Computer simulation of Jupiter's Trojan asteroids

April 2015

Abstract

The orbits of asteroids bound to Lagