{\partial \theta \partial r} & -\frac{\partial^2 V}{\partial \theta^2} & -\frac{\partial^2 V}{\partial \theta \partial z} & -\lambda \\ -\frac{\partial^2 V}{\partial z \partial r} & -\frac{\partial^2 V}{\partial z \partial \theta} & -\frac{\partial^2 V}{\partial z^2} - \lambda^2 & 0 \end{vmatrix} =$$

$$\begin{vmatrix} -\frac{2l_z}{r^3} & -\lambda & 0 & \frac{1}{r^2} \\ \frac{l_z^2}{r^4} - \frac{\partial^2 V}{\partial r^2} - \lambda^2 & -\frac{\partial^2 V}{\partial r \partial \theta} + \frac{2l_z}{r}\lambda & -\frac{\partial^2 V}{\partial r \partial z} & 0 \\ -\frac{\partial^2 V}{\partial \theta \partial r} - \frac{2l_z}{r}\lambda & -\frac{\partial^2 V}{\partial \theta^2} - \lambda^2 r^2 & -\frac{\partial^2 V}{\partial \theta \partial z} & 0 \\ -\frac{\partial^2 V}{\partial z \partial r} & -\frac{\partial^2 V}{\partial z \partial \theta} & -\frac{\partial^2 V}{\partial z^2} - \lambda^2 & 0 \end{vmatrix} =$$

$$-\frac{1}{r^2}\begin{vmatrix} \frac{l_z^2}{r^4} - \frac{\partial^2 V}{\partial r^2} - \lambda^2 & -\frac{\partial^2 V}{\partial r \partial \theta} + \frac{2l_z}{r}\lambda & -\frac{\partial^2 V}{\partial r \partial z} \\ -\frac{\partial^2 V}{\partial \theta \partial r} - \frac{2l_z}{r}\lambda & -\frac{\partial^2 V}{\partial \theta^2} - \lambda^2 r^2 & -\frac{\partial^2 V}{\partial \theta \partial z} \\ -\frac{\partial^2 V}{\partial z \partial r} & -\frac{\partial^2 V}{\partial z \partial \theta} & -\frac{\partial^2 V}{\partial z^2} - \lambda^2 \end{vmatrix} =$$

$$\begin{vmatrix} -\frac{l_z^2}{r^4} + \frac{\partial^2 V}{\partial r^2} + \lambda^2 & \frac{\partial^2 V}{\partial r \partial \theta} - \frac{2l_z}{r}\lambda & \frac{\partial^2 V}{\partial r \partial z} \\ \frac{\partial^2 V}{\partial \theta \partial r} + \frac{2l_z}{r}\lambda & \frac{\partial^2 V}{\partial \theta^2} + \lambda^2 r^2 & \frac{\partial^2 V}{\partial \theta \partial z} \\ \frac{\partial^2 V}{\partial z \partial r} & \frac{\partial^2 V}{\partial z \partial \theta} & \frac{\partial^2 V}{\partial z^2} + \lambda^2 \end{vmatrix} =$$

$$(-\frac{l_z^2}{r^4} + \frac{\partial^2 V}{\partial r^2} + \lambda^2)\begin{vmatrix} \frac{\partial^2 V}{\partial \theta^2} + \lambda^2 r^2 & \frac{\partial^2 V}{\partial \theta \partial z} \\ \frac{\partial^2 V}{\partial z \partial \theta} & \frac{\partial^2 V}{\partial z^2} + \lambda^2 \end{vmatrix} - (\frac{\partial^2 V}{\partial r \partial \theta} - \frac{2l_z}{r}\lambda)\begin{vmatrix} \frac{\partial^2 V}{\partial \theta \partial r} + \frac{2l_z}{r}\lambda & \frac{\partial^2 V}{\partial \theta \partial z} \\ \frac{\partial^2 V}{\partial z \partial r} & \frac{\partial^2 V}{\partial z^2} + \lambda^2 \end{vmatrix} +$$

$$(\frac{\partial^2 V}{\partial r \partial z})\begin{vmatrix} \frac{\partial^2 V}{\partial \theta \partial r} + \frac{2l_z}{r}\lambda & \frac{\partial^2 V}{\partial \theta^2} + \lambda^2 r^2 \\ \frac{\partial^2 V}{\partial z \partial r} & \frac{\partial^2 V}{\partial z \partial \theta} \end{vmatrix} =$$

$$(-\frac{l_z^2}{r^4} + \frac{\partial^2 V}{\partial r^2} + \lambda^2)((\frac{\partial^2 V}{\partial \theta^2} + \lambda^2 r^2)(\frac{\partial^2 V}{\partial z^2} + \lambda^2) - (\frac{\partial^2 V}{\partial \theta \partial z})^2) +$$

$$(-\frac{\partial^2 V}{\partial r \partial \theta} + \frac{2l_z}{r}\lambda)((\frac{\partial^2 V}{\partial \theta \partial r} + \frac{2l_z}{r}\lambda)(\frac{\partial^2 V}{\partial z^2} + \lambda^2) - \frac{\partial^2 V}{\partial \theta \partial z}\frac{\partial^2 V}{\partial z \partial r}) +$$

$$(\frac{\partial^2 V}{\partial r \partial z})((\frac{\partial^2 V}{\partial \theta \partial r} + \frac{2l_z}{r}\lambda)(\frac{\partial^2 V}{\partial z \partial \theta}) - (\frac{\partial^2 V}{\partial \theta^2} + \lambda^2 r^2)(\frac{\partial^2 V}{\partial z \partial r})) =$$

$$\begin{gathered}
(-\frac{l_z^2}{r^4} + \frac{\partial^2 V}{\partial r^2} + \lambda^2)(\frac{\partial^2 V}{\partial\theta^2}\frac{\partial^2 V}{\partial z^2} + \lambda^2 r^2 \frac{\partial^2 V}{\partial z^2} + \lambda^2 \frac{\partial^2 V}{\partial\theta^2} + \lambda^4 r^2 - (\frac{\partial^2 V}{\partial\theta\partial z})^2) + \\
(\frac{4l_z^2}{r^2}\lambda^2 - (\frac{\partial^2 V}{\partial\theta\partial r})^2)(\frac{\partial^2 V}{\partial z^2} + \lambda^2) + \frac{\partial^2 V}{\partial r\partial\theta}\frac{\partial^2 V}{\partial\theta\partial z}\frac{\partial^2 V}{\partial z\partial r} - \frac{2l_z}{r}\lambda\frac{\partial^2 V}{\partial\theta\partial z}\frac{\partial^2 V}{\partial z\partial r} + \\
(\frac{\partial^2 V}{\partial r\partial z})((\frac{\partial^2 V}{\partial\theta\partial r} + \frac{2l_z}{r}\lambda)(\frac{\partial^2 V}{\partial z\partial\theta}) - (\frac{\partial^2 V}{\partial\theta^2} + \lambda^2 r^2)(\frac{\partial^2 V}{\partial z\partial r})) =
\end{gathered}$$

$$\begin{gathered}
(-\frac{l_z^2}{r^4} + \frac{\partial^2 V}{\partial r^2} + \lambda^2)(\frac{\partial^2 V}{\partial\theta^2}\frac{\partial^2 V}{\partial z^2} + \lambda^2 r^2 \frac{\partial^2 V}{\partial z^2} + \lambda^2 \frac{\partial^2 V}{\partial\theta^2} + \lambda^4 r^2 - (\frac{\partial^2 V}{\partial\theta\partial z})^2) + \\
((\frac{4l_z^2}{r^2}\lambda^2 - (\frac{\partial^2 V}{\partial\theta\partial r})^2)(\frac{\partial^2 V}{\partial z^2} + \lambda^2) + \frac{\partial^2 V}{\partial r\partial\theta}\frac{\partial^2 V}{\partial\theta\partial z}\frac{\partial^2 V}{\partial z\partial r} - \frac{2l_z}{r}\lambda\frac{\partial^2 V}{\partial\theta\partial z}\frac{\partial^2 V}{\partial z\partial r} + \\
\frac{\partial^2 V}{\partial r\partial z}\frac{\partial^2 V}{\partial z\partial\theta}\frac{\partial^2 V}{\partial\theta\partial r} + \frac{2l_z}{r}\lambda\frac{\partial^2 V}{\partial r\partial z}\frac{\partial^2 V}{\partial z\partial\theta} - (\frac{\partial^2 V}{\partial\theta^2} + \lambda^2 r^2)(\frac{\partial^2 V}{\partial z\partial r})^2 =
\end{gathered}$$

$$\begin{gathered}
(-\frac{l_z^2}{r^4} + \frac{\partial^2 V}{\partial r^2} + \lambda^2)(\frac{\partial^2 V}{\partial\theta^2}\frac{\partial^2 V}{\partial z^2} + \lambda^2 r^2 \frac{\partial^2 V}{\partial z^2} + \lambda^2 \frac{\partial^2 V}{\partial\theta^2} + \lambda^4 r^2 - (\frac{\partial^2 V}{\partial\theta\partial z})^2) + \\
(\frac{4l_z^2}{r^2}\lambda^2 - (\frac{\partial^2 V}{\partial\theta\partial r})^2)(\frac{\partial^2 V}{\partial z^2} + \lambda^2) + 2\frac{\partial^2 V}{\partial r\partial z}\frac{\partial^2 V}{\partial z\partial\theta}\frac{\partial^2 V}{\partial\theta\partial r} + \\
-\frac{\partial^2 V}{\partial\theta^2}(\frac{\partial^2 V}{\partial z\partial r})^2 - \lambda^2 r^2(\frac{\partial^2 V}{\partial z\partial r})^2 =
\end{gathered}$$

$$\begin{gathered}
(\frac{\partial^2 V}{\partial r^2} + \lambda^2)(\frac{\partial^2 V}{\partial\theta^2}\frac{\partial^2 V}{\partial z^2} + \lambda^2 r^2 \frac{\partial^2 V}{\partial z^2} + \lambda^2 \frac{\partial^2 V}{\partial\theta^2} + \lambda^4 r^2 - (\frac{\partial^2 V}{\partial\theta\partial z})^2) + \\
-\frac{l_z^2}{r^4}\frac{\partial^2 V}{\partial\theta^2}\frac{\partial^2 V}{\partial z^2} - \frac{l_z^2}{r^2}\lambda^2\frac{\partial^2 V}{\partial z^2} - \frac{l_z^2}{r^4}\lambda^2\frac{\partial^2 V}{\partial\theta^2} - \frac{l_z^2}{r^2}\lambda^4 + \frac{l_z^2}{r^4}(\frac{\partial^2 V}{\partial\theta\partial z})^2 + \\
-(\frac{\partial^2 V}{\partial\theta\partial r})^2\lambda^2 + \frac{4l_z^2}{r^2}\lambda^4 + \frac{4l_z^2}{r^2}\lambda^2\frac{\partial^2 V}{\partial z^2} - (\frac{\partial^2 V}{\partial\theta\partial r})^2\frac{\partial^2 V}{\partial z^2} + 2\frac{\partial^2 V}{\partial r\partial z}\frac{\partial^2 V}{\partial z\partial\theta}\frac{\partial^2 V}{\partial\theta\partial r} + \\
-\frac{\partial^2 V}{\partial\theta^2}(\frac{\partial^2 V}{\partial z\partial r})^2 - \lambda^2 r^2(\frac{\partial^2 V}{\partial z\partial r})^2 =
\end{gathered}$$

$$\begin{gathered}
(\frac{\partial^2 V}{\partial r^2} + \lambda^2)(\frac{\partial^2 V}{\partial\theta^2}\frac{\partial^2 V}{\partial z^2} + \lambda^2 r^2 \frac{\partial^2 V}{\partial z^2} + \lambda^2 \frac{\partial^2 V}{\partial\theta^2} + \lambda^4 r^2 - (\frac{\partial^2 V}{\partial\theta\partial z})^2) + \\
-\frac{l_z^2}{r^4}\frac{\partial^2 V}{\partial\theta^2}\frac{\partial^2 V}{\partial z^2} + \frac{3l_z^2}{r^2}\lambda^2\frac{\partial^2 V}{\partial z^2} - \frac{l_z^2}{r^4}\lambda^2\frac{\partial^2 V}{\partial\theta^2} + \frac{3l_z^2}{r^2}\lambda^4 + \frac{l_z^2}{r^4}(\frac{\partial^2 V}{\partial\theta\partial z})^2 + \\
-(\frac{\partial^2 V}{\partial\theta\partial r})^2\lambda^2 - (\frac{\partial^2 V}{\partial\theta\partial r})^2\frac{\partial^2 V}{\partial z^2} + 2\frac{\partial^2 V}{\partial r\partial z}\frac{\partial^2 V}{\partial z\partial\theta}\frac{\partial^2 V}{\partial\theta\partial r} + \\
-\frac{\partial^2 V}{\partial\theta^2}(\frac{\partial^2 V}{\partial z\partial r})^2 - \lambda^2 r^2(\frac{\partial^2 V}{\partial z\partial r})^2 =
\end{gathered}$$

$$\begin{gathered}
\frac{\partial^2 V}{\partial\theta^2}\frac{\partial^2 V}{\partial z^2}\lambda^2 + \lambda^4 r^2\frac{\partial^2 V}{\partial z^2} + \lambda^4\frac{\partial^2 V}{\partial\theta^2} + \lambda^6 r^2 - (\frac{\partial^2 V}{\partial\theta\partial z})^2\lambda^2 + \\
\frac{\partial^2 V}{\partial r^2}\frac{\partial^2 V}{\partial\theta^2}\frac{\partial^2 V}{\partial z^2} + \lambda^2 r^2\frac{\partial^2 V}{\partial r^2}\frac{\partial^2 V}{\partial z^2} + \lambda^2\frac{\partial^2 V}{\partial r^2}\frac{\partial^2 V}{\partial\theta^2} + \lambda^4 r^2\frac{\partial^2 V}{\partial r^2} - \frac{\partial^2 V}{\partial r^2}(\frac{\partial^2 V}{\partial\theta\partial z})^2 \\
-\frac{l_z^2}{r^4}\frac{\partial^2 V}{\partial\theta^2}\frac{\partial^2 V}{\partial z^2} + \frac{3l_z^2}{r^2}\lambda^2\frac{\partial^2 V}{\partial z^2} - \frac{l_z^2}{r^4}\lambda^2\frac{\partial^2 V}{\partial\theta^2} + \frac{3l_z^2}{r^2}\lambda^4 + \frac{l_z^2}{r^4}(\frac{\partial^2 V}{\partial\theta\partial z})^2 + \\
-(\frac{\partial^2 V}{\partial\theta\partial r})^2\lambda^2 - (\frac{\partial^2 V}{\partial\theta\partial r})^2\frac{\partial^2 V}{\partial z^2} + 2\frac{\partial^2 V}{\partial r\partial z}\frac{\partial^2 V}{\partial z\partial\theta}\frac{\partial^2 V}{\partial\theta\partial r} + \\
-\frac{\partial^2 V}{\partial\theta^2}(\frac{\partial^2 V}{\partial z\partial r})^2 - \lambda^2 r^2(\frac{\partial^2 V}{\partial z\partial r})^2 =
\end{gathered}$$

$$r^2\lambda^6 + (r^2\frac{\partial^2 V}{\partial z^2} + \frac{\partial^2 V}{\partial \theta^2} + r^2\frac{\partial^2 V}{\partial r^2} + \frac{3l_z^2}{r^2})\lambda^4$$
$$(-(\frac{\partial^2 V}{\partial\theta\partial z})^2 +$$
$$\frac{\partial^2 V}{\partial \theta^2}\frac{\partial^2 V}{\partial z^2} + r^2\frac{\partial^2 V}{\partial r^2}\frac{\partial^2 V}{\partial z^2} + \frac{\partial^2 V}{\partial r^2}\frac{\partial^2 V}{\partial \theta^2} - r^2(\frac{\partial^2 V}{\partial z\partial r})^2 + \frac{3l_z^2}{r^2}\frac{\partial^2 V}{\partial z^2} - (\frac{\partial^2 V}{\partial\theta\partial r})^2 - \frac{l_z^2}{r^4}\frac{\partial^2 V}{\partial \theta^2})\lambda^2$$
$$-\frac{\partial^2 V}{\partial r^2}(\frac{\partial^2 V}{\partial\theta\partial z})^2 +$$
$$-(\frac{\partial^2 V}{\partial\theta\partial r})^2\frac{\partial^2 V}{\partial z^2} + 2\frac{\partial^2 V}{\partial r\partial z}\frac{\partial^2 V}{\partial z\partial\theta}\frac{\partial^2 V}{\partial\theta\partial r} + \frac{\partial^2 V}{\partial r^2}\frac{\partial^2 V}{\partial \theta^2}\frac{\partial^2 V}{\partial z^2} + -\frac{\partial^2 V}{\partial \theta^2}(\frac{\partial^2 V}{\partial z\partial r})^2 - \frac{l_z^2}{r^4}\frac{\partial^2 V}{\partial \theta^2}\frac{\partial^2 V}{\partial z^2} + \frac{l_z^2}{r^4}(\frac{\partial^2 V}{\partial\theta\partial z})^2$$

This is a cubic equation in λ^2 which can be solved algebraically using Cardano's formula. After taking the square roots, this yields for the current orbit of Jupiter:

$$\lambda_1 = -4.13590306 \cdot 10^{-25} + 1.67189035 \cdot 10^{-8}i, \lambda_2 = -4.13590306 \cdot 10^{-25} - 1.67189035 \cdot 10^{-8}i,$$
$$\lambda_3 = -5.88073717 \cdot 10^{-25} + 9.49530837 \cdot 10^{-10}i, \lambda_4 = -5.88073717 \cdot 10^{-25} - 9.49530837 \cdot 10^{-10}i,$$
$$\lambda_5 = -1.53228342 \cdot 10^{-26} + 1.67249205 \cdot 10^{-8}i, \lambda_6 = -1.53228342 \cdot 10^{-26} - 1.67249205 \cdot 10^{-8}i \quad \text{(B.3)}$$

The real parts are very small and due to numerical errors. In fact, we know they should actually be zero, since the Trojans orbits do not converge towards the Lagrange point. From the eigenvalues we obtain a time step of

$$dt \leq \frac{2.5}{|\lambda|_{\text{max}}} = \frac{2.5}{1.6724920458162246 \cdot 10^{-8}} \approx 149477542 \text{ s} \approx 4.74 \text{ y} \quad \text{(B.4)}$$

which is less than a half times the orbital cycle time of Jupiter. One could, however, expect a smaller time step further away from the equilibrium point. Using the analysis above and taking into consideration that Ji et al. used a time step of a half a year[2], a time step less than half a year seems a reasonable estimate. One could do an error estimation to determine whether the used time step is indeed small enough for a specific Trojan.

C
Alternative Trojan Conversion

As described in 4, it is also possible to convert the Trojans by making the rough assumption Jupiter has no influence on a Trojans orbit during small periods of time, allowing all Trojans to start from the same moment. The 9 steps described in 4 then become the following instead: 1. Assume Jupiter's influence on the Trojan is a slow and small effect. Thus the Kepler elements from NASA represent constant Kepler orbits around the Sun ($m_J = 0$).
2. Calculate the distance to the sun and the velocity of the Trojan at the time of the data in cartesian coordinates.
3. Use these quantities and the anomaly to create vectors for the position and velocity in the x,y (polar) plane.
4. Use a rotation matrix to rotate the vectors to the plane of the orbit.
5. Calculate the last moment in time before J2000 that Jupiter has 0 anomaly
6. Use numerical integration to calculate the position and velocity vectors of the Trojan at the same moment in time.
7. Rotate the Trojan such that Jupiter moves in the polar plane.
8. Assume Jupiter now is present and correct for the shifted centre of mass.
9. Convert to cylindrical coordinates in the rotating frame.

D Additional results

Below are all the result tables of the found simulations that are not considered in the main text. Some tables are not discussed in the main text, since the simulation times were to short to make proper conclusion. However, they were used to obtain some first insight at early attempts on what short of simulations one could run.

D.1. Tables

	L4'	L5'	Horse shoe'	Unstable'	Total
L4	190	0	0	6	196
L5	0	196	0	0	196
Horse shoe	0	0	0	0	0
Unstable	0	0	8	8	8
Total'	190	196	0	38	400

Table D.1: Trojans of each type after a migration of 2 AU of Jupiter from its current position over a time of 100 ky, $\tau = 1000$ y. The primes stand for the new types after migration, while the types without prime stand for the type without migration. The first 100 Trojans from [14] and its four symmetric copies were used. Jupiter was assumed to be in a perfect circular orbit.

	L4'	L5'	Horse shoe'	Unstable'	Total
L4	75	0	0	23	98
L5	0	98	0	0	98
Horse shoe	0	0	0	0	0
Unstable	0	0	0	4	4
Total'	75	98	0	27	200

Table D.2: Trojans of each type after a migration of 2 AU of Jupiter from its current position over a time of one million years with $\tau = 1000$ y. The primes stand for the new types after migration, while the types without prime stand for the type without migration. The first 100 Trojans from [14] and its primal symmetric copies were used. Jupiter was assumed to have its current eccentricity of 0.04839266 during the entire simulation.

	L4'	L5'	Horse shoe'	Unstable'	Total
L4	1	16	0	81	98
L5	1	52	0	45	98
Horse shoe	0	0	0	0	0
Unstable	0	0	0	4	4
Total'	2	68	0	130	200

Table D.3: Trojans of each type after a migration of 2 AU of Jupiter from its current position over a time of one million years with $\tau = 300$ y. The primes stand for the new types after migration, while the types without prime stand for the type without migration. The first 1000 Trojans from [14] and its primal symmetric copies were used. Jupiter was assumed to have its current eccentricity of 0.04839266 during the entire simulation.

	L4'	L5'	Horse shoe'	Unstable'	Total
L4	888	0	0	102	1000
L5	1	952	1	38	991
Horse shoe	0	0	0	0	4
Unstable	0	0	0	15	15
Total'	889	952	1	155	2000

Table D.4: Trojans of each type after a migration of 1 AU of Jupiter from its current position over a time of one million years with $\tau = 300$ y. The primes stand for the new types after migration, while the types without prime stand for the type without migration. The first 1000 Trojans from [14] and its primal symmetric copies were used. Jupiter was assumed to have three times its current eccentricity of 0.04839266 during the entire simulation.

	L4'	L5'	Horse shoe'	Unstable'	Total
L4	806	0	0	184	1000
L5	1	986	1	5	992
Horse shoe	0	2	0	2	4
Unstable	0	3	0	12	15
Total'	806	991	1	155	2000

Table D.5: Trojans of each type after a migration of 2 AU of Jupiter from its current position over a time of one million years with $\tau = 300$ y. The primes stand for the new types after migration, while the types without prime stand for the type without migration. The first 1000 Trojans from [14] and its primal symmetric copies were used. Jupiter was assumed to have its current eccentricity of 0.04839266 during the entire simulation.

	L4'	L5'	Horse shoe'	Unstable'	Total
L4	490	0	0	2	492
L5	0	486	0	0	486
Horse shoe	0	4	0	2	6
Unstable	0	0	0	16	16
Total'	490	490	0	20	10000

Table D.6: Trojans of each type after a migration of 0.5 AU of Jupiter from its current position over a time of 100k y, tau=500y

	L4'	L5'	Horse shoe'	Unstable'	Total
L4	466	0	0	30	492
L5	0	496	0	0	486
Horse shoe	0	0	0	2	6
Unstable	0	0	8	8	16
Total'	466	496	0	38	1000

Table D.7: Trojans of each type after a migration of 2 AU of Jupiter from its current position over a time of 100.000 year. The migration is described by equation 4.1, with τ=500 y. The primes stand for the new types after migration, while the types without prime stand for the type without migration. The initial conditions of the first 250 Trojans are from [14], after which we used all the symmetries to create 1000 Trojans. The 250 Trojans could not be considered fully random, but this result can be used for a first insight in what might happen.

	L4’	L5’	Horse shoe’	Unstable’	Total
L4	494	0	0	2	496
L5	0	496	0	0	496
Horse shoe	0	0	0	0	0
Unstable	0	0	0	0	8
Total’	494	496	0	2	1000

Table D.8: Trojans of each type after a migration of 0.5 AU of Jupiter from its current position over a time of 100 ky, $\tau = 500$ y. The primes stand for the new types after migration, while the types without prime stand for the type without migration.

	L4’	L5’	Horse shoe’	Unstable’	Total
L4	1945	0	7	417	1952
L5	0	1968	4	397	1972
Horse shoe	45	14	7	1361	1427
Unstable	5	2	1	42932	42939
Total’	1995	1984	19	45107	50220

Table D.9: Trojans of each type after a migration of 1 AU of Jupiter from its current position over a time of 100 ky, $\tau = 1000$ y. The primes stand for the new types after migration, while the types without prime stand for the type without migration. There is a 2000 years pre run before the migration started.

	L4’	L5’	Horse shoe’	Unstable’	Total
L4	14021	0	13	2513	16547
L5	0	14509	16	2009	16534
Horse shoe	945	986	6	1354	3291
Unstable	2080	2525	27	9001	13633
Total’	17046	18020	62	14877	50220

Table D.10: Trojans of each type after a migration of 1 AU of Jupiter from its current position over a time of 100 ky, $\tau = 1000$ y. The primes stand for the new types after migration, while the types without prime stand for the type without migration. There is a 2000 years pre run before the migration started. The Trojans had no radial speed initially.

L4	L5	Horse shoe	Unstable	Total
24832	0	0	5	24837
0	24844	0	38	24882
0	6	0	42	48
0	4	0	394	398

Table D.11: Trojans of each type after a migration of 0.5 AU of Jupiter from its current position over a time of 33333 years using a time step of 1/3 years using the function 6.2.

	L4’	L5’	Horse shoe’	Unstable’	Total’
L4	24820	0	0	64	24884
L5	0	24836	0	46	24882
Horse shoe	0	6	0	44	50
Unstable	0	4	0	398	402

Table D.12: Trojans of each type after a migration of 0.5 AU of Jupiter from its current position over a time of 33333 years using a time step of 1/3 years.

	L4’	L5’	Horse shoe’	Unstable’	Total
L4	492	0	0	2	2
L5	0	488	0	6	6
Horse shoe	2	0	2	4	4
Unstable	0	0	0	4	4
Total’	494	488	2	16	16

Table D.13: Trojans of each type after a migration of 1 AU of Jupiter from its current position over a time of 100 ky, $\tau = 1000$ y. The primes stand for the new types after migration, while the types without prime stand for the type without migration. There is a 2000 years pre run before the migration started. The Trojans had no radial speed initially. The R function is now smooth

	L4’	L5’	Horse shoe’	Unstable’	Total
L4	24236	0	20	330	24586
L5	0	24248	2	322	24572
Horse shoe	26	16	14	336	4
Unstable	4	0	0	628	632
Total’	24266	24264	36	1616	50182

Table D.14: Trojans of each type after a migration of 1 AU of Jupiter from its current position over a time of 100 ky, $\tau = 500$ y. The primes stand for the new types after migration, while the types without prime stand for the type without migration. There is a 2000 years pre run before the migration started. The Trojans had no radial speed initially. The R functions is now smooth

	L4’	L5’	Horse shoe’	Unstable’	Total
L4	24188	0	20	166	24586
L5	0	24198	0	176	24572
Horse shoe	32	3	14	128	4
Unstable	46	32	2	115	632
Total’	24266	24264	36	1616	50182

Table D.15: Same as D.13, but now the normal simulation is run for 100000 years as well to match up the lengths of the simulations.

	L4’	L5’	Horse shoe’	Unstable’	Total
L4	24184	0	20	151	24586
L5	0	24194	0	162	24572
Horse shoe	30	28	14	101	4
Unstable	0	0	2	0	0
Total’	24266	24264	36	1616	50182

Table D.16: Same as D.14, but now originally unstable Trojans and its mirrors are removed

	L4’	L5’	Horse shoe’	Unstable’	Total
L4	2686	0	0	20	2686
L5	0	2674	0	38	2674
Horse shoe	0	8	14	0	22
Unstable	8	2	2	0	12
Total’	2694	2684	14	17	

Table D.17: Same as D.13, but now only Trojans with initial inclinations of 5 degrees are used.

	L4’	L5’	Horse shoe’	Unstable’	Total
L4	2684	0	0	22	2686
L5	0	2670	0	42	2674
Horse shoe	0	4	0	20	22
Unstable	8	0	0	0	172
Total’	2694	2684	14	17	

Table D.18: Same as D.17, but now the simulations are extended to a million years, resulting to a more strict result.

	L4’	L5’	Horse shoe’	Unstable’	Total
L4	1006	0	0	10	1016
L5	0	1000	0	14	1014
Horse shoe	0	0	0	6	6
Unstable	2	0	0	94	94
Total’	1008	1000	0	134	

Table D.19: Same as D.17, but now inclinations less than 3 degrees are considered.

	L4’	L5’	Horse shoe’	Unstable’	Total
L4	132	0	0	6	138
L5	0	130	0	2	132
Horse shoe	0	0	0	0	0
Unstable	0	0	0	24	24
Total’	132	130	0	32	

Table D.20: Same as D.17, but now inclinations less than 1 degrees are considered.

	L4	L5	Horse shoe	Unstable	Total’
L4’	24820	0	0	64	24884
L5’	0	24836	0	46	24882
Horse shoe’	0	6	0	44	50
Unstable’	0	4	0	398	402

Table D.21: Trojans of each type after a migration of 0.5 AU of Jupiter from its current position over a time of 33333 years using a time step of 1/3 years.

L4’	L4’	L5’	HS’	Unstable’	Total	L4’	L5’	HS’	Unstable’	Total
L4	975	0	0	4	979	972	0	0	7	979
L5	0	971	0	5	976	0	969	0	7	976
HS	0	0	0	8	8	0	0	0	8	8
Unstable	0	0	0	36	36	0	0	0	36	36
Total	975	971	0	43		972	969	0	48	

(b) One million years

(c) Five million years

Table D.22: Trojans of each type after a migration of 1 AU of Jupiter from its current position over two different simulation times using a time step of 1/3 years. The migration rate is described equation 6.2 with $\tau = 1000$ y. The initial Trojans are 1000 randomly selected actual Trojans from [14] with its symmetric copies to obtain as many L4 and L5 Trojans initially. The prime denotes the types after the migration and HS stands for Horse shoe.

D.2. Plots

Figures D.2, 6.6, D.7 and D.3 show some plots belonging to different migrations and are not discussed in the main text.

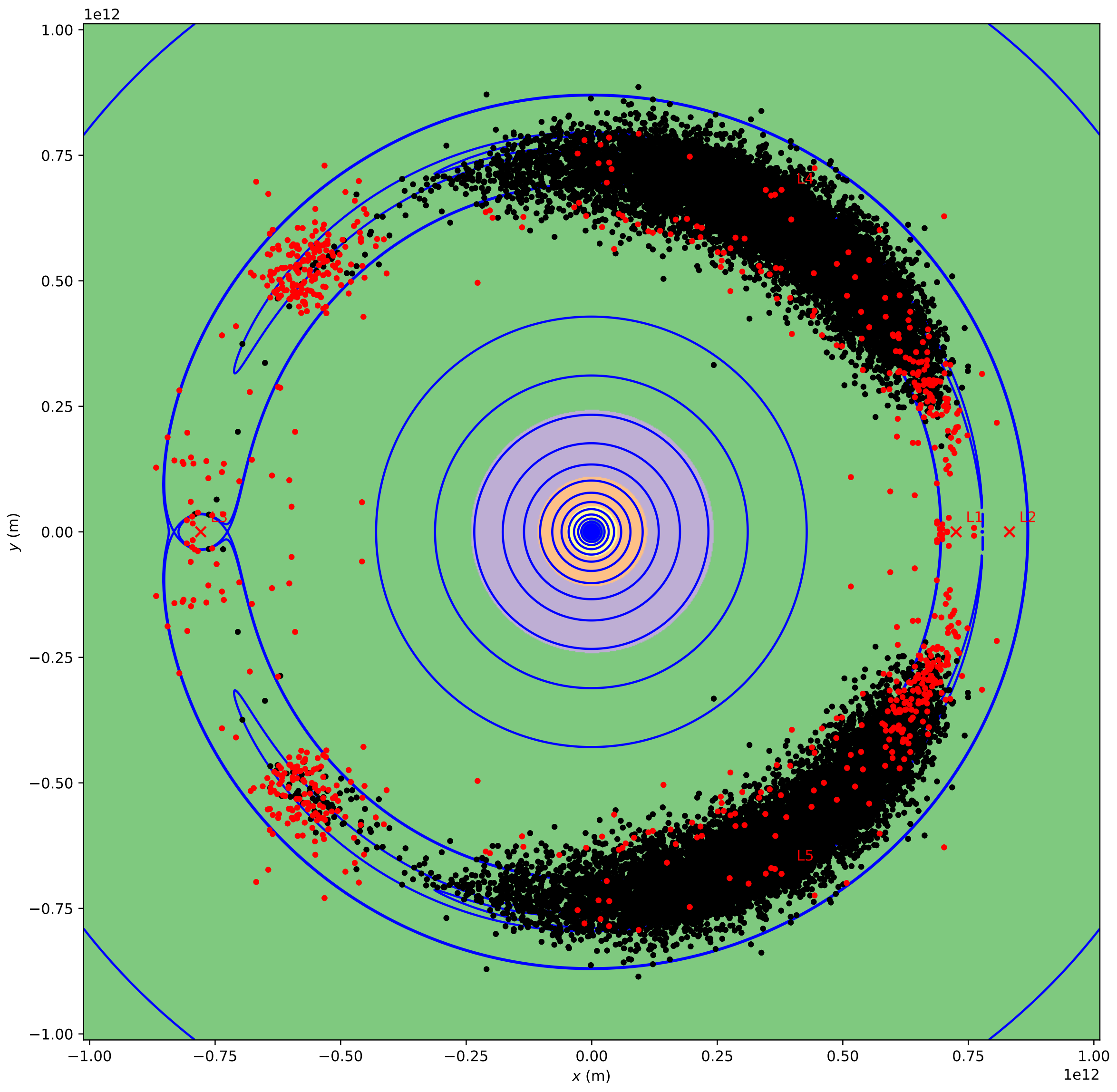
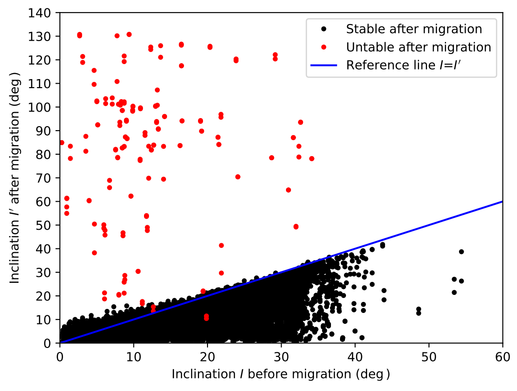


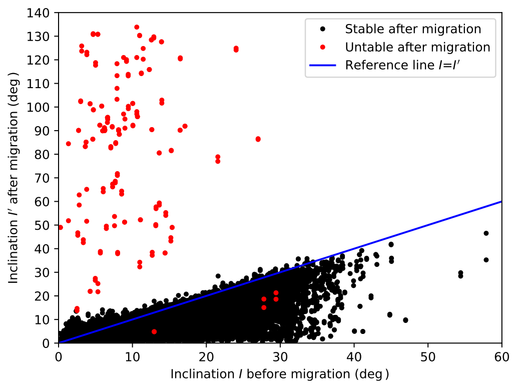
Figure D.1: Distribution of the initial data set. The Trojans that have become unstable after a migration of 1 AU over 100000 years using $\tau = 1000$ y are marked red. The Trojan data is downloaded from NASA[14]. It has different epochs, thus the distribution shown does not represent any current distribution, while each position is an actual existing position.

6.6. From this figure it appears that first two quantities and the last one are not relevant and therefore they will not be considered in other plots. In the third plot, it appears that L5 Trojans are 'better' at handling extreme semi-major axes than L4 Trojans, but this has to be investigated. The last plot we make is the plot where the minimum and maximum angles are plotted in absolute values in figure 6.7.

Figure D.4, D.5 and 2.7 show some more examples of Trojan orbits.

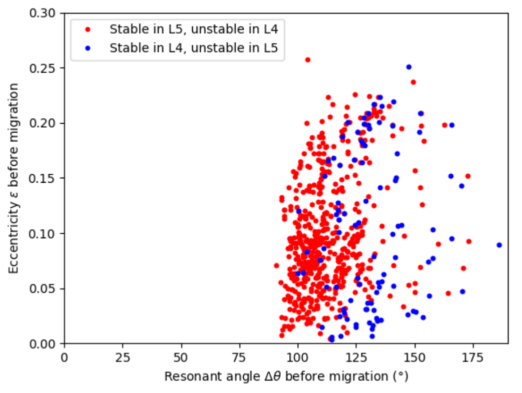


(a) L4

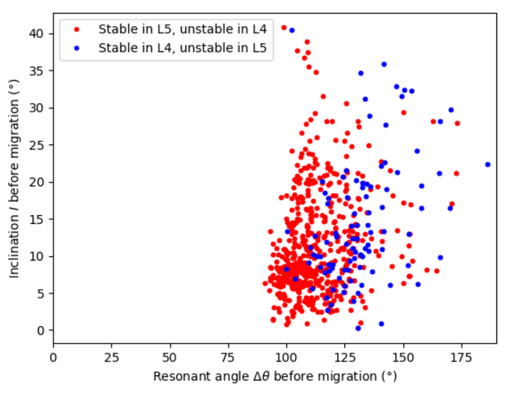


(b) L5

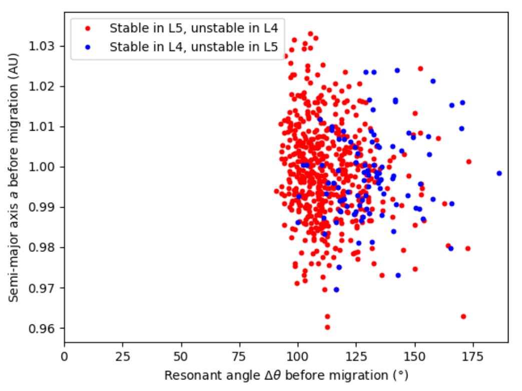
Figure D.2: The inclinations of Trojans before and after a migration of 1 AU over 100000 years of Jupiter using $\tau = 1000$ y. The red Trojans are unstable, but it have to be noted that the inclinations after are not well defined for unstable Trojans. From the figure one deduces that most Trojans obtain lower inclinations after the migration and that for L5 Trojans, higher inclined Trojans might have a higher probability to survive the migration.



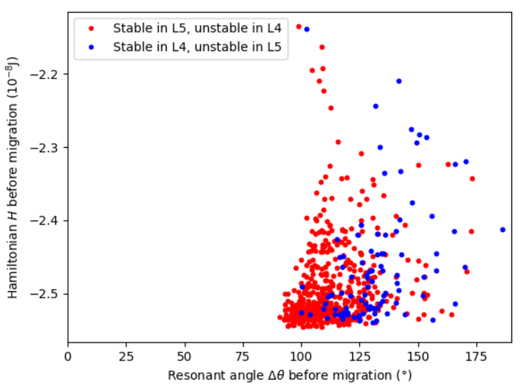
(a) Eccentricity



(b) Inclination



(c) Semi-major axis



(d) Hamiltonian

Figure D.3: Plots of the initial resonant angle $\Delta\theta$ of some Trojans versus several different other initial quantities. The stability is determined after a migration of Jupiter over 1 AU by the function from equation 6.2 with parameter $\tau = 1000$ years and the current Jupiter's orbital eccentricity is used. The total simulation time is one million years. The Trojans considered in both cases are the same and are real Trojans from the NASA database[14] or symmetric copies of those. For this plot. Trojans that led to unstable Trojans for both the original Trojan and a copy or to stable Trojans in both cases, are omitted to emphasize the different between the original Trojans and the copies.

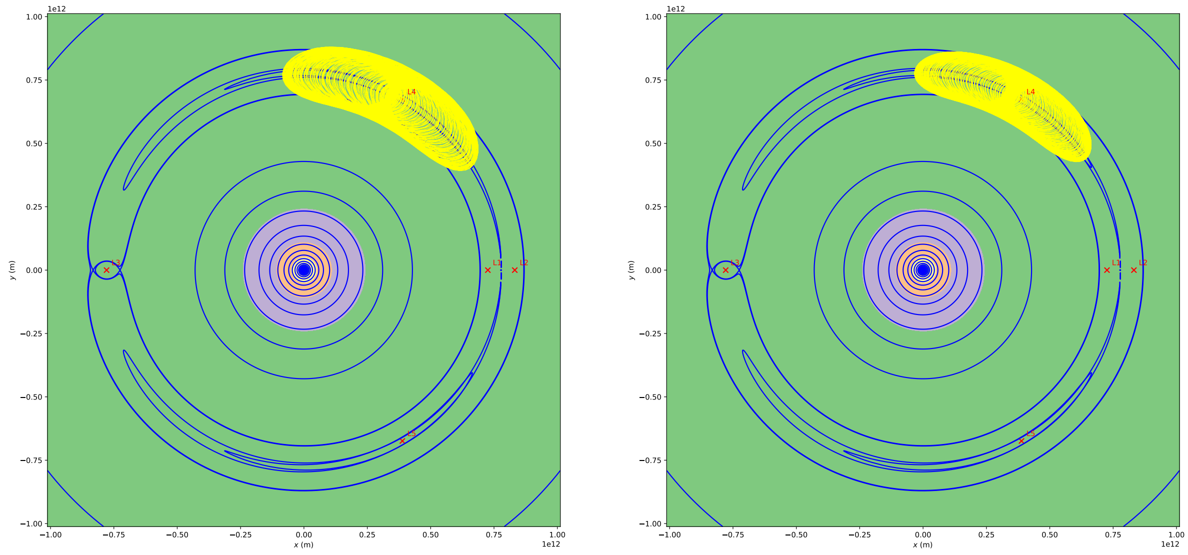


Figure D.4: Two examples of actual Trojans orbiting the L4 point in the horizontal plane together with the equipotential lines. The black dot represents Jupiter and the colored circles ranges of the potentials, where the scale is logarithmic. One notices that the curves are both banana shaped, but have different sizes and bendings.

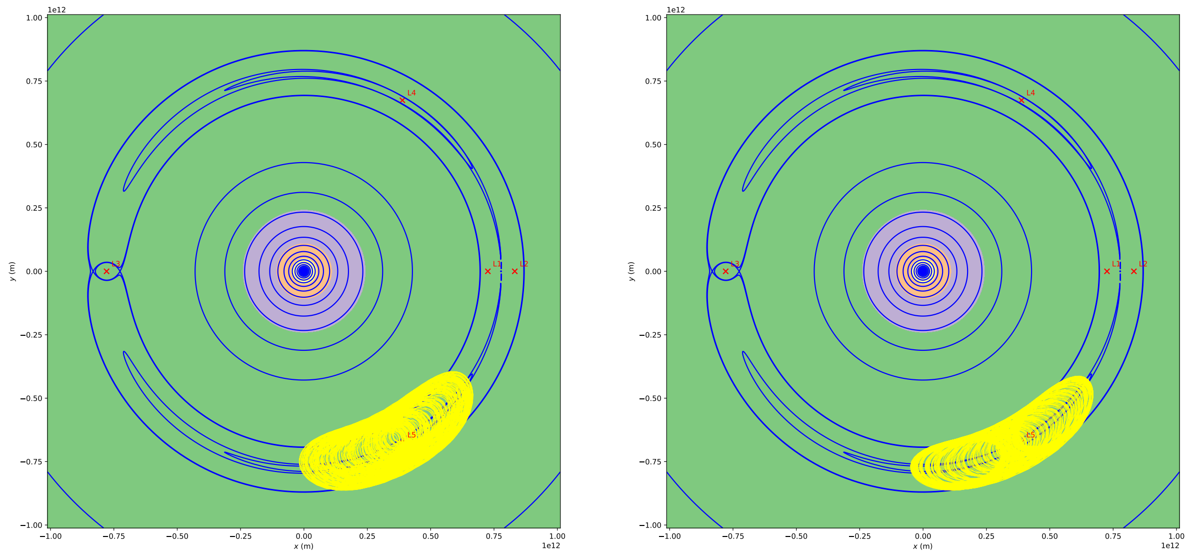


Figure D.5: Two examples of actual Trojans orbiting the L4 point in the horizontal plane together with the equipotential lines. The colored represent circles ranges of the potentials, where the scale is logarithmic.

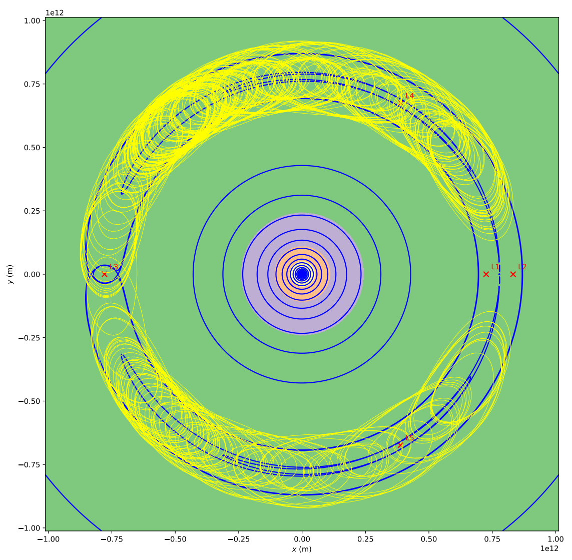


Figure D.6: Examples of an actual Trojan in a horse shoe orbit in the horizontal plane together with the equipotential lines. The colored circles represent ranges of the potentials, where the scale is logarithmic.

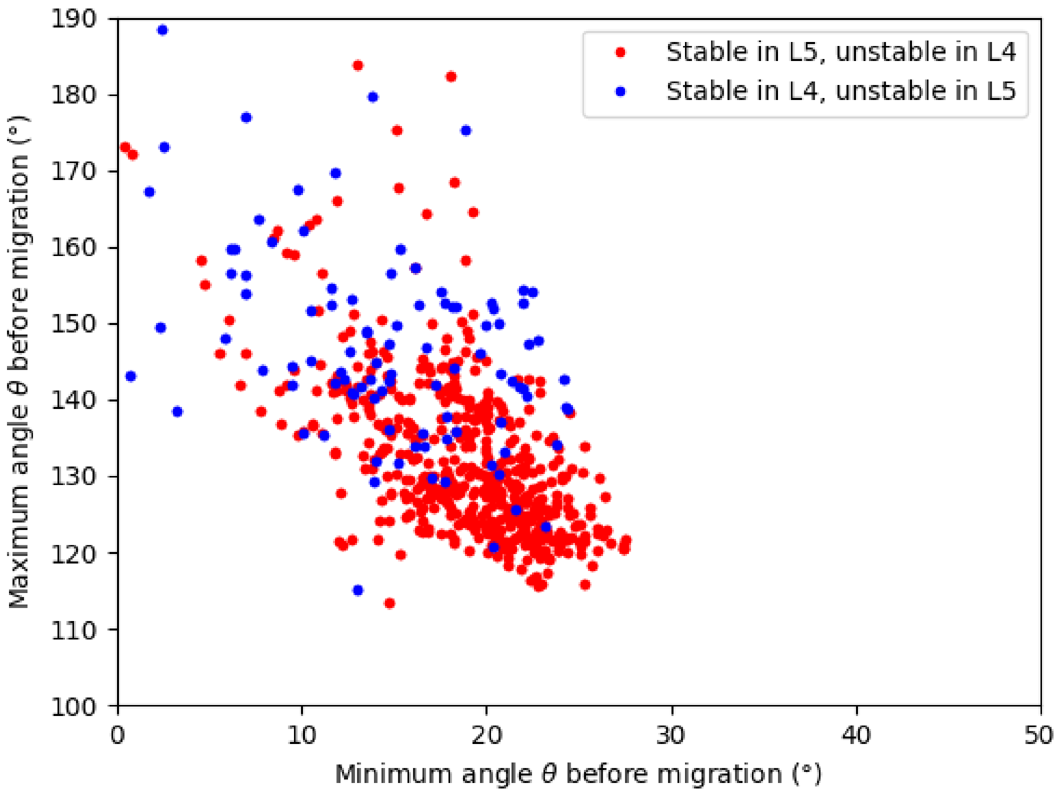


Figure D.7: Plots of the initial minimum and maximum angle θ of some Trojans relative to Jupiter. The stability is determined after a migration of Jupiter over 1 AU prescribed by the function from equation 6.2 with parameter $\tau = 1000$ years and the current Jupiter's orbital eccentricity is used. The total simulation time is one million years. The Trojans considered in both cases are the same and are real Trojans from the NASA database[14] or symmetric copies of those. For this plot. Trojans that led to unstable Trojans for both the original Trojan and a copy or to stable Trojans in both cases, are omitted to emphasize the different between the original Trojans and the copies. The resonant angle has a different definition than in the main text and is defined by the difference between the maximum and minimum angle.

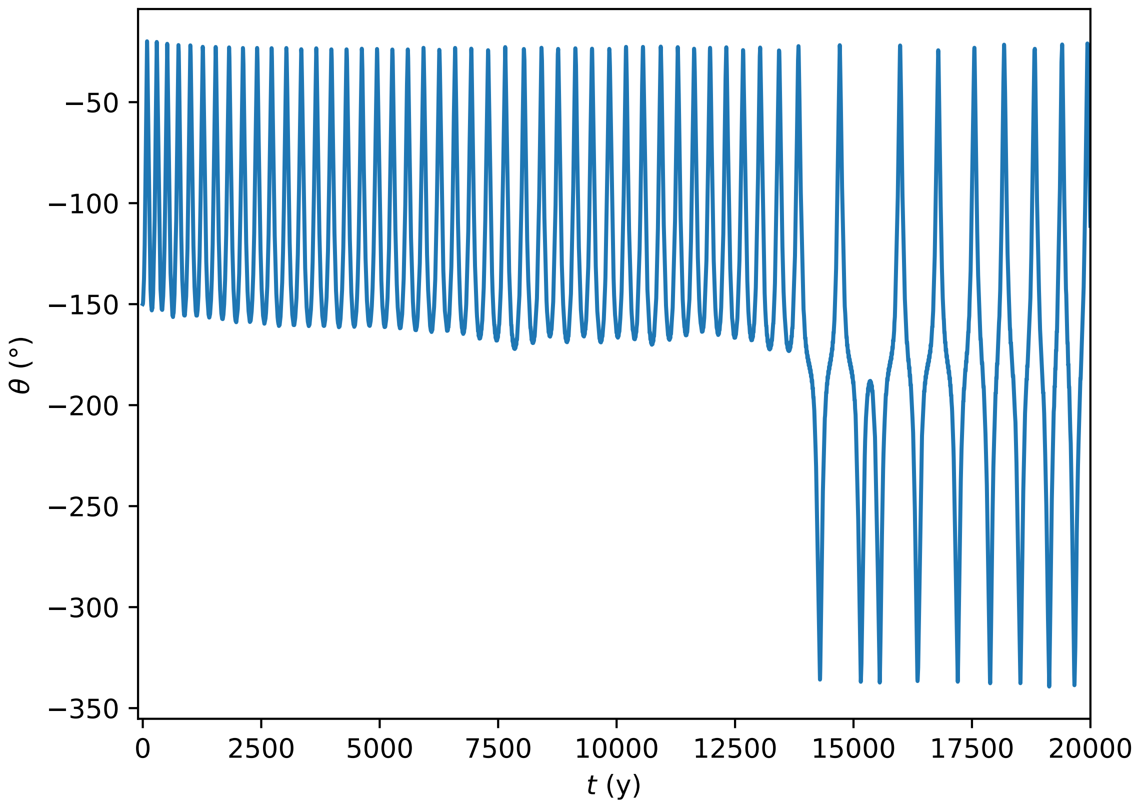


Figure D.8: Graph 5.2 extended for the L5 Trojan. Here we see that the Trojan obtains a horse shoe orbit before it becomes unstable.

E
Code

Below, all used code is printed. The two main codes, which are the C code, which executes the actual simulations, and the Python_function code, which is a library to easily handle the C code and create some quick plots to obtain some insight. This allows fast simulation, but also humanly readable code and easy data handling using Numpy. There is also an example code provided of some data handling. Furthermore,the code used to create some plots are provided.

E.1. C code

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>


#define G       (6.6384e-11)                                    // Gravitational constant
#define m_S     (1.98847e30)                            // Mass sun (kg)
#define m_J     (1.89813e27)                            // Mass Jupiter (kg)
#define mu      (m_J/m_S)
#define AU      (1.496e11)                                      // Astronomical unit (m)
#define year    (3600*24*365)                                   // convert year to seconds
//#define m       (m_J + m_S)                                   // Combined mass(kg)
#define Gm_S     (6.6384e-11*m_S)
#define Gm_J     (6.6384e-11*m_J)

// Constants for the Yoshida integrator
#define d1 1/(4-cbrt(16))                               // d1/2 = d3/2 (the numbers are
    halved to save computation time later)
#define d2 -cbrt(2)/(4-cbrt(16))                        // d2/2                (the numbers
    are halved to save computation time later)


// much faster than pow(x, 3.0/2.0)
__attribute__((const))double pow32(double x) {
        return x*sqrt(x);
}

/*Rc and Rs are R*cosθ() and R*sinθ(). Since they have to be calculated multiple times, they
    are given as paramters to decrease execution times*/

// B1 and B2 are some common base terms, which are given as parameters to save computation
    time (2-3x improvement)
__attribute__((const)) double V2_r(double r,double Rc, double B1, double B2){
    return B1*(r+mu*Rc)+B2*(r-Rc);
}

__attribute__((const)) double θV2_(double rRs, double B1, double B2){
    return (B2-mu*B1)*rRs;

```

```c
}

__attribute__((const)) double V2_z(double z, double B1, double B2){
    return z*(B1+B2);
}


__attribute__((const)) double B_1(double r, double rRc, double z, double R){
        return Gm_S/pow32(r*r+z*z+2*mu*rRc+R*R*mu*mu);
}


__attribute__((const)) double B_2(double r, double rRc, double z, double R){
        return Gm_J/pow32(r*r+z*z-2*rRc+R*R);
}



// Normal orbit calculations. Not optimised since it is unstable for our system. Could be commented.
__declspec(dllexport) void calc_orbits_c(double wn[][6], double R, size_t N, double dt, const char* method){

        double ω_J = sqrt(Gm_S/(R*R*R));
        dt = dt*year;

        if(strcmp(method, "Yoshida")==0){
                double c1 =1/(8-cbrt(128));                              // c1 = c4
        double c2 =(1-cbrt(2))/(4-cbrt(16));                     // c2 = c3
                double buffer[6];
                void mat_add(double *vec1, double *vec2, double N){  //add vector 2 to vector 1 and return the result to vector 1
        for(int i=0; i<N; i++){
                vec1[i] += vec2[i];
        }
}

double *mul_c(double c, double* vec, int N){    //multiplies a vector with a constant and changes its value. Size of array needed
        for(double *ptr=vec; ptr<vec+N; ptr++){
                *ptr *= c;
        }
        return vec;
}
// Potential
double V(double r, double rRc, double z, double R){
   return -(Gm_S/sqrt(pow(r,2)+pow(z,2)+2*mu*rRc+pow(R*mu,2))+Gm_J/sqrt(pow(r,2)+pow(z,2)-2*rRc+pow(R,2)));
}

// V derivatives:

double V_r(double r,double Rc, double z, double R){
    return (Gm_S*(r+mu*Rc)/pow32(pow(r,2)+pow(z,2)+2*mu*r*Rc+pow(R*mu,2))+Gm_J*(r-Rc)/pow32(pow(r,2)+pow(z,2)-2*Rc*r+pow(R,2)));
}

double θV_(double r, double rRc, double rRs, double z, double R){
    return ((-Gm_J*rRs)/pow32(pow(r,2)+pow(z,2)+2*mu*rRc+pow(R*mu,2))+Gm_J*(rRs)/pow32(pow(r,2)+pow(z,2)-2*rRc+pow(R,2)));

}

double V_z(double r, double rRc, double z, double R){
    return z*(Gm_S/pow32(pow(r,2)+pow(z,2)+2*mu*rRc+pow(R*mu,2))+Gm_J/pow32(pow(r,2)+pow(z,2)-2*rRc+pow(R,2)));
}

                // Acceleration part of the differential vector
                double *a(double x[3], double v[3]){
                        double r, θ, z, p, l_z, p_z;
                        r = x[0];θ
                         = x[1];
                        z = x[2];
                        p = v[0];
                        l_z = v[1];
                        p_z = v[2];
                        double Rc = R*cosθ();
                        buffer[0] = pow(l_z,2)/pow(r,3)-V_r(r, Rc , z, R), buffer[1]=-θV_(r,r
                            *Rc, r*R*sinθ(), z, R), buffer[2]= -V_z(r, r*Rc, z, R);
                        return buffer;
                }

                // Velocity part of the differential vector
                double *b(double x[3], double v[3]){
                        double r, θ, z, p, l_z, p_z;
                        r = x[0];θ
                         = x[1];
                        z = x[2];
                        p = v[0];
                        l_z = v[1];
                        p_z = v[2];
                        //double *vecb = malloc(sizeof(double)*3);
                        buffer[0] = p, buffer[1]=l_z/pow(r,2)ω-_J, buffer[2] = p_z;
                        return buffer;
                }

                void Yoshida(double wn[][6], int i){                            // i is the
                    iterative
                        double x[3] = {wn[i][0], wn[i][1], wn[i][2]};
                        double v[3] = {wn[i][3], wn[i][4], wn[i][5]};
                        //dt = dt*year;
                                                            // convert dt to seconds

                        mat_add(x,  mul_c(c1*dt, b(x,v), 3), 3);
                        mat_add(v,  mul_c(d1*dt/2, a(x,v), 3), 3);
                        mat_add(x,  mul_c(c2*dt, b(x,v), 3), 3);
                        mat_add(v,  mul_c(d2*dt/2, a(x,v), 3), 3);
                        mat_add(x,  mul_c(c2*dt, b(x,v), 3), 3);
                        mat_add(v,  mul_c(d1*dt/2, a(x,v), 3), 3);
                        mat_add(x,  mul_c(c1*dt, b(x,v), 3), 3);


                        wn[i+1][0] = x[0];
                        wn[i+1][1] = x[1];
                        wn[i+1][2] = x[2];
                        wn[i+1][3] = v[0];
                        wn[i+1][4] = v[1];
                        wn[i+1][5] = v[2];
                }

                size_t j;
                for(j=0; j<N; j++){
                        Yoshida(wn, j);

                }
        }

        if(strcmp(method,"Improved_Yoshida")==0){


                 __attribute__((always_inline)) void f(double x[6], double y[6], double h){

                        double r = x[0];
                        double θ = x[1];
                        double z = x[2];
                        double p = x[3];
                        double l_zdivr = x[4]/r;                        // l_z/r to improve
                            speed
                        double Rc = R*cosθ();
                        double B1 = B_1(r, r*Rc, z, R);
                        double B2 = B_2(r, r*Rc, z, R);


                        y[0] += h*p;
                        y[1] += h*(l_zdivr/rω-_J);
                        y[2] += h*x[5], y[3] += h*(l_zdivr*l_zdivr/r-V2_r(r, Rc, B1, B2));
                        y[4] +=-h*θV2_(r*R*sinθ(), B1, B2);
                        y[5] += -h*V2_z(z, B1, B2);

                }

                double h1 = d1*dt;
                double h2 = d2*dt;

                __attribute__((always_inline)) void A2(double h, double r_norm[6], double
                    p_norm[6]){
                            // h is some 'step size', depending on dt and Yoshida constants
                        f(p_norm, r_norm, h);
                        f(r_norm, p_norm, 2*h);
                        f(p_norm, r_norm, h);
                }


                // Bit slower, but fourth order!
                for(int i=0; i<N; i++){
                        double r_norm[6] = {wn[i][0], wn[i][1], wn[i][2], wn[i][3], wn[i][4],
                             wn[i][5]};                         // vector containing (r, p~)
                        double p_norm[6];

                        memcpy(p_norm, r_norm, sizeof(double)*6);

                                                  // Initially, the vectors equal

                        A2(h1, r_norm, p_norm);
                        A2(h2, r_norm, p_norm);
                        A2(h1, r_norm, p_norm);

                for(int j=0; j<6; j++){
                        wn[i+1][j] = (r_norm[j]+p_norm[j])/2.0;
                }
                }

        }
}

__declspec(dllexport) void migration_orbits_c(double wn[][6], double init_R, double tau,
    size_t N, double dt){

        double R = init_R;
        double ω_J;
                            // Jupiters angular velocity (rad/s)
        double t = dt/2;
                    // The time will be evaluated at the mean time of an iteration

        __attribute__((always_inline)) void f(double x[6], double y[6], double h){


                        double r = x[0];
                        double θ = x[1];
                        double z = x[2];
                        double p = x[3];
                        double l_zdivr = x[4]/r;                        // l_z/r to improve
                            speed
                        double Rc = R*cosθ();                           // Speed improvement
                        double B1 = B_1(r, r*Rc, z, R);
                        double B2 = B_2(r, r*Rc, z, R);


                        y[0] += h*p;
                        y[1] += h*(l_zdivr/rω-_J);
                        y[2] += h*x[5], y[3] += h*(l_zdivr*l_zdivr/r-V2_r(r, Rc, B1, B2));
                        y[4] +=-h*θV2_(r*R*sinθ(), B1, B2);
                        y[5] += -h*V2_z(z, B1, B2);

                }

                register double h1 = d1*dt*year;
                register double h2 = d2*dt*year;

        __attribute__((always_inline)) void A2(double h, double r_norm[6], double p_norm[6]){
                                                // h is some 'step size', depending on dt and
            Yoshida constants
                        f(p_norm, r_norm, h);
                        f(r_norm, p_norm, 2*h);
                        f(p_norm, r_norm, h);
                }

                //double εj = 0.04839266;
                            // Jupiters eccentricity. Uncomments when considering it

