The Trojan Asteroids

Part II Physics Computing Project, Easter 2017

Abstract

Using Python, the equations of motion for a Sun, Jupiter and asteroid system were solved numerically with an adaptive Runge-Kutta technique. It was then shown that asteroids at the 'Greek' and 'Trojan' Lagrange points remained stable over many orbits of Jupiter around the Sun. Next, it was shown that varying the initial positions of the asteroids slightly could still result in stable orbits over thousands of years, and a plot of the initial positions leading to stable orbits was made. The program was then run for different planetary masses, and for a range of stable orbits the range of wander was found to vary as $\mathrm{M}^{-0.5}$. For very small or large masses, outside of a few orders of magnitude of the mass of Jupiter, no particular relationship was found.

1 Introduction

The problem of three masses interacting via the classical laws of gravitation and Newtonian mechanics is commonly referred to as the three-body problem. Since in this problem the mass of the asteroid is assumed negligible, the Sun-Jupiter-asteroid system can be reduced to a circularly restricted three-body problem.[1] In a rotating reference frame for which the two massive bodies are stationary, the asteroid can be seen to remain stationary, orbit, or possibly escape the system. We aim to find the equations of motion for the asteroid and then solve numerically using the Runge-Kutta technique.

When a small object is placed at one of the 5 Lagrangian points in a non-rotating frame, the force of gravity from the two massive bodies on the object is equal to the centripetal force required to orbit with them. The Greek and Trojan Lagrange points lie $\pi/3$ radians from each mass with respect to the Sun-Jupiter axis, each forming an equilateral triangle with the asteroid at the third vertex (see Figure 1). These two positions are stable, so objects can orbit them in the rotating frame.

Lagrange points are of interest due to their potential use in space exploration and habitation. Satellites have been placed at the on-axis Lagrange points (known as L1, L2 and L3). NASA have considered 'parking' spaceships at the Lagrange point beyond the

Moon (L2) in an Earth-Moon system, in order to remotely operate robots on the lunar far side.[2] Habitations at the Greek and Trojan positions (L4 and L5) could be employed in the Earth-Sun system as waypoints in space travel.[3]

The aim of this project is to present the results of the program written to solve the equations of motion. It starts off by testing stable orbits in the Sun-Jupiter-asteroid system, then moves on to varying the initial conditions around the points of stability. Finally a relationship is derived linking the range of wander and planetary mass. The theoretical background is outlined in Section 2. Section 3 gives a summary of the computational methods used, while Section 4 details the implementation. Section 5 includes the discussion of results, errors and analysis. The overall conclusions of the project are presented in Section 6, and the code is included in the Appendix (Section 7).

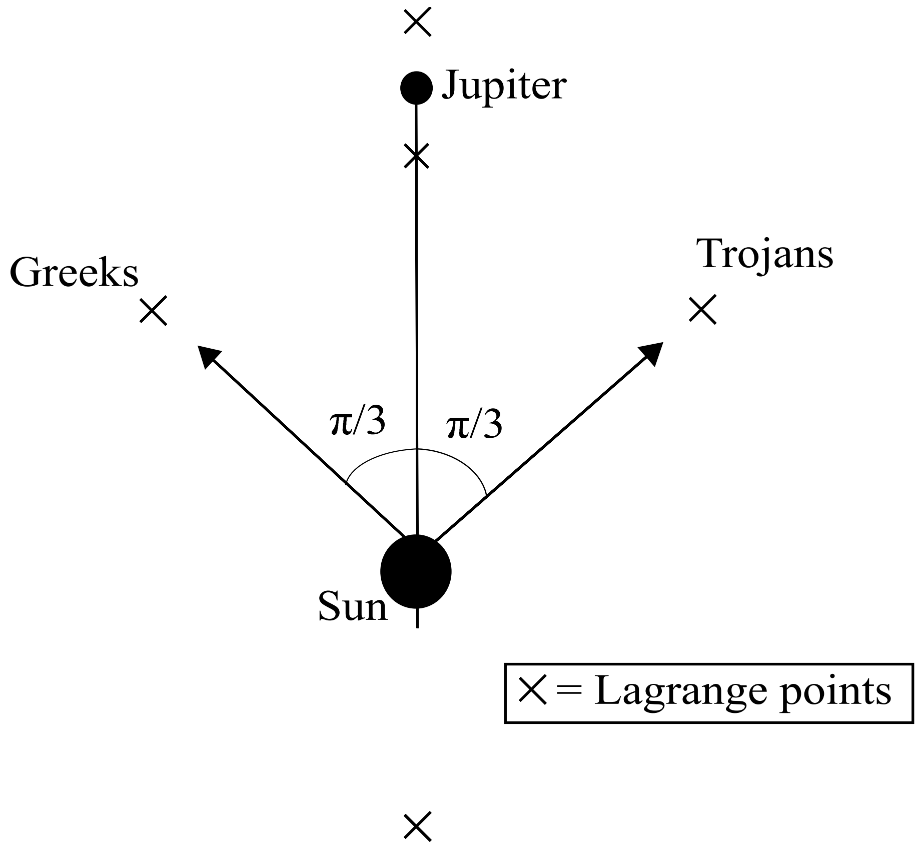

Figure 1: The positions of the 5 Lagrange Points in the Sun-Jupiter system (not to scale). This image is my own and was created using inkscape.

2 Theoretical Background

2.1 Units and Constants

Solar units are used in this problem, with M_s taken to be unity, $M_j = 0.001M_s$. One year is taken to be unity for time, t. Unit distance is one astronomical unit (AU), and the Sun-Jupiter separation is 5.2 AU. Hence the gravitational constant is $G = 4\pi^2$. As previously stated, the asteroid mass is taken as negligible so it does not change the motion of the two-body massive system.

2.2 The Equations of Motion

Figure 1 shows the general set-up but it is easier to find the equations of motion for the asteroid by analysing the system with respect to a frame rotating about the centre of mass of the two massive bodies, shown in Figure 2. It is assumed that the radial distance $R = |\mathbf{r_j}| + |\mathbf{r_s}|$ between Jupiter and the Sun is constant in time, so the orbit is taken to be completely circular. This is an acceptable approximation as the observed orbital eccentricity of Jupiter is $e = 0.048 \approx 0$.[4]

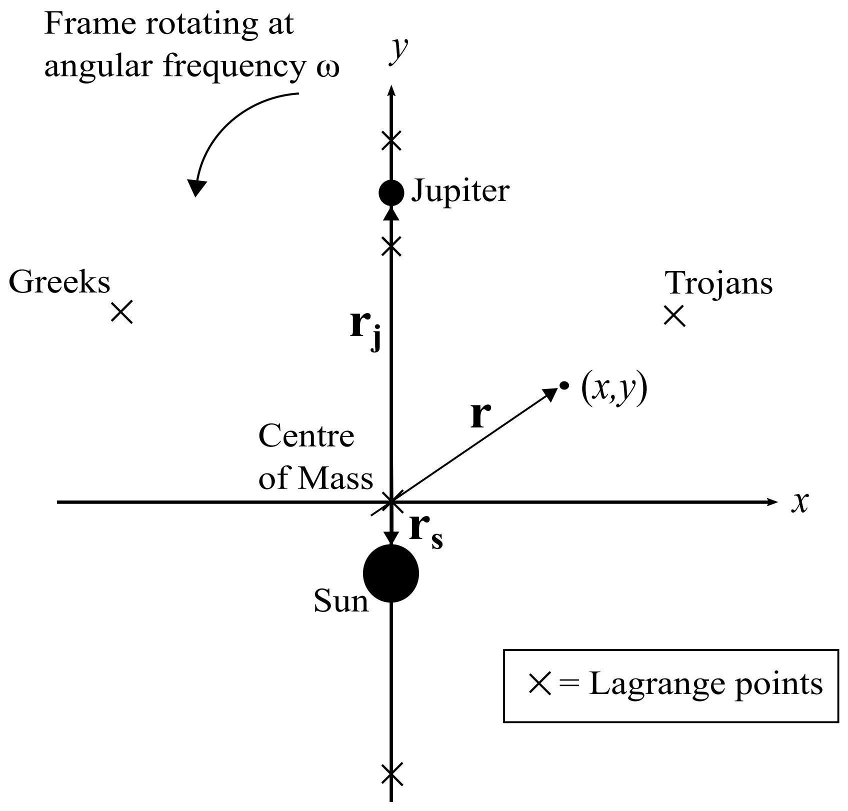

Figure 2: The co-ordinates used in the rotating frame. Note the centre of mass/origin is not a Lagrange point. Also not to scale. This image is my own and was created using inkscape.

It is then simple to derive the x and y components of acceleration, which are the equations of motion, equations (1) and (2). $\mathbf{r}$ is the position of the asteroid from the centre of mass. The angular velocity of the frame is given by $\omega = \sqrt{G(M_s + M_j)/R^3}$. G is the gravitational constant, M_s and M_j are the masses of the Sun and Jupiter respectively. Taking: $\mathbf{r_s} = (0, \frac{M_j R}{M_s + M_J})$ and $\mathbf{r_s} = (0, -\frac{M_s R}{M_s + M_J})$. We have:

$$\ddot{x} = -\frac{GM_s x}{|\mathbf{r} - \mathbf{r_s}|^3} - \frac{GM_j x}{|\mathbf{r} - \mathbf{r_j}|^3} + x\omega^2 + 2\omega\dot{y} \tag{1}$$

$$\ddot{y} = -\frac{GM_s(y + r_s)}{|\mathbf{r} - \mathbf{r_s}|^3} - \frac{GM_j(y - r_j)}{|\mathbf{r} - \mathbf{r_j}|^3} + y\omega^2 - 2\omega\dot{x} \tag{2}$$

The equations of motion are found by considering the components of the gravitational force from each body, in addition to the two fictitious forces - the centrifugal force $\propto r\omega^2$ and the Coriolis force $\propto \omega\dot{r}$.

3 Computational Analysis

3.1 Analysis of computational aspects

The equations of motion are solved with a scipy.integrate.ode integrator. The type of integrator can be selected, and a real-valued variable-coefficient ODE solver was tested for consistency alongside 4th and 8th order Runge-Kutta methods. The respective computational errors of the 4th and 8th order Runge-Kutta methods can be seen in Figures 3 and 4 for orbits on the Trojan Lagrange point. A while loop runs the integration for a specified time and over a specified step size to output the path of the asteroid. The respective errors for each regime are presented in Table 1. Throughout the project, unless otherwise stated, the 8th order Runge-Kutta method was adopted as the absolute error was smaller than that of the 4th order method. Although the real-valued ODE solver had a lower error, this project specified the use of a Runge-Kutta method. It was useful to be able to check everything was working, however.

Integrator	Absolute error	Reference
Real-valued ODE solver	$\sim 10^{-12}$	'vode'
4th Order Runge-Kutta	$\sim 10^{-10}$	'dopri5'
8th Order Runge-Kutta	$\sim 10^{-11}$	'dop853'

Table 1: Computational erros for each integration regime for a 5000 year orbit about the Trojan Lagrange point. Errors given in AU

3.2 Range of Wander

Still working in the rotating frame, the range of wander was calculated by taking the largest and smallest values of the x and y components of the orbit and taking an average.

For a bound orbit this corresponds to good estimate of the average of the leading and trailing groups. When the orbit is unbound this range of wander calculation is not very useful and will be related primarily to how long the program has been run for. Hence to analyse this properly we need to be able to compute the initials positions around the Lagrange points which lead to stable orbits.

3.3 Conditions for a Stable Bound Orbit

We define a function which checks if the orbit is bound or unbound. It is simply an 'if' statement which checks how far away from the centre of the system the asteroid travels. If the asteroid escapes from the system i.e. if the x or y values are greater or less than a certain cutoff point, then the function returns a zero value. Altering these cutoff values arbitrarily did not change the results very much because unbound orbits would quickly escape to infinity. A section of the code produces a plot of the stable initial conditions by looping over many thousands of initial values of x and y.

4 Implementation

The specifics of the code should hopefully be clear enough from the annotations included in the .py script. See the Appendix (Section 7) for the full program listing.

4.1 Approach to Implementing the Algorithms

Once the adaptive Runge-Kutta method lines of code used to solve the equations of motion were written and shown to work, the rest of the project followed reasonably simply. The integrator was included in a series of loops to produce the plots required to solve each specific part of the problem, and the .append command was used extensively to create the arrays of data to be plotted. Pylab was used to create the plots, and these are included in the single script.

When running the script, I recommend not running 'part 2' and 'part 3' together, and to ensure the time specified is appropriate to which plot is trying to be generated to ensure computation times are not excessive.

4.2 Energy Calculation

In order to test the workings of the code, it is possible to plot the energy evolution over time for an orbit. The conserved quantity U is given by:

$$U = \frac{1}{2}\dot{x}^2 + \frac{1}{2}\dot{y}^2 - \frac{1}{2}(\omega x)^2 - \frac{1}{2}(\omega y)^2 - \frac{GM_s x}{|\mathbf{r} - \mathbf{r_s}|} - \frac{GM_j x}{|\mathbf{r} - \mathbf{r_j}|} \tag{3}$$

4.3 Performance

Most of the program was very quick to execute, and the errors on the orbits were very low as shown in the previous sections. However, looping over thousands of initial asteroid positions, each time solving the equations of motion, proved sluggish and, depending on the time parameter chosen, took anywhere between $1-30$ minutes. If we were to consider errors for this part of the calculation, they would be significantly higher as we can only consider a finite number of initial conditions, when there are theoretically an infinite number of positions which will lead to bound orbits. It was found that no particular advantages in terms of the results seen were gained for choosing the conditions which resulted in longer computation times.

5 Results and Analysis

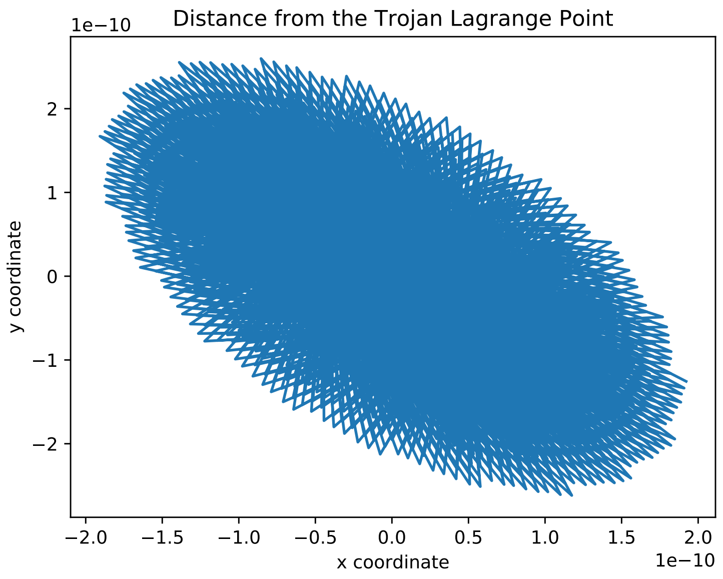

Figure 3: The path of the orbit about the Trojan Lagrange point for t = 5000 years using the 4th Order Runge-Kutta integration method.

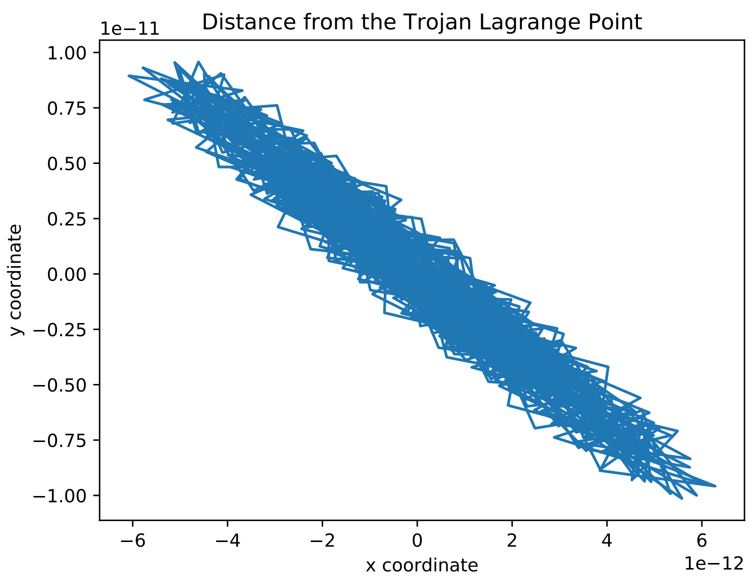

Figure 4: The path of the orbit about the Trojan Lagrange point for t = 5000 years using the 8th Order Runge-Kutta integration method. Initial conditions $(x_{\text{lagrange}}, y_{\text{lagrange}})$

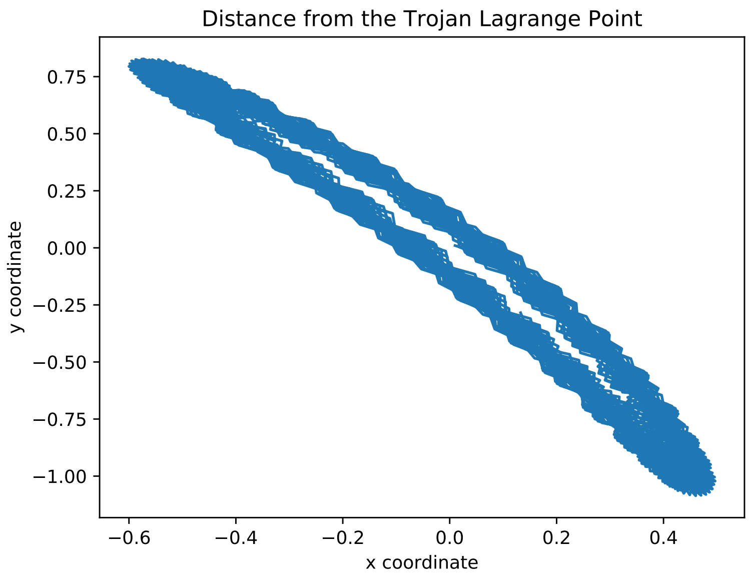

Figure 5: The path of the orbit about the Trojan Lagrange point for t = 5000 years. Initial conditions $(x_{\text{lagrange}} + 0.01, y_{\text{lagrange}} + 0.01)$

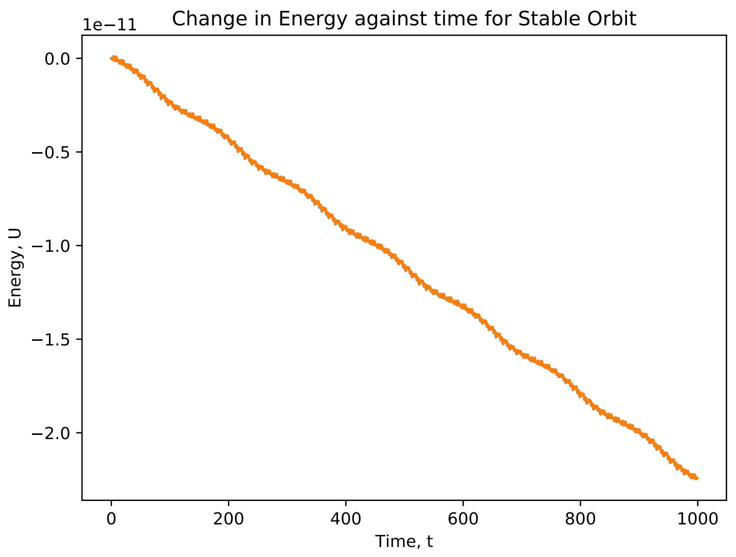

Figure 6: The change in Energy U (as defined in Equation (3)) with time for 1000 years for the stable orbit in Figure 5.

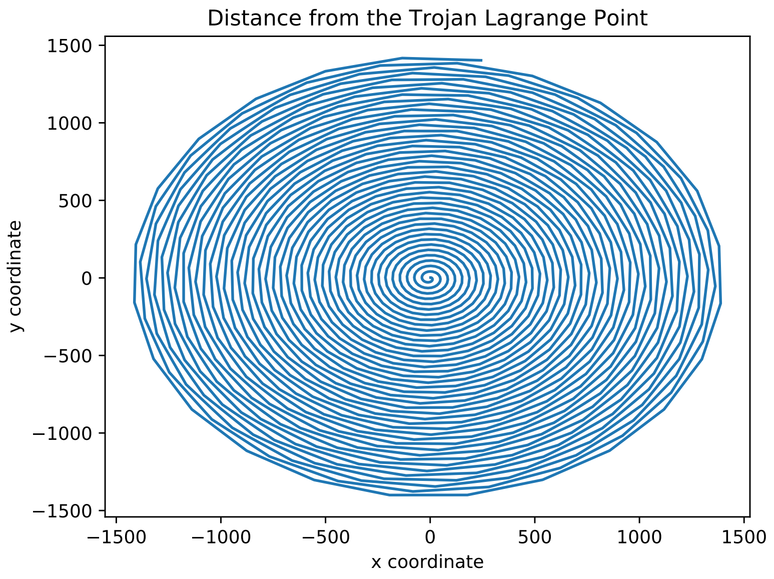

Figure 7: The path of an unstable orbit about the Trojan Lagrange point for t = 500 years. Initial conditions $(x_{\text{lagrange}} + 2.00, y_{\text{lagrange}} + 2.00)$.

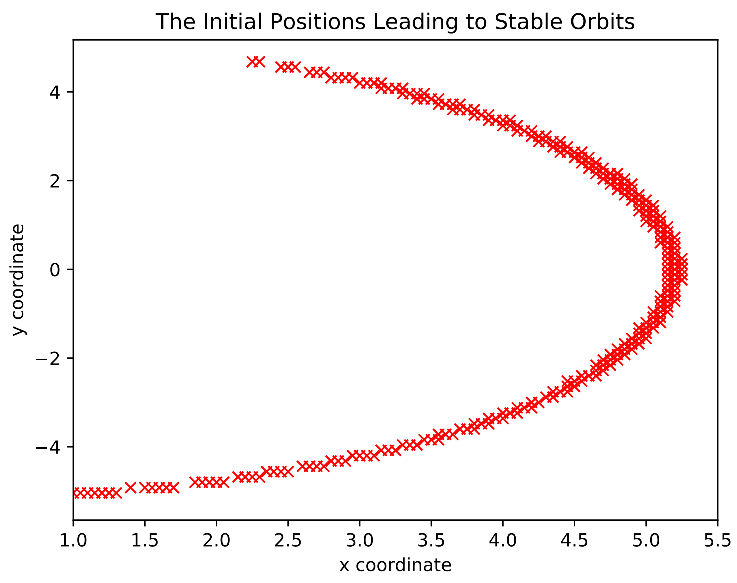

Figure 8: A plot of the initial positions in our samples of x and y co-ordinates leading to bound orbits for the Sun-Jupiter system.

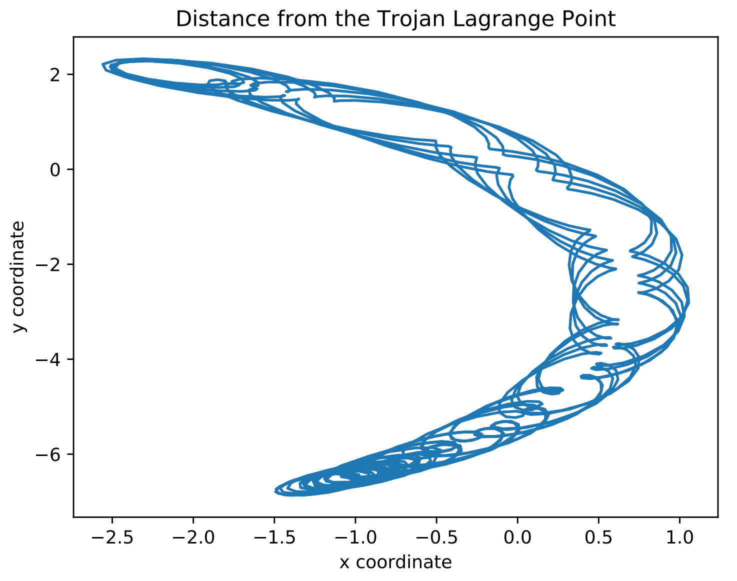

Figure 9: A plot of an orbit with the initial conditions $(5.25, 0)$ as found by observing the extremal point in Figure 8. Notice how its shape and range of wander is comparable to the shape of stable initial conditions in Figure 8.

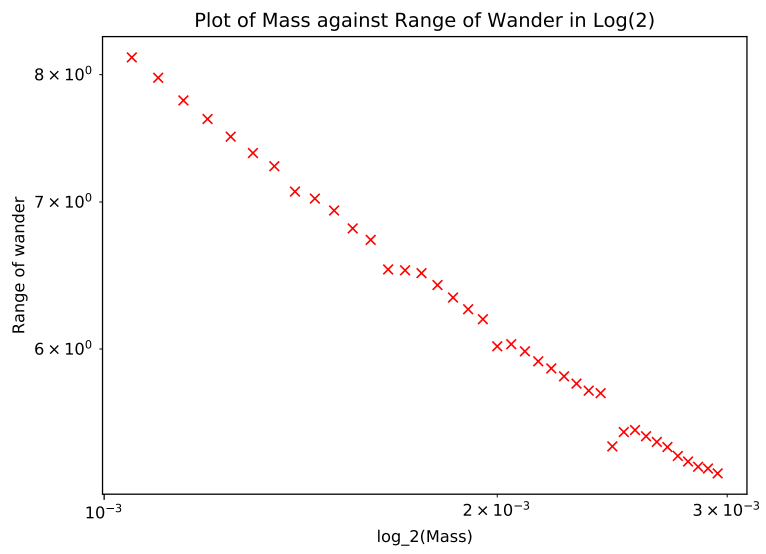

Figure 10: Range of Wander against Mass, M for $0.001 < M < 0.003$. Initial conditions $(1.01x_{\text{lagrange}}, 1.01y_{\text{lagrange}})$. i.e. a small radial displacement from the Lagrange point.