                //double vj = 0;
                            // Initial value of Jupiters True Anomly. Uncomment when
                    consideren Jupiters eccentricity

                // Bit slower, but fourth order!
                for(int i=0; i<N; i++){
                        double r_norm[6] = {wn[i][0], wn[i][1], wn[i][2], wn[i][3], wn[i][4],
                             wn[i][5]};                         // vector containing (r, p~)
                        double p_norm[6];
                        memcpy(p_norm, r_norm, sizeof(double)*6);

                                              // Initially, the vectors equal




                // For all different descriptions of the migration rate, it is important to
                    uncomment exactly one.
                //if(R<init_R+AU){

                                                    // Used for a constant migration
                        //      R+=AU*3*0.0001*dt;
                //}

                        R = (init_R + (1- exp(-t/tau))*1*AU);   // Standard migration
                            discription considerd. When considering the eccentricity, replace
                             R by double a            ω
                        _J = sqrt(Gm_S/(R*R*R));                                // Jupiters
                            angulat velocity. Comment when considering eccentricity

                        //R = init_R + exp(-tau/t)*AU;             // Alternative migration
                            descriprion (smoother start)

                        //Migration code//
                        //R = aε*(1-jε*j)ε/(1+j*cosv(j));ω
                        //_J = sqrt(Gm_Sε*(1+j*cosv(j))/(R*R*R));
                        //R = R*(1-mu);

                        // Midpoint Yoshida integration step
                        A2(h1, r_norm, p_norm);
                        A2(h2, r_norm, p_norm);
                        A2(h1, r_norm, p_norm);v


                        //j += ω_J*dt*year;                                              //
                            Update Jupiters true anomaly. Uncomment when considering Jupiters
                             eccentricity

                for(int j=0; j<6; j++){
                        wn[i+1][j] = (r_norm[j]+p_norm[j])/2.0;         // Write new values
                            to result matrix
                }
                t+=dt;
                                    // Update time
                }


}





// The following code could be uncommented and used for the alternative Trojan conversion
/*
__declspec(dllexport) void J2000(double wn[6], double dt, double epoch){
        double t=0;
        double t_end = epoch - 2354767.4;
        dt = dt*year;
                            // convert dt to seconds

        wn[4]=-wn[4], wn[5]=-wn[5], wn[3]=-wn[3];                                     // Time
            reversing, thus flipping velocity


        // Potential
        double V(double r, double θ, double z, double R){
                return -G*(m_S/sqrt(pow(r,2)+pow(z,2)));
        }

        // V derivatives:
        double V_r(double r,double θ, double z, double R){
                return G*(m_S*(r)/pow(pow(r,2)+pow(z,2),3.0/2.0));
        }

        double θV_(double r, double θ, double z, double R){
                return 0;
        }


        double V_z(double r, double θ, double z, double R){
                return G*(m_S*z/pow(pow(r,2)+pow(z,2),3.0/2.0));
        }

        double buffer[6];                                                                  //Max
             length of matrixes used


        double R = 5.20336301*AU;
        double ω_J = sqrt(G*m_S/pow(R,3));


        // Acceleration part of the differential vector
        double *a(double x[3], double v[3]){
                double r, θ, z, p, l_z, p_z;
                r = x[0];θ
                 = x[1];
                z = x[2];
                p = v[0];
                l_z = v[1];
                p_z = v[2];
                //double *veca = malloc(sizeof(double)*3);
                buffer[0] = pow(l_z,2)/pow(r,3)-V_r(r, θ, z, R), buffer[1]=0, buffer[2]= -V_z
                    (r, θ, z, R);
                return buffer;
        }

        // Velocity part of the differential vector
        double *b(double x[3], double v[3]){
                double r, θ, z, p, l_z, p_z;
                r = x[0];θ
                 = x[1];
                z = x[2];
                p = v[0];
                l_z = v[1];
                p_z = v[2];
                buffer[0] = p, buffer[1]=l_z/pow(r,2), buffer[2] = p_z;
                return buffer;
        }
        double *f(double x[6]){
                double r, θ, z, p, l_z, p_z;                                    // executed
                    nine times per timestep
                double ω_J = sqrt(G*m/pow(R,3));
                r = x[0];θ
                 = x[1];
                z = x[2];
                p = x[3];
                l_z = x[4];
                p_z = x[5];

                buffer[0] = p, buffer[1]=l_z/pow(r,2)ω-_J, buffer[2] = p_z, buffer[3] = pow(
                    l_z,2)/pow(r,3)-V_r(r, θ, z, R), buffer[4]=-θV_(r, θ, z, R), buffer[5]= -
                    V_z(r, θ, z, R);
                return buffer;
        }

        void midY(double wn[6], double dt){                                 // i is the
            iterative, midpoint Yoshida improved method
                double r_norm[6] = {wn[0], wn[1], wn[2], wn[3], wn[4], wn[5]};
                            // vector containing (r, p~)
                double p_norm[6];
                                                    // vector containing (r~, p)
                memcpy(p_norm, r_norm, sizeof(double)*6);
                            // Initially, the vectors equal

                        void A2(double vecpold[6], double vecrold[6], double h){
                                                                                // h is
                            some 'step size', depending on dt and Yoshida constants
                                mul_c(h, f(p_norm));
                                mat_add(r_norm, buffer);
                                mul_c(2*h, f(r_norm));
                                mat_add(p_norm, buffer);
                                mul_c(h, f(p_norm));
                                mat_add(r_norm, buffer);

                        }


                A2(r_norm, p_norm, 2*d1*dt*year);
                A2(r_norm, p_norm, 2*d2*dt*year);
                A2(r_norm, p_norm, 2*d1*dt*year);

                for(int j=0; j<6; j++){
                        wn[j] = (r_norm[j]+p_norm[j])/2.0;
                }
        }

        while(t<t_end){
                midY(wn, dt);
                t += dt;

        }
        wn[3] = -wn[3];
        wn[4] = -wn[4];
        wn[5] = -wn[5];
}*/
```

E.2. Python functions

Below is the code for all useful python functions. The location of the files should be changed to the directories of the operating computer. The code for the equipotential plots is an edited version of the code from [23].

```python
import numpy as np
import matplotlib.pyplot as plt
from numpy import sqrt, cos, sin, pi, matmul, cbrt
import time
from ctypes import c_double, c_int, CDLL, pointer, POINTER, c_size_t, windll, c_char_p
import ctypes
import math, os
import pickle
from Jupiter_physical_constants import * # All constants are in one file for easy acces
import csv
import bz2
from copy import deepcopy

# Imporing the ctools library
dirpath = os.path.dirname(__file__)                                           # directory
    path
ctools = np.ctypeslib.load_library("Trojan_ctools.dll", dirpath)              # import c
    library
doublep = POINTER(c_double)                                                   # Pointer
    type for double arrays
ctools.calc_orbits_c.argtypes = [doublep, c_double, c_size_t, c_double, c_char_p]
                      # C function Paramter types
ctools.calc_orbits_c.restype = None                                           # Void func
    has no return value
ctools.migration_orbits_c.argtypes = [doublep, c_double, c_double, c_size_t, c_double]
                      # C function Paramter types
ctools.migration_orbits_c.restype = None

# Needed for alternative Trojan conversion
#ctools.J2000.argtypes = [doublep, c_double, c_double]                  # C function
    Paramter types
#ctools.J2000.restype = None                                            # Void func has no
    return value


# Constants for Yoshida integrator
d1 = 1/(2-np.cbrt(2))                # d1 = d3
d2 = -np.cbrt(2)/(2-np.cbrt(2))      # d2
c1 = 1/(4-np.cbrt(16))               # c1 = c4
c2 = (1-np.cbrt(2))/(4-np.cbrt(16)) # c2 = c3


# After the orbits are calculated, they can be put into a class
class orbits:

    def __init__(self, cilindrical, t, dt, method='MYI'):
        t1 = time.time()
        self.r = cilindrical[0]
        selfθ. = cilindrical[1]
        self.z = cilindrical[2]
        self.p = cilindrical[3]
        self.l_z = cilindrical[4]
        self.p_z = cilindrical[5]
        self.t = t
        self.dt = dt
        self.method = method

    def save(self, name=None, dirname = ''):               # save as Python object
        if name == None:
            name = dirname+str(self.w0).replace(',',"_")+str(self.N)+'_'+str(self.dt)+'.obj'
        else:
            name = dirname+name+'.obj'
        file = open(name, 'wb')
        pickle.dump(self, file)
        file.close()

    @property
    def w0(self):
        return [self.r[0],selfθ.[0], self.z[0],self.p[0],self.l_z[0],self.p[0]]

    @property
    def N(self):
        return len(self.r)-1

    # Merging two consecuetive simulations of a Trojan
    def __add__(self, other):
        return orbits(np.array([np.append(self.r, other.r),np.append(selfθ., otherθ.),np.
            append(self.z, other.z),np.append(self.p, other.p),np.append(self.l_z, other.l_z)
            ,np.append(self.p_z, other.p_z)]), np.append(self.t, other.t) ,self.dt, self.
            method)

    # Returns the final Trojan position
    @property
    def position(self):
        return [self.r[-1],selfθ.[-1], self.z[-1],self.p[-1],self.l_z[-1],self.p[-1]]

    # Function for quickly plotting quantities. For the best axis labels, it is adviced to
        create the plots manually
    def plot(self, quantity, relative=False, show=True, save=False, mean=False, name=None,
        dirname=''):

        # Orbit in the regular cartesian x,y-plane
        if quantity == 'carthesian':
            r = self.rθ
             = selfθ.

            # Coordinates transform to carthesian coordinates
            x = r*cosθ()
            y = r*sinθ()
            plt.plot(x, y)
            plt.xlabel('$x$ (m)')
            plt.ylabel('$y$ (m)')
            if not save:
                plt.title('Orbit of a Trojan')

        # In astronomical units
        elif quantity == 'equipotential':
            r = self.rθ
             = selfθ.

            # Coordinates transform to carthesian coordinates
            x = r*cosθ()/AU
            y = r*sinθ()/AU
            make_plot(m_J/(m_J+m_S), 2)
            plt.plot(x, y, color= 'yellow', linewidth=.5)

        # Other quantities over time
        else:
            vec = getattr(self,quantity)

            # Relative change
            if relative:
                rel = (vec[1:]-vec[:-1])/vec[:-1]
                plt.plot(self.t[1:], rel)
                plt.xlabel('$t$ (y)')
                plt.ylabel('Relative change of $'+quantity+ '$')
                if mean:
                    mean_arr = [np.mean(rel[:i+1]) for i in range(len(self.t)-1)]
                    plt.plot(self.t[1:], mean_arr, color='yellow', label='mean')
                    plt.legend()
                if not save:
                    plt.title('Relative change of' + names[quantity] + ' of a Trojan over time')

            # Values
            else:
                plt.plot(self.t*self.dt, vec)
                plt.xlabel('$t$ (y)')
                plt.ylabel('$'+quantity+ '$ (' + units[quantity] +')')
                if mean:
                    mean_arr = [np.mean(vec[:i+1]) for i in range(len(self.t)-1)]
                    plt.plot(self.t[1:], mean_arr, color='yellow', label='mean')
                    plt.legend()
                if not save:
                    plt.title(names[quantity] + ' of a Trojan over time')

        if save:
            if relative:
                name = dirname+'_'+quantity+ '_rel_'+ str(self.w0).replace(',',"_")+str(self.N)+'_'+str(self.dt)+'.pdf'
            else:
                name = dirname+'_'+quantity+ '_'+ str(self.w0).replace(',',"_")+str(self.N)+'_'+str(self.dt)+'.pdf'
            plt.savefig(name, format='pdf')

        if show:
            plt.show()

    # Richardson error estimation. Works best when N is even
    def error(self):
        return [(getattr(calc_orbits(self.w0, int(self.N/2), 2*self.dt, b'Improved_Yoshida'), var)[-1]-getattr(self,var)[-1])/15 for var in ('r', 'θ', 'z')]

    @property
    def Lpoint(self):
        if not self.stable:
            return 'False'
        # Changed to half the running time to allow Horse Shoe orbits to become stable after a migration
        if max(selfθ.[int(len(self.t)/2):-1])-min(selfθ.[int(len(self.t)/2):-1])>pi:
            return 'None'
        if 0<selfθ.[-1]<pi:
            return 'L4'
        elif -pi<selfθ.[-1]<0:
            return 'L5'
        # Back up for when none of the creteria is met
        else:
            return 'error'

    @property
    def stable(self):
        # Once unstable in a run, it is assumed a Trojan always remains unstable
        if lenθ([ for θ in selfθ. if absθ()>2*pi])>0:
            return False
        else:
            return True

    # Richardson error estimation
    def mig_error(self):
        return [(getattr(migration_orbits(self.w0, r_Lgr, .5/10**4*AU, int(self.N/2), 2*self.dt),var)[-1]-getattr(self,var)[-1])/15 for var in ('r', 'θ', 'z')]
```

```python

    def Incl(self, i):                                # Inclination
        r, θ, z, p, l_z, p_z = self.r[i], selfθ.[i], self.z[i], self.p[i], self.l_z[i], self.p_z[i]
        return math.atan(sqrt(((r*p_z-z*p)/l_z)**2+(z/r)**2))

    # Hamiltonian
    @property
    def H(self):
        r, θ, z, p, l_z, p_z = self.r, selfθ., self.z, self.p, self.l_z, self.p_z
        return (p**2+p_z**2)/2+l_z**2/2/r**2+V(rθ,, z)-l_zω*_J


# draw lines of equipotentials for a rotating binary system.
#
# The techniques here come from "Astrophysics with a PC: An
# Introduction to Computational Astrophysics" by Paul Hellings
# The code is copied from the above sources and adjusted to the needs of this project

class Equipotentials(object):
    def __init__(self, mu, N):
        self.mu = mu
        self.N = N

        self.xmin = -1.3
        self.xmax = 1.3
        self.ymin = -1.3
        self.ymax = 1.3

        self.EPS = 1.e-12
        self.NITER = 100

        self.x = np.linspace(self.xmin, self.xmax, self.N, dtype=np.float64)
        self.y = np.linspace(self.ymin, self.ymax, self.N, dtype=np.float64)

        self.X, self.Y = np.meshgrid(self.x, self.y)

        # this is the dimensionless potential, in the z=0 plane
        self.V = self.Vf(self.X, self.Y)

    def get_L1(self):
        # find L1
        x0 = 0.0

        x_L1 = self._solve(-1.0, 1.0, x0)
        y_L1 = 0.0
        V_L1 = self.Vf(x_L1, y_L1)

        return x_L1, y_L1, V_L1

    def get_L2(self):
        # find L2
        x0 = -1.0

        x_L2 = self._solve(-1.0, -1.0, x0)
        y_L2 = 0.0
        V_L2 = self.Vf(x_L2, y_L2)

        return x_L2, y_L2, V_L2

    def get_L3(self):
        # find L3
        x0 = 1.0

        x_L3 = self._solve(1.0, 1.0, x0)
        y_L3 = 0.0
        V_L3 = self.Vf(x_L3, y_L3)

        return x_L3, y_L3, V_L3

    def get_L4(self):
        # L4
        x_L4 = self.mu - 0.5
        y_L4 = math.sin(math.pi/3.0)

        return x_L4, y_L4

    def get_L5(self):
        # L5
        x_L5 = self.mu - 0.5
        y_L5 = -math.sin(math.pi/3.0)

        return x_L5, y_L5

    def _solve(self, a, b, x0):

        for n in range(self.NITER):
            dVX = self.dVXdx(a, b, x0)
            d2VX = self.d2VXdx2(a, b, x0)

            x1 = x0 - dVX/d2VX
            err = abs(x1 - x0)/abs(x0 + self.EPS)

            if err < self.EPS:
                break
            x0 = x1

        return x0

    def Vf(self, x, y):
        V = (1.0 - self.mu) / np.sqrt((x - self.mu)**2 + y**2) + \
            self.mu/np.sqrt((x + 1.0 - self.mu)**2 + y**2) + \
            0.5 * (x**2 + y**2)
        return V

    def dVXdx(self, h1, h2, x):
        # this is the first derivative of V(x, y = 0) -- this is used
        # to find L1, L2, and L3 (all of which lie on y = 0).
        #
        # Here, h1 and h2 are sign parameters:
        #
        # h1 = sign(x - mu)
        # h2 = sign(x + 1 - mu)
        #
        # these appear in the denominator of V when we set y = 0,
        # and take the sqrt([...]^2) as an absolute value

        dVX = -h1*(1.0 - self.mu)/(x - self.mu)**2 - \
               h2*self.mu/(x + 1.0 - self.mu)**2 + x
        return dVX

    def d2VXdx2(self, h1, h2, x):
        # this is the second derivative of V(x, y = 0) -- this is used
        # to find L1, L2, and L3 (all of which lie on y = 0).

        d2VX = 2.0*h1*(1.0 - self.mu)/(x - self.mu)**3 + \
               2.0*h2*self.mu/(x + 1.0 - self.mu)**3 + 1.0
        return d2VX

# Creates a trojan object from inital values and can calculate an initial value
class Trojan:
    def __init__(self, a, ε, I, ω, Ω, M, epoch, set_dt):
        a, ε, I, ω, Ω, M, epoch = float(a)*AU, floatε(), float(I)/180*pi, floatω()/180*pi,
            floatΩ()/180*pi, float(M)/180*pi, float(epoch)

        # Trojan Kepler elements (saved to object)
        self.a = a                                                        # Semi-major axis
             (m)
        selfε. = ε                                                        # Eccentricity
        self.I = I                                                        # Inclination (
            rad)
        selfω. = ω                                                          # Argument of
            perihelion (rad)
        selfΩ. = Ω                                                          # Ascending node
            (rad)

        self.M = M                                                          # mean anomaly (
            rad)v
         = M + εε(2*-**3/4)*sin(M)ε+5/4***2*sin(2*M) ε+13***3/12*sin(3*M)  # True anomaly (
             rad) (Fourier series estimation)
        selfv. = v
        self.epoch = epoch                                                  # Epoch

        self.r = aε*(1-**2)ε/(1+*cosv())*matmul(Kepler_rot_mat(self.I, selfω., selfΩ.),[cos
            v(), sinv(), 0])                    # distance to sun (m)
        self.v = sqrt(G*(m_S)/(a*sqrtε(1-**2)))*matmul(Kepler_rot_mat(self.I, selfω., selfΩ.)
            ,[-sinv(), cosv() + ε,0])       # absolute velocity (m/s)
        w0 = np.array([*self.r, *self.v], dtype=c_double)
                                                                               # Temporarily
            in carthesian units
        del a, ε, I, ω, Ω, M, v

        # Relevant Kepler elements for Jupiter
        Ij = 1.30530                                                                    #
            Inclination (deg)Ω
        j = 100.55615                                                                  #
            Argument of perihelion (deg)ω
        j = Ω14.75385-j                                                                #
            Ascending node (deg)

        Mj = ω34.40438-jΩ-j                                                             #
            Mean anomaly at J2000 (deg)

        Mj = Mj + 0.08290676319200468*(epoch - 2451545.0)                               #
            Using Jupiters mean motion to estimate the mean anomaly at the Trojan epoch

        # Convert to radiansε
        j = 0.04839266
        Ij = Ij/180*piΩ
        j = Ωj/180*pi                                                                  #
            Argument of perihelion (rad)ω
        j = ωj/180*pi
        Mj = Mj/180*piv
        j = Mj + ε(2*jε-j**3/4)*sin(Mj)ε+5*j**2/4*sin(2*Mj) ε+13*j**3/12*sin(3*Mj)

        # Matrix to transform Jupiters orbit to the polar plane
        M3 = np.array([[cosΩ(j), sinΩ(j),0],[-sinΩ(j),cosΩ(j),0],[0,0,1]])
        M2 = np.array([[1,0,0],[0, cos(Ij), sin(Ij)],[0, -sin(Ij), cos(Ij)]])
        M1 = np.array([[cosω(j), sinω(j),0],[-sinω(j), cosω(j), 0],[0, 0, 1]])
        R_inverse = np.matmul(np.matmul(M1,M2),M3)

        # Transfering the Trojan with Jupiter
        w0 = [*matmul(R_inverse, w0[:3]), *matmul(R_inverse, w0[3:])]                   #
            Temporarily in Cartesian coordinates
        A = np.array([[cosv(j),sinv(j),0],[-sinv(j), cosv(j), 0],[0,0,1]])
        w0 = [*matmul(A, w0[:3]), *matmul(A, w0[3:])]
        w0[0] = w0[0]-m_J/m_S*R                                                         #
            Defining r with respect to the centre of mass


        # Transfering w0 to cylindrical coordinates
        r = sqrt(w0[0]**2+w0[1]**2)
        z= w0[2]θ
         = np.arctan2(w0[1],w0[0])
        p = w0[3]*cosθ()+w0[4]*sinθ()
        l_z = -w0[3]*r*sinθ()+w0[4]*r*cosθ()
        p_z = w0[5]
        self.w0 = [r, θ, z, p, l_z, p_z]                                                # Now
             in cylindrical coordinates

    # Plotting the position in the horizontal plane
    def plot(self, color='r', save=False, show = False, name=None, dirname=''):
```

```python

        x = self.w0[0]*cos(self.w0[1])
        y = self.w0[0]*sin(self.w0[1])

        plt.plot(x, y, color, '.', linewidth=1)

        if show:
            make_plot(m_J/(m_J+m_S))
            plt.show()

    # Hamiltonian
    @property
    def H(self):
        r, θ, z, p, l_z, p_z = self.w0
        return (p**2+p_z**2)/2+l_z**2/2/r**2+V(rθ,, z)-l_zω*_J


def Kepler_rot_mat(I, ω, Ω):                                                            # Rotation matrix for Kepler orbits
    M1 = np.array([[cosΩ(), -sinΩ(),0],[sinΩ(),cosΩ(),0],[0,0,1]])
    M2 = np.array([[1,0,0],[0, cos(I), -sin(I)],[0, sin(I), cos(I)]])
    M3 = np.array([[cosω(), -sinω(),0],[sinω(), cosω(), 0],[0, 0, 1]])
    return np.matmul(np.matmul(M1,M2),M3)


def Incl(r, θ, z, p, l_z, p_z):                             # Inclination
    return math.atan(sqrt(((r*p_z-z*p)/l_z)**2+(z/r)**2))   # Faster than numpy


# Load initial data from sheet
def load_Trojan_values(sheet, set_dt, filetype='obj'):
    if filetype == 'obj':
        file = open(sheet + '.obj', 'rb')
        Trojans = pickle.load(file)
        file.close()

    else:
        file=open(sheet+'.csv', "r")
        reader=(list(csv.reader(file))[1:])
        Trojans = list(map(lambda x: Trojan(*x, set_dt), reader))
        file.close()

        name = sheet + '1.obj'
        file = open(name, 'wb')
        pickle.dump(Trojans, file)
        file.close()
        file = open(name, 'rb')
        Trojans = pickle.load(file)
        file.close()

    return Trojans


def make_plot(mu, maxscale=1):

    R=r_Lgr/AU
    eq = Equipotentials(mu, 1024)

    plt.imshow(np.log10(eq.V), origin="lower", cmap="Accent",
               extent=[eq.xmin*R*maxscale, eq.xmax*R*maxscale, eq.ymin*R*maxscale, eq.ymax*R*maxscale])

    # draw contours -- these values seem reasonable for a range of mu's
    Vmin = 1.5
    Vmax = 1000.0  # np.max(V)
    nC = 25

    C = np.logspace(math.log10(Vmin), math.log10(Vmax), nC)

    plt.contour(-eq.x*R, eq.y*R, eq.V, C, colors="b")

    x_L1, y_L1, V_L1 = eq.get_L1()
    x_L2, y_L2, V_L2 = eq.get_L2()
    x_L3, y_L3, V_L3 = eq.get_L3()

    # special contours right through the lagrange points
    plt.contour(eq.x*R, eq.y*R, eq.V, [V_L1], colors="b")
    plt.contour(eq.x*R, eq.y*R, eq.V, [V_L2], colors="b")
    plt.contour(eq.x*R, eq.y*R, eq.V, [V_L3], colors="b")

    # mark the Lagrange points and write the names
    xeps = 0.025
    # changed some distances to have it in the regular plane
    plt.scatter([-x_L1*R], [-y_L1*R], marker="x", color="r", s=50)
    plt.text(-x_L1*R+xeps*R, -y_L1*R+xeps*R, "L1", color="r")

    plt.scatter([-x_L2*R], [-y_L2*R], marker="x", color="r", s=50)
    plt.text(-x_L2*R+xeps*R, -y_L2*R+xeps*R, "L2", color="r")

    plt.scatter([-x_L3*R], [-y_L3*R], marker="x", color="r", s=50)
    plt.text(-x_L3*R+xeps*R, -y_L3*R+xeps*R, "L3", color="r")

    x_L4, y_L4 = eq.get_L4()
    plt.scatter([-x_L4*R], [y_L4*R], marker="x", color="r", s=50)
    plt.text(-x_L4*R+xeps*R, y_L4*R+xeps*R, "L4", color="r")

    x_L5, y_L5 = eq.get_L5()
    plt.scatter([-x_L5*R], [y_L5*R], marker="x", color="r", s=50)
    plt.text(-x_L5*R+xeps*R, y_L5*R+xeps*R, "L5", color="r")

    plt.axis([eq.xmin*R, eq.xmax*R, eq.ymin*R, eq.ymax*R])

    #plt.title(r"Equipotentials, $\mu = M_2/(M_1 + M_2) = {:5.3f}$".format(mu), fontsize=12)

    plt.xlabel("$x$ (AU)")
    plt.ylabel("$y$ (AU)")

    f = plt.gcf()
    f.set_size_inches(10.8, 10.8)

    plt.tight_layout()

def load_orbits(file, dirname = ''):                    # Load orbit data python object
    file = open(dirname+file, 'rb')
    temp = pickle.load(file)
    file.close()
    return temp

### Functions to easily call the C functions ###
# Regular orbit calculation
def calc_orbits(w0, R, N, dt, method=b'Improved_Yoshida'):
    w0 = np.array(w0, dtype=c_double)                   # Inital values
    wn = (np.zeros((N+1,6), dtype=c_double) )         # dtype needs to be defined for C
    wn[0] = w0                                        # Plugging in the initial value
    wn_c = wn.ctypes.data_as(doublep)
    ctools.calc_orbits_c(wn_c, R, N, dt, method)                   # The pointer needs to be
        passed trough the C function. Also the size is needed.
    return orbits(wn.T, np.arange(N+1), dt=dt, method=str(method))

# Orbits with migrating Jupiter
def migration_orbits(w0, init_R, tau, N, dt):
    w0 = np.array(w0, dtype=c_double)                   # Inital values
    wn = (np.zeros((N+1, 6), dtype=c_double))           # dtype needs to be defined for C
    wn[0] = w0                                          # Plugging in the initial value
    wn_c = wn.ctypes.data_as(doublep)
    ctools.migration_orbits_c(wn_c, init_R, tau, N, dt)                  # The pointer needs
         to be passed trough the C function. Also the size is needed.
    return orbits(wn.T, np.arange(N+1), dt=dt, method='migration')

### ###

# Solving the cubic equation
```

```python
def Cardano(a,b,c,d):
    R = complex((9*a*b*c-27*a**2*d-2*b**3)/(54*a**3))
    Q = complex((3*a*c-b**2)/(9*a**2))
    def ccbrt(a):
        import cmath
        r, phi = cmath.polar(a)
        return cbrt(r)*cmath.exp(phi/3*1j), cbrt(r)*cmath.exp((phi+2*pi)/3*1j), cbrt(r)*cmath
            .exp((phi+4*pi)/3*1j)


    Sall = ccbrt(R+sqrt(Q**3+R**2))
    Tall = ccbrt(R-sqrt(Q**3+R**2))

    ls = []
    for S in Sall:
        for T in Tall:
            if abs((S * T) + Q) < abs(Q)*10e-6:
                ls += [S+T - b/(3*a)]
    return ls

# Characteristic equation for the Jacobian
def solve_char_equation(r, θ, z, p, l_z, p_z):
    Vrr = V_rr(rθ,,z)θ
    Vr = θV_r(rθ,,z)
    Vrz = V_rz(rθ,,z)
    Vzz = V_zz(r, θ, z)θ
    Vz = θV_z(rθ,,z)θθ
    V = θθV_(rθ,,z)

    a=r**2
    b=r**2*Vrr+r**2*Vzz+θθV+3*l_z**2/r**2
    c = θθV*(Vzz+Vrr-l_z**2/r**4)+Vzz*(r**2*Vrr+3*l_z**2/r**2)-r**2*(Vrz)**2-θVz**2-θVr**2
    d = Vrr*(Vzz*θθV-θVz**2)-θθV*(Vrz**2+l_z**2/r**4*Vzz)-Vzz*θVr**2+l_z**2/r**4*θVz**2+2*Vrz
        *θVz*θVr

    return *sqrt(Cardano(a,b,c,d)), *(-sqrt(Cardano(a,b,c,d)))


# Funtions for the numerical integrators. These are implemented in C for a faster simulation
    and therefore commented
##def f(wn):
##    return np.append(b(wn), a(wn))
##
### Acceleration part of the differential vector
##def a(wn):
##     r, θ, z, p, l_z, p_z = wn[0], wn[1], wn[2], wn[3], wn[4], wn[5]
##     return np.array([l_z**2/r**3-V_r(r, θ, z), -θV_(r, θ, z), -V_z(r, θ, z)])
##
### Velocity part of the differential vector
##def b(wn):
##     r, θ, z, p, l_z, p_z = wn[0], wn[1], wn[2], wn[3], wn[4], wn[5]
##     return np.array([p, (l_z/rω**2-_J), p_z])

# Potential
def V(rθ,, z):
    m = m_J+m_S
    return -G*(m_S/sqrt(r**2+z**2+2*m_J/m_S*R*r*cosθ()+(R*m_J/m_S)**2)+m_J/sqrt(r**2+z**2-2*R
        *r*cosθ()+R**2))

# V derivatives. Second order derivatives are only used in the Jacobian and therefore
    estimated
def V_r(rθ,, z):
    m = m_J+m_S
    return G*(m_S*(r+m_J/m_S*R*cosθ())/(r**2+z**2+2*m_J/m*R*r*cosθ()+(R*m_J/m_S)**2)**(3/2)+
        m_J*(r-R*cosθ())/(r**2+z**2-2*R*r*cosθ()+R**2)**(3/2))

def θV_(r, θ, z):
    m = m_J+m_S
    return G*((-m_J*R*r*sinθ())/(r**2+z**2+2*m_J/m_S*R*r*cosθ()+(R*m_J/m_S)**2)**(3/2)+m_J*(r
        *R*sinθ())/(r**2+z**2-2*R*r*cosθ()+R**2)**(3/2))

```