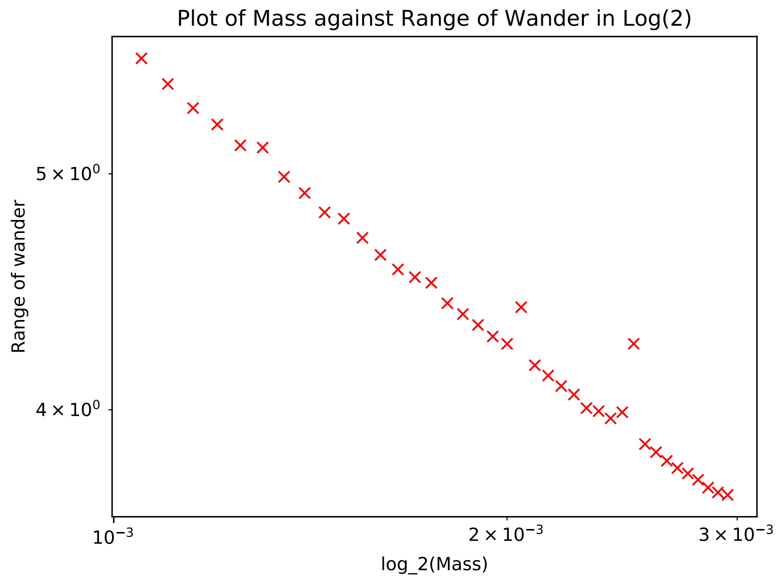

Figure 11: Range of Wander against Mass, M_j for $0.001 < M_j < 0.003$. This time the initial conditions are $(0.99x_{\text{lagrange}}, 0.99y_{\text{lagrange}})$.

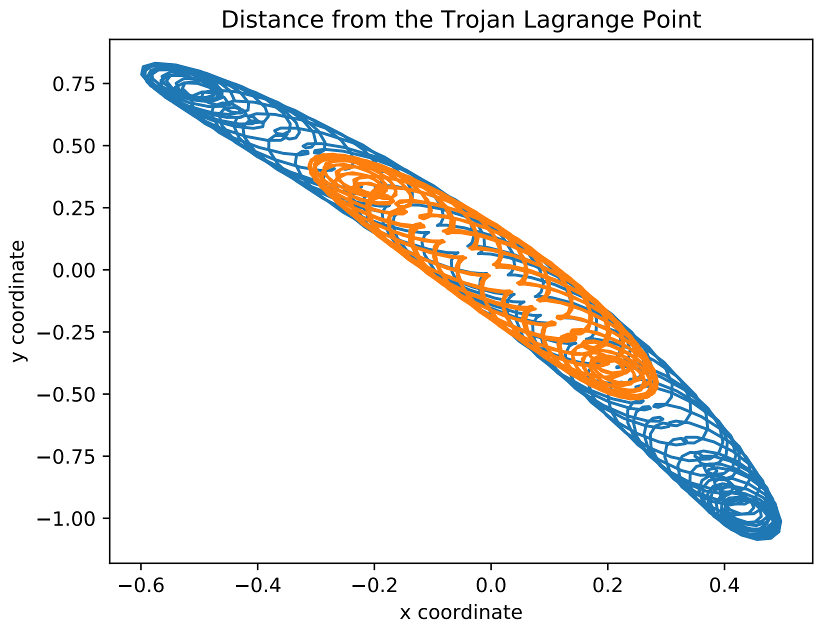

Figure 12: Distance from the Trojan Lagrange Point for $M_j = 0.001$ (blue) and $M_j = 0.005$ (orange), qualitatively agreeing with the relationship shown in Figures 7 and 8.

5.1 Preliminary Testing

The first thing to test was if the integration of the equations of motion produced sensible solutions. The most obvious test was for an asteroid starting at a Lagrange point. Figures 3 and 4 show the distance from the Trojan Lagrange point for an orbit over 5000 years (many hundred orbits of the Sun-Jupiter sytem). As expected, only small computational error deviations from the starting points are observed.

Next, the initial starting position were varied around the Lagrange point, and stable orbits were still observed. An example is shown in Figure 5. An example of an unstable orbit which was started significantly further away from the a Lagrange point is shown in Figure 7.

Finally, just to ensure everything was in order, the change in the constant U (defined by Equation (3)) was plotted against time and the small deviation shown in Figure 6 is within the order of the expected computational error.

Many different initial conditions were experimented with and the findings above were found to hold for all the positions tested. In addition, the initial velocities were changed from zero and both bound and unbound orbits were observed. At this point we can be confident that the basic code to solve the equations of motion and plot paths of orbits works.

5.2 Stable Initial Positions

Moving onto Part 2 of the code, the plot of stable initial positions for an asteroid placed in the Sun-Jupiter system is shown in Figure 8. This plot is useful in allowing us to plot stable orbits in interesting positions, and derive their range of wander for example. An example of an orbit which was at the extremal x value of the plot in Figure 8 is shown in Figure 9.

5.3 Range of Wander

The range of wander for a small initial radial displacement of an asteroid for a range of masses is shown in Figures 10 and 11. Plotted on a $\log_2$ scale the relationship is linear for masses of the 'Jupiter' planet between $0.001M_S$ and $0.003M_S$, suggesting a $M^{-0.5}$ relationship. For higher masses at this initial displacement from the Lagrange point, as well as for very tiny masses, the relationship did not hold as the orbits became unbound. This qualitative observation is in line with what might be expected, and agrees with observations made by Fleming and Hamilton.[5]

5.4 Analysis of Results

As the mass of the Jupiter planet is increased, it has been shown that the range of wander for a small radial displacement falls off as approximately $M^{-0.5}$ for masses within roughly an order of magnitude of M_j. This is also seen by observing the orbits (Figure 12). However, as the mass is increased significantly beyond $0.001M_s$, the stability of the orbits start to decrease and many become unbound, making deriving any kind of quantitive relationship very difficult. Very small masses also do no obey the derived relationship. Neither of these are surprising results as one can imagine the stability of the orbits falling away at extremal masses.

Also interesting is the comparison of Figures 8 and 9 which adopt quite similar shapes and scales, with the range of wander of the orbit covering the same kind of regions as the stable initial conditions. This is not surprising and is easy to imagine in the low orbit velocity regime.

6 Conclusion

It has been shown that the adaptive Runge-Kutta progam written in Python to solve the equations of motion is effective for modelling the orbits of asteroids placed at and around the stable Lagrange points in the Sun-Jupiter system, and other systems of comparable masses. It has been demonstrated that asteroids will stay fixed at Lagrange points over many orbits of Jupiter around the Sun, and that asteroids can follow the paths of bound orbits when the initial positions are varied. A plot of the stable initial conditions has been made, and a relationship established between the mass of the Jupiter planet and the range of wander, which falls off as $M^{-0.5}$ for masses comparable to the mass of Jupiter.

Potential improvements that could be made to this investigation include optimising the code, especially to reduce computation time for deriving stable initial conditions, which could allow for a more thorough investigation into the range of wander against mass calculation (i.e. by looping over a third dimension, M to produce a 3-dimensional landscape plot of stable conditions), as it is possible here that the $M^{-0.5}$ relationship holds only in our very limited situations for a small range of masses and a small radial displacement from the Lagrange point. In addition, varying the initial velocity of the asteroid, which have been kept as zero for this investigation, could have interesting applications for example in gravitational slingshotting. Obvious improvements such as making the model more realistic by considering the mass of the asteroids or the eccentricity of Jupiter's orbit could also be implemented, albeit at the cost of more complicated computational elements, and potentially a longer run time.

References

1. http://scienceworld.wolfram.com/physics/RestrictedThree-BodyProblem.html
2. http://www.space.com/14518-nasa-moon-deep-space-station-astronauts.html
3. http://en.wikipedia.org/wiki/Lagrange_point_colonization
4. http://hyperphysics.phy-astr.gsu.edu/hbase/Solar/soldata2.html
5. H. J. Fleming & D. P. Hamilton, *On The origin of the Trojan asteroids: Effects of Jupiter's mass accretion and radial migration*, 1999 https://arxiv.org/pdf/astro-ph/0007296.pdf