```
def V_z(r, θ, z):
    m = m_J+m_S
    return G*z*(m_S/(r**2+z**2+2*m_J/m_S*R*r*cosθ()+(R*m_J/m_S)**2)**(3/2)+m_J/(r**2+z**2-2*R*r*cosθ()+R**2)**(3/2))

def V_rr(rθ,, z):
    dr = r/(10e6)
    return (V_r(r+dr, θ, z)-V_r(r, θ, z))/dr

def θV_r(r, θ, z):θ
    d = θ/1000
    return (V_r(r, θ+θd, z)-V_r(r, θ, z))/θd

def V_rz(r, θ, z):
    dz = r/(10e6)
    return (V_r(r, θ, z+dz)-V_r(r, θ, z))/dz

def V_zz(r, θ, z):
    dz = r/(10e6)
    return (V_z(r, θ, z+dz)-V_z(r, θ, z))/dz

def θθV_(r, θ, z):
    return -3/2*G*((m_J**2/m_S*R**2*r**2*sinθ()**2)/(r**2+z**2+2*m_J/m_S*R*r*cosθ()+(R*m_J/m_S)**2)**(5/2)+m_J*(r*R*sinθ())**2/(r**2+z**2-2*R*r*cosθ()+R**2)**(5/2))+θV_(r, θ, z)*cosθ()/sinθ()

def θV_z(r, θ, z):
    dz = r/(10e6)
    return (θV_(r, θ, z+dz)-θV_(r, θ, z))/dz

# Jacobian matrix
def Jacobian(r, θ, z, p, l_z, p_z):
    return np.array([[0,0,0,1,0,0],[-2*l_z/r**3,0,0,0,1/r**2,0],[0,0,0,0,0,1],[-3*l_z**2/r**4-V_rr(r, θ, z), -θV_r(r, θ, z), -V_rz(r, θ, z),0, 2*l_z/r**3,0],[-θV_r(r, θ, z), -θθV_(r, θ, z), -θV_z(r, θ, z),0,0,0],[-V_rz(r, θ, z), -θV_z(r, θ, z), -V_zz(r, θ, z),0,0,0]])


# Richard estimation for the order. Returns 2^{order}, provided dt and N are 'applicable' for the estimation.
# This is assumed to be true if 'order' is almost a whole number.
def order(w0, N, dt, method, testvar):
    if method == 'migration':
        (getattr(migration_orbits(w0, 2*N, 2*dt, method), testvar)[-1]-getattr(migration_orbits(w0, N, 4*dt, method), testvar)[-1])/(getattr(migration_orbits(w0, 4*N,dt, method),testvar)[-1]-getattr(migration_orbits(w0, 2*N, 2*dt, method), testvar)[-1])

    return (getattr(calc_orbits(w0, 2*N, 2*dt, method), testvar)[-1]-getattr(calc_orbits(w0, N, 4*dt, method), testvar)[-1])/(getattr(calc_orbits(w0, 4*N,dt, method),testvar)[-1]-getattr(calc_orbits(w0, 2*N, 2*dt, method),testvar)[-1])

def randomise_w0(bounds):             #inputs are vectors with the bounds. Only the positions are randomised, initial velocities are zero in the rotating frame
    r   = np.random.uniform(*bounds[0])θ
       = np.random.uniform(*bounds[1])
    z   = np.random.uniform(*bounds[2])
    t1 = time.time()

    return np.array([r, θ, z, 0, sqrt(G*m_S/R**3)*r**2, 0])
```

E.3. Jupiter constants

Below is the code for the constants of Jupiter used in the Python codes (be aware that they are defined separately in the C code):

```
from numpy import sqrt
# Physical constants (source to be added)
G       = 6.6384e-11         # Gravitational constant
m_S     = 1.98847e30         # Mass sun (kg)
```

```
m_J     = 1.89813e27         # Mass Jupiter (kg)
AU      = 1.496e11           # Astronomical unit (m)
r_Lgr   = 5.20336301*AU      # Lagrange distance, Jupiter distance (m)ω
_J = sqrt(G*(m_S+m_J)/r_Lgr**3)      # Jupiter angular velocity
R=r_Lgr

#We will work in years to prevent numerical overflows:
year = 3600*24*365           # Convert year to seconds

# Some information directories used for plotting
units = {'r': 'm', 'θ': 'rad', 'z': 'm', 'p': 'm/s', 'l_z': 'rad/s', 'p_z': 'm/s', 'H': 'Jkg$
    ^{-1}$'}
names = {'r': 'Radius', 'θ': 'Angle θ', 'z': 'Hight z', 'p': 'Momentum', 'l_z': 'Angular
    momentum', 'p_z': 'Momentum z', 'H': 'Hamiltonian'}
```

E.4. Integrator choice

```
# Important run constants:
'''Only put the following True if the C code has been changed or will be changed regularly'''
CCompile = False

import numpy as np
import matplotlib.pyplot as plt
from numpy import sqrt, cos, sin, pi
import time
import math
from ctypes import POINTER, c_double
import ctypes
from scipy.spatial.transform import Rotation

# Compile the C code if needed
if CCompile:
    import os, subprocess
    dirpath = os.path.dirname(__file__)
    cmd = ['C:\\Users\\jaspe\\Downloads\\winlibs-x86_64-posix-seh-gcc-12.2.0-llvm-16.0.0-
        mingw-w64ucrt-10.0.0-r5\\mingw64\\bin\\gcc', '-shared','-Os', '-s', '-o', dirpath+'\\
        Trojan_ctools.dll', dirpath+'\\Trojan_ctools.c']
    p = subprocess.run(cmd, cwd='C:\\', capture_output=True)
    print(p)



# Modules files
from Trojan_pythonfunctions import orbits, V, calc_orbits, order, randomise_w0, load_orbits,
    migration_orbits, ctools, Trojan, load_Trojan_values, solve_char_equation, Jacobian
from Jupiter_physical_constants import * # All constants are in one file for easy acces
doublep = POINTER(c_double)


# Constants numerical method (can be overruled later in the code)
dt = 1/300  # Time step (y)
N = 100000  # Number of simulations w0 =
w0 = np.array([r_Lgr, pi/3, 0, 0, sqrt(G*m_S*r_Lgr), 0])                 # r, theta, z, p,
    l_z,
bounds = [[r_Lgr*0.9, r_Lgr*1.1], [pi/4, pi/2], [-10**6, 10**6]]    # bounds for generating
    random initial values w0
dirname = 'Integrator_comp\\'

Trojans = load_Trojan_values('Trojan_data_20230514', dt/5, filetype='obj')
dirname = 'Trojan_data_dt0.0033333_N100000\\'

print(Jacobian(R, pi/3, 0, 0, sqrt(G*m_S*R), 0))
print(R,sqrt(G*m_S*R), r_Lgr)
print('\n\n')
print(np.linalg.eig(Jacobian(R, pi/3, 0, 0, sqrt(G*m_S*R), 0))[0])
print(max(np.abs(np.linalg.eig(Jacobian(R, pi/3, 0, 0, sqrt(G*m_S*R), 0))[0])))
print(solve_char_equation(R, pi/3, 0, 0, sqrt(G*m_S*R), 0))

t1 = time.time()
```

```
nums = [i for i in range(1,5)]
Tlist = list(map(lambda i: calc_orbits(Trojans[i].w0, N, dt, b'Improved_Yoshida'), nums))
#list(map(lambda T: T.plot(quantity='H', relative=False, save=True, mean=False, dirname=
    dirname, show=False), Trojans))
#list(map(lambda T: T.plot(quantity='carthesian',save=True, mean=False, dirname=dirname, show
    =False), Trojans))
#list(map(lambda T: T.plot(quantity='equipotential',save=True, mean=False, dirname=dirname,
    show=False), Trojans))
print(time.time()-t1)

for i in range(3):
    #Torbit[i].check_stable
    Tlist[i].plot(quantity='H',save=False, mean=False, dirname=dirname)
    Tlist[i].plot(quantity='carthesian',save=False, mean=False, dirname=dirname)
    Tlist[i].plot(quantity='equipotential',save=False, mean=False, dirname=dirname)

'''
for i, T in enumerate(Trojans):
    w1 = T.w0
    Torbit = calc_orbits(w1, N, dt, b'Improved_Yoshida')

    Torbit.save('Torbit'+str(i), dirname)
    #Trojanorbit.plot(quantity='H', relative=False, save=False, mean=False, dirname=dirname)
    #Trojanorbit.plot(quantity='carthesian',save=True, mean=False, dirname=dirname)
    #Trojanorbit.plot(quantity='equipotential',save=True, mean=False, dirname=dirname)
print('done. runtime: ', time.time()-t1)
exit()
'''



## Testing the two Yoshida integrators for a L5 near Trojan
'''
N = 100000
# Testing the two orders
print(order(w0, 100, 1/30, b'Improved_Yoshida', 0''))
print(order(w0, 100, 1/30, b'Yoshida', 0''))
'''
#w1 = np.array([r_Lgr/2, -pi/3, 0, 0, sqrt(G*m_S*r_Lgr/2), 0])                 # r, theta, z,
     p, l_z,
MYIorbit = calc_orbits(w1, N, dt, b'Improved_Yoshida')
#print(Migorbit.r[0:2])
dirname = 'Trojan_data\\'
MYIorbit.plot(quantity='H', relative=False, save=True, mean=False, dirname=dirname)
MYIorbit.plot(quantity='carthesian',save=True, mean=False, dirname=dirname)
MYIorbit.plot(quantity='equipotential',save=True, mean=False, dirname=dirname)
exit
#Migorbit.save('Mig_100000_0.333y_nearstable1L4', dirname)

'''
# Saving the data
dirname = 'Integrator_comp\\'
MYIorbit.save('MYI_100000_0.333y_nearstable', dirname)
OYIorbit.save('OYI_100000_0.333y_nearstable', dirname)
'''
'''
# Generating and saving the Hamiltonian_plots
MYIorbit.plot(quantity='H', relative=True, save=True, mean=True, dirname=dirname)
MYIorbit.plot(quantity='H',save=True, mean=False, dirname=dirname)
OYIorbit.plot(quantity='H', relative=True, save=True, mean=True, dirname=dirname)
OYIorbit.plot(quantity='H', relative=False, save=True, mean=False, dirname=dirname)
'''
'''
N = 100000
MYIorbit = calc_orbits(w0, N, dt, b'Improved_Yoshida')
OYIorbit = calc_orbits(w0, N, dt, b'Yoshida')

# Saving the data
dirname = 'Integrator_comp\\'
MYIorbit.save('MYI_10000000_nearstable', dirname)
OYIorbit.save('OYI_10000000_nearstable', dirname)
```

```

#dirname = 'Integrator_comp\\'
#MYIorbit=load_orbits('MYI_100000_0.333y_nearstable.obj', dirname = dirname)
#OYIorbit=load_orbits('OYI_10000000_nearstable.obj', dirname = dirname)

#MYIorbit.plot(quantity='l_z',save=False, mean=False, dirname=dirname)
#OYIorbit.plot(quantity='H', relative=False, save=True, mean=False, dirname=dirname)
#MYIorbit.plot(quantity='carthesian',save=True, mean=False, dirname=dirname)
#MYIorbit.plot(quantity='equipotential',save=True, mean=False, dirname=dirname)
#OYIorbit.plot(quantity='carthesian',save=True, mean=False, dirname=dirname)
#OYIorbit.plot(quantity='equipotential',save=True, mean=False, dirname=dirname)
#MYIorbit.plot(quantity='H', relative =True, save=True, mean=False, dirname=dirname)
#OYIorbit.plot(quantity='H', relative=True, save=True, mean=False, dirname=dirname)
#MYIorbit = calc_orbits(w0, N, dt, b'Improved_Yoshida')
#MYIorbit.plot(quantity='equipotential',save=False, mean=False, dirname='' )
#MYIorbit.plot(quantity='carthesian')

'''
''''
N= 10000
for i in range(10):
    w0 = randomise_w0(bounds)
    x = calc_orbits(w0, N, dt, b'Improved_Yoshida')
    x.plot('carthesian')
    x.plot('equipotential')
    #x.plot('H', relative=True)
'''




'''
dt = 1/30
N = int(1e4)

w1 = np.array([r_Lgr, pi/3, 0, 0, ω_J*r_Lgr**2, 0])
t1 = time.time()
orbit2 = calc_orbits(w1, N, dt)
t2 = time.time()
print(t2-t1)
print(orbit2.p)

orbit2.plot(quantity='carthesian')
#orbit2.plot(quantity='H')
#orbit2.plot(quantity='H', relative=True)
#orbit2.plot(quantity='equipotential')

#t1 = time.time()
#orbit2 = calc_orbits(w0, N*10, dt)
#print(time.time()-t1)
'''


N=1000000
dirname = 'randomised_w0\\'
for i in range(10):
    w0 = randomise_w0()
    print(order(w0, 30, dt, b'Improved_Yoshida', 'θ'))
    MYIorbit=calc_orbits(w0, N, dt, b'Improved_Yoshida')
#OYIorbit=load_orbits('OYI_10000000_nearstable.obj', dirname = dirname)

    MYIorbit.plot(quantity='carthesian',save=False, mean=False, dirname=dirname)
    MYIorbit.plot(quantity='H',save=False, mean=False, dirname=dirname)
    MYIorbit.save(('randomised3'+str(i)), dirname)
    print(MYIorbit.r[0]/R)
#OYIorbit.plot(quantity='H', relative=False, save=True, mean=False, dirname=dirname)
```

E.5. Stability Trojans

The code below shows how the simulations where ran in parallel and how the data analyses are done. The code could be commented out to use only the parts needed and the graphs etc. could be adjusted to the exact graphs or variables desired. The code makes advantage of the library from appendix E.2.

```python
# Important run constants:
'''Only put the following True if the C code has been changed or will be changed regularly'''
CCompile = False #Tau edited!, exp no none smooth at start!

import numpy as np
import matplotlib.pyplot as plt
from numpy import sqrt, cos, sin, pi, arctan2, matmul
import time
import math
from ctypes import POINTER, c_double
from multiprocessing.pool import ThreadPool, Pool
from threading import Lock
from copy import deepcopy
import csv


# Compile the C code if needed
if CCompile:
    import os, subprocess
    dirpath = os.path.dirname(__file__)
    cmd = ['C:\\Users\\jaspe\\Downloads\\winlibs-x86_64-posix-seh-gcc-12.2.0-llvm-16.0.0-
        mingw-w64ucrt-10.0.0-r5\\mingw64\\bin\\gcc','-O3','-shared','-Os', '-s', '-o',
        dirpath+'\\Trojan_ctools.dll', dirpath+'\\Trojan_ctools.c']
    p = subprocess.run(cmd, cwd='C:\\', capture_output=True)
    print(p)

# Fast tau file: 'migtau500_300000_3_250x4T1707.txt'

# Modules files
from Trojan_pythonfunctions import orbits, calc_orbits, migration_orbits, Trojan,
    load_Trojan_values, make_plot, Kepler_rot_mat, Incl
from Jupiter_physical_constants import * # All constants are in one file for easy acces


# Loading Trojan data
dt = 1/30
Trojans = load_Trojan_values('Trojan_data_20230514', dt/5, filetype='obj')
#Trojans2 = deepcopy(Trojans)
Trojans3 = deepcopy(Trojans)     # Belonging to the primal symmetry
#Trojans4 = deepcopy(Trojans)


# Optionally, the other two symmetries could be used as well:
#def Tfunc1(T):
#    T.w0[1] = -T.w0[1]
#    T.w0[2] = -T.w0[2]
#    T.w0[3] = -T.w0[3]
#list(map(Tfunc,Trojans2))
#Trojans = Trojans+Trojans2

# Using the primal symmetry
def Tfunc2(T):
    T.w0[1] = -T.w0[1]
    T.w0[3] = -T.w0[3]
    T.w0[5] = -T.w0[5]
list(map(Tfunc,Trojans3))
Trojans = Trojans+Trojans3

# Optionally, the other two symmetries could be used as well:
#def Tfunc3(T):
#    T.w0[2] = -T.w0[2]
#    T.w0[5] = -T.w0[5]
#list(map(Tfunc,Trojans4))
#Trojans = Trojans+Trojans4


# Loading data without migration
normal = eval(open('enter filename', 'r').read())

```

```
# Example fast plots for a near equilibrium particle
P = calc_orbits([r_Lgr, pi/3,0,0,sqrt(G*m_S*r_Lgr),0], r_Lgr, 300000, 1/10, b'
    Improved_Yoshida')
P.plot('equipotential')
P = migration_orbits([r_Lgr, pi/3,0,0,sqrt(G*m_S*r_Lgr),0], r_Lgr, 1000, 300000, 1/10, b'
    Improved_Yoshida')
P.plot('equipotential')

# Generate random Trojans
def ranfunc():
        Ij = 1.30530                                                                   #
            Inclination (rad)Ω
        j = 100.55615                                                                 #
            Argument of perihelion (rad)ω
        j = Ω14.75385-j                                                               #
            Ascending node (rad)

        a = r_Lgr                                                                      #
            Semi-major axis (m)ε
         = np.random.uniform(0,0.3)                                                   #
             Eccentricity
        I = 0.01*180/pi                                                                #
            Inclination (rad)v

         = np.random.uniform(90/180*pi,140/180*pi)
                                                    # Epoch

        r = aε*(1-**2)ε/(1+*cosv())*np.array([cosv(), sinv(), 0])                      #
            distance to sun (m)
        v = sqrt(G*(m_S)/(a*sqrtε(1-**2)))*np.array([-sinv(), cosv() + ε,0])           #
            absolute velocity (m/s)
        w0 = np.array([*r, *v], dtype=c_double)
                                                                               # Temporarily
            in carthesian units
        A = np.array([[1,0,0],[0, cos(I), -sin(I)],[0,sin(I),cos(I)]])
        w0 = [*matmul(A, w0[:3]), *matmul(A, w0[3:])]
        w0[0] = w0[0]-m_J/m_S*R                                                        #
            Defining R with respect to the centre of mass


        # Transfering w0 to cylindrical coordinates
        r = sqrt(w0[0]**2+w0[1]**2)
        z= w0[2]θ
         = np.arctan2(w0[1],w0[0])
        p = w0[3]*cosθ()+w0[4]*sinθ()
        l_z = -w0[3]*r*sinθ()+w0[4]*r*cosθ()
        p_z = w0[5]
        return [r, θ, z, p, l_z, p_z]


# Generating random Trojans and saving the data
##wlist = [ranfunc() for i in range(10000)]
##wlist_90_140deg = [ranfunc() for i in range(10000)]
##w2list = deepcopy(wlist_90_140deg)
##file = open('wlist0708_90_140deg', 'w')
##file.write(str(wlist0708_90_140deg))
#file.close()

#list(map(Tfunc2,wlist_90_140deg))
#wlistnew =wlist_90_140deg + w2list

# Some functions to be able to obtain the results when running in parrallel
def unstable_func(i):
    T = calc_orbits(Trojans[i].w0, r_Lgr, 300000, 1/3, b'Improved_Yoshida')

    r, θ, z, p, l_z, p_z = T.position

    return [T.Lpoint, math.atan(sqrt(((r*p_z-z*p)/l_z)**2+(z/r)**2)), i]


def minmax_func(i):
```

```python

    T = calc_orbits(Trojans[i].w0, r_Lgr, 300000, 1/30, b'Improved_Yoshida')


    return [min(Tθ.),max(Tθ.),i]

def minmax_funcr(i):

    T = calc_orbits(Trojans[i].w0, r_Lgr, 30000, 1/3, b'Improved_Yoshida')


    return [min(T.r),max(T.r),i]

def mig_unstable_func(i):
    T = migration_orbits(Trojans[i].w0, r_Lgr, 1000, 3000000, 1/3)
    r, θ, z, p, l_z, p_z = T.position

    return [T.Lpoint, math.atan(math.sqrt(((r*p_z-z*p)/l_z)**2+(z/r)**2)), i, T.position]

# Running in a threadpool to use maximum computational power
nums = np.array([i for i in range(len(Trojans))])
pool = ThreadPool()#
result = pool.map(mig_unstable_func, nums)
pool.close()
pool.join()
file = open('savetofile', 'w')
file.write(str(result))
file.close()
print("TIME")
print(time.time()-t1)       # Print elapsed time

# Load migration file
migration = eval(open('enterfilename', 'r').read())



# If less Trojans are simulated in the migration file, we select the correct Trojan indices
    in the normal file, which allways consist of 50220 Trojans
def select_normT(normal, migration):
    l = int(len(migration)/4)
    indices = []
    for j in range(4):
        indices += [i for i in range(j*12555,l+j*12555)]
    return np.array([normal[i] for i in indices])


# Easily establishing the result:
def result_num_mat(normal, migration):
    nums = [i for i in range(len(normal))]
    x = np.array([len([i for i in nums if migration[i][0]=='L4' and normal[i][0]=='L4']), len
        ([i for i in nums if migration[i][0]=='L5' and normal[i][0]=='L4']), len([i for i in
        nums if migration[i][0]=='None' and normal[i][0]=='L4']), len([i for i in nums if
        migration[i][0]=='False' and normal[i][0]=='L4']),
    len([i for i in nums if migration[i][0]=='L4' and normal[i][0]=='L5']), len([i for i in
        nums if migration[i][0]=='L5' and normal[i][0]=='L5']), len([i for i in nums if
        migration[i][0]=='None' and normal[i][0]=='L5']), len([i for i in nums if migration[i
        ][0]=='False' and normal[i][0]=='L5']),
    len([i for i in nums if migration[i][0]=='L4' and normal[i][0]=='None']), len([i for i in
         nums if migration[i][0]=='L5' and normal[i][0]=='None']), len([i for i in nums if
        migration[i][0]=='None' and normal[i][0]=='None']), len([i for i in nums if migration
        [i][0]=='False' and normal[i][0]=='None']),
    len([i for i in nums if migration[i][0]=='L4' and normal[i][0]=='False']), len([i for i
        in nums if migration[i][0]=='L5' and normal[i][0]=='False']), len([i for i in nums if
         migration[i][0]=='None' and normal[i][0]=='False']), len([i for i in nums if
        migration[i][0]=='False' and normal[i][0]=='False'])], dtype=float)
    return x.reshape([4,4])


# Or stablish the required indexes of the normal simulations manually:
Listlen = 12555
nums = [i for i in range(Listlen)]+[i for i in range(12555*2,12555*2+Listlen)]
```

```python
normal = [normal[i] for i in nums]


L4 = [i for i in range(25110) if normal[i][0]=='L4']
L5 = [i for i in range(25110) if normal[i][0]=='L5']

# Determine the types after the migration
uL4 = [i for i in L4 if migration[i][0]=='False']
sL4 = [i for i in L4 if i not in uL4]
uL5 = [i for i in L5 if migration[i][0]=='False']
sL5 = [i for i in L5 if i not in uL5]


# Inclination plots L4 (adjust for zoomed version)
x = list(map(lambda i: Incl(*Trojans[i].w0)*180/pi, sL4))
y = list(map(lambda i: migration[i][1]*180/pi, sL4))
plt.plot(x, y, color=[0,0,0], marker='.', linestyle='', alpha=1, label='Stable')

x = list(map(lambda i: Incl(*Trojans[i].w0)*180/pi, uL4))
y = list(map(lambda i: migration[i][1]*180/pi, uL4))
plt.plot(x, y, color='r', marker='.', linestyle='', alpha=1, label='Unstable')

plt.ylabel(r"Inclination $I'$ after migration °()")
plt.xlabel(r'Inclination $I$ before migration °()')

plt.plot([0,200], [0,200], 'b', label='Reference line $I$=$I\'$')
plt.xlim(0,50)
plt.ylim(0,90)
plt.legend()
plt.savefig('saveL4withname.pdf', format='pdf')

plt.show()


# Inclination plots L5 (adjust for zoomed version)
x = list(map(lambda i: Incl(*Trojans[i].w0)*180/pi, sL5))
y = list(map(lambda i: migration[i][1]*180/pi, sL5))
plt.plot(x, y, color=[0,0,0], marker='.', linestyle='', alpha=1, label='Stable')

x = list(map(lambda i: Incl(*Trojans[i].w0)*180/pi, uL5))
y = list(map(lambda i: migration[i][1]*180/pi, uL5))
plt.plot(x, y, color='r', marker='.', linestyle='', alpha=1, label='Unstable')

plt.ylabel(r"Inclination $I'$ after migration °()")
plt.xlabel(r'Inclination $I$ before migration °()')

plt.plot([0,200], [0,200], 'b', label='Reference line $I$=$I\'$')
plt.xlim(0,50)
plt.ylim(0,90)
plt.legend()
plt.savefig('saveL5withname.pdf', format='pdf')

plt.show()




# The errors could occur when numpy is unable to allocate memory during a parallel run for
    the high
# numbers that might occur for unstable Trojans. For the tables, they could be considered as
    unstable, but for
# The inclination graphs they are omitted since the final inclination might not be correct.
for i in range(len(normal)):
    if normal[i][0] == 'error':
        normal[i][0] = 'False'

for i in range(len(migration)):
    if migration[i][0] == 'error':
        migration[i][0] = 'False'


```

```
print(result_num_mat(normal, migration))


minmaxlist = eval(open('10kyminmaxr.txt', 'r').read())
minmaxlisttheta = eval(open('maxandminanglesallTrojans(eccignored)10000Y', 'r').read())

# Establishing the differences between the L4 and L5 copies
def findcopy(i):
    if i<12555:
        return i+12555
    if 12555<=i<12555*2:
        return i-12555
    if 2*12555<=i<12555*3:
        return i+12555
    else:
        return i-12555


ls1 = [i for i in L4 if migration[findcopy(i)][0]!='L5']
ls2 = [i for i in L5 if migration[findcopy(i)][0]!='L4']


# Inclination plots L5 (adjust for zoomed version)
x = list(map(lambda i: abs(minmaxlisttheta[i][1]-minmaxlisttheta[i][0])*180/pi, ls1))
y = list(map(lambda i: Incl(*Trojans[i].w0)*180/pi, ls1))
plt.plot(x, y, color='b', marker='.', linestyle='', alpha=1, label='Stable in L4, unstable in
     L5')

x = list(map(lambda i: abs(minmaxlisttheta[i][1]-minmaxlisttheta[i][0])*180/pi, ls2))
y = list(map(lambda i: Incl(*Trojans[i].w0)*180/pi, ls2))
plt.plot(x, y, color='r', marker='.', linestyle='', alpha=1, label='Stable in L5, unstable in
     L4')

plt.ylabel(r"Resonant angle $\Delta\theta$ before migration °()")
plt.xlabel(r'Inclination $I$ before migration °()')

plt.xlim(0,180)
plt.ylim(0,50)
plt.legend()
plt.savefig('differencesbyinitialconditions.pdf', format='pdf')
```

E.6. Stability MYI

The code below is used to make a graph of the stability region of the MYI.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from math import cbrt

def f(x,y):
    z = x+y*1j
    return np.abs(1+z+z**2/2+z**3/6+z**4/24+(1-2*cbrt(2))/(96-48*cbrt(2))**2*z**5-1*cbrt(2)
        /(96-48*cbrt(2))**3*z**6-1*cbrt(2)/(384-192*cbrt(2))**4*z**7)

def xaxis(x,y):
    return x

def yaxis(x,y):
    return y

steps = 2000
bound = 4
stepsize = bound/steps

d = np.linspace(-bound,bound,steps)
x,y = np.meshgrid(d,d)
z = (f(x,y)<=1.0).astype(float)


```

```
'''
cmap = matplotlib.colors.ListedColormap(["w",(0.8,0.8,0.8)])
plt.contourf(x,y,z, cmap=cmap, extent=(-3,1,-4,4))
#d = np.linspace(-4,4,20)
w = (f(x,y)<=1.0).astype(float)
cmap = matplotlib.colors.ListedColormap(["k","k"])

axiscolor = matplotlib.colors.ListedColormap([(0.5,0.5,0.5)])


axis = ((xaxis(x,y)<2*stepsize) & (xaxis(x,y)>-2*stepsize)).astype(float)
plt.contour(x,y, axis, cmap=axiscolor, alpha=0.01)
axis = ((yaxis(x,y)<2*stepsize) & (yaxis(x,y)>-2*stepsize)).astype(float)
plt.contour(x,y, axis, cmap=axiscolor, alpha=0.01)

plt.contour(x,y,w, cmap=cmap)

plt.text(0, bound*0.9, 'Im($\lambda$d$t$)', fontdict=None)
plt.text(bound*0.75, -0.1*bound, 'Re($\lambda$d$t$)', fontdict=None)

#plt.savefig('MYIstabilityregion.pdf')

plt.clf()
'''

def halfcircle(x,y):
    return x**2+y**2

radius = 2.8
z = (2*(f(x,y)<=1) + ((halfcircle(x,y)<=2.5**2) & (x<=0))+4*(f(x,y)>1)).astype(float)
cmap = matplotlib.colors.ListedColormap([(0.8,0.8,0.8), (0,0,1), "w"])

plt.contourf(x,y,z, cmap=cmap, extent=(-3,1,-4,4))
#d = np.linspace(-4,4,20)
w = (f(x,y)<=1.0).astype(float)
cmap = matplotlib.colors.ListedColormap(["k","k"])

axiscolor = matplotlib.colors.ListedColormap([(0.5,0.5,0.5)])


axis = ((xaxis(x,y)<2*stepsize) & (xaxis(x,y)>-2*stepsize)).astype(float)
plt.contour(x,y, axis, cmap=axiscolor, alpha=0.01)
axis = ((yaxis(x,y)<2*stepsize) & (yaxis(x,y)>-2*stepsize)).astype(float)
plt.contour(x,y, axis, cmap=axiscolor, alpha=0.01)

plt.contour(x,y,w, cmap=cmap)

plt.text(0, bound*0.9, 'Im($\lambda$d$t$)', fontdict=None)
plt.text(bound*0.75, -0.1*bound, 'Re($\lambda$d$t$)', fontdict=None)
plt.plot([0],[0], 'b', label='$\lambda$d$t\leq2.5$' )
plt.legend()

plt.savefig('MYIstabilityregion_sufficient.pdf')




plt.show()
```

E.7. Distance function plots

The code below is used to plot the distance functions used to describe the migration of Jupiter.

```
import numpy as np
import matplotlib.pyplot as plt
from Jupiter_physical_constants import * # All constants are in one file for easy acces


t = np.array([t for t in range(10**6)])/10
t[0]=0.0000001
```

```python

# Constant migration
def constantmig(t, Rinit, T, dx):
    if t<T:
        return Rinit + dx*t/T
    else:
        return Rinit+dx

# Migration as in the research of Li and described by Malhorta
def LiR(Rinit, t, tau, dx):
    return Rinit + (1-np.exp(-t/tau))*dx

# Smoother start of the migration
def smoothR(Rinit, t, tau, dx):
    return Rinit + np.exp(-tau/t)

R = list(map(lambda t: constantmig(t, r_Lgr/AU, 3333, 1), t))

plt.plot(t,R, label='Constant migration rate')
plt.plot(t,LiR(r_Lgr/AU,t,1000,1), label='$R_{init}+\Delta R(1-\exp{(-t/\\tau)})$')
#plt.plot(t,smoothR(r_Lgr/AU,t,150,1), label='$R_{init}+\Delta R\exp{(-\\tau_2/t)}$')
plt.xlim(0,10000)
plt.xlabel('Time (year)')
plt.ylabel(r'Jupiters distance $R_J$ to the sun (AU)')
plt.legend()
plt.savefig('RLicomp.pdf', format='pdf')

plt.show()
```