7 Appendix

```
import numpy as np
from scipy import integrate
from scipy.integrate import ode
import matplotlib.pyplot as plt
from pylab import genfromtxt
import pylab
import matplotlib.axes as ax


# PART 1: defining values and solving for one asteroid orbit

# y = [y,dy/dx]
G = 4*((np.pi)**2)
m_s = 1
m_j = 0.001
R = 5.2
omega = (2*np.pi*np.sqrt(m_s + m_j))/pow(R,1.5)
h = [G, m_s, m_j, R, omega] # parameters
```

```
def derivatives2(t,y,h):
    rj = h[1]*h[3]/(h[1]+h[2]) # distance from jupiter to centre of mass
    rs = h[2]*h[3]/(h[1]+h[2]) # distance from sun to centre of mass
    ra = np.sqrt(y[0]**2+(y[1]-rj)**2) # asteroid-jupiter distance
    rb = np.sqrt(y[0]**2+(y[1]+rs)**2) # asteroid-sun distance
    # Equations of Motion
    f =[y[2], y[3], -h[0]*h[2]*y[0]/pow(ra,3)-h[0]*h[1]*y[0]/pow(rb,3) +
        y[0]*pow(h[4],2)+2*h[4]*y[3], -h[0]*h[2]*(y[1]-rj)/pow(ra,
        3)-h[0]*h[1]*(y[1]+rs)/pow(rb, 3) + y[1]*pow(h[4],2) - 2*h[4]*y[2]]
    return f


x_lagrange=h[3]*np.sin(np.pi/3) # x coordinate of Trojan Lagrange point
#x_lagrange=-h[3]*np.sin(np.pi/3) # x coordinate of Greek Lagrange point
y_lagrange=(h[3]*np.cos(np.pi/3)) -(h[3]*h[2])/(h[1]+h[2]) # y coordinate of
    Trojan/Greek Lagrange point
y0 = [x_lagrange+ 0.01, y_lagrange +0.01, 0, 0] # y0 = [x, y, vel(x), vel(y)]
time = 100
steps = 100

t = np.linspace(0,time,steps)
dt = time/steps
sol = np.array([np.array(y0)])
# method = "vode" # Real-valued Variable-coefficient Ordinary Differential
    Equation solver
# method = "dopri5" # Runge-Kutta Method of Order 4
method = "dop853" # Runge-Kutta Method of Order 8
integrator = integrate.ode(derivatives2).set_integrator(method)
integrator.set_initial_value(y0,0.0)
integrator.set_f_params((h))
while integrator.successful() and integrator.t<time:
    sol = np.append(sol,[integrator.integrate(integrator.t+dt)],axis=0)


#pylab.plot(sol[:,0]-x_lagrange,sol[:,1]-y_lagrange)
#pylab.title("Distance from the Trojan Lagrange Point")
#pylab.xlabel("x coordinate")
#pylab.ylabel("y coordinate")


def U(i):
    rj=h[1]*h[3]/(h[1]+h[2])
    rs=h[2]*h[3]/(h[1]+h[2])
    ra=np.sqrt(pow(sol[i,0],2)+pow((sol[i,1]-rj),2))
    rb=np.sqrt(pow(sol[i,0],2)+pow((sol[i,1]+rs),2))
    return
        0.5*pow(sol[i,2],2)+0.5*pow(sol[i,3],2)-0.5*pow((h[4]*sol[i,0]),2)-0.5*pow((h[4]*sol[i
        - h[0]*h[2]/ra - h[0]*h[1]/rb

enrgy = []

for i in range(0,steps):
    enrgy.append(U(i))

#pylab.plot(enrgy - enrgy[0])
#pylab.title("Change in Energy, U against time for Stable Orbit")
#pylab.xlabel("Time, t")
#pylab.ylabel("Energy, U")


# PART 2: Finding the Initial Conditions which produced Bound Orbits
# This is the slowest part of the progam and I would reccommend running for
    time<100 otherwise computation times can be very long

def condition(y,x):
    z=1
    if (np.any(y > 10) or np.any(y < -10) or np.any(x > 10) or np.any(x < 0)):
        z+=-1
    return z

ite = []
ite2 = []
l1 = []
l2 = []
l3 = []

for k in range(0,100):
    for l in range(0,100):
        y3 = [(0.5+k/20), (-6+0.12*l), 0, 0]
        sol3 = np.array([np.array(y3)])
        # method = "vode"
        # method = "dopri5"
        method = "dop853"
        integrator = integrate.ode(derivatives2).set_integrator(method)
        integrator.set_initial_value(y3,0.0)
        integrator.set_f_params((h))
        while integrator.successful() and integrator.t<time:
            sol3 =
                np.append(sol3,[integrator.integrate(integrator.t+dt)],axis=0)
        ite.append(sol3)
        ite2.append(y3)
for p in range(0,(100*100)):
    x = ite[p][:,0]
    y = ite[p][:,1]
    l1.append(condition(y,x))
    l2.append(ite2[p][0])
    l3.append(ite2[p][1])
    xj = np.multiply(l1,l2)
    yj = np.multiply(l1,l3)

pylab.plot(xj,yj, 'xr')
pylab.title("The Initial Positions Leading to Stable Orbits")
pylab.xlabel("x coordinate")
pylab.ylabel("y coordinate")
pylab.xlim([1,5.5])


# PART 3: Solving for a range of masses

row = []
massJ= []

for i in range(1,40):
    h2 = [G, m_s, 0.00005*i +0.001, R, (2*np.pi*np.sqrt(m_s +
        0.00005*i))/pow(R,1.5)]
    x_lagrange2=h2[3]*np.sin(np.pi/3) # x coordinate of Lagrange point
    y_lagrange2=(h2[3]*np.cos(np.pi/3)) - h2[3]*(h2[2])/(h2[1]+h2[2]) # y
        coordinate of Lagrange point
    y20 = [1.01*x_lagrange2,1.01*y_lagrange2,0,0]
    t = np.linspace(0,time,steps)
    dt = time/steps
    sol2 = np.array([np.array(y20)])
    # method = "vode"
    # method = "dopri5"
    method = "dop853"
    integrator = integrate.ode(derivatives2).set_integrator(method)
    integrator.set_initial_value(y20,0.0)
    integrator.set_f_params((h2))
    while integrator.successful() and integrator.t<time:
        sol2 = np.append(sol2,[integrator.integrate(integrator.t+dt)],axis=0)
    loop = (np.sqrt((max(sol2[:,0]) - min(sol2[:,0]))**2) +
        np.sqrt((max(sol2[:,1]) - min(sol2[:,1]))**2))*0.5
    row.append(loop)
    massJ.append(h2[2])

fig1,ax1 = plt.subplots()
ax1.loglog(massJ, row, 'xr', basex=2)
ax1.set_xscale('log')
pylab.title("Plot of Mass against Range of Wander in Log(2)")
pylab.xlabel("log_2(Mass)")
pylab.ylabel("Range of wander")
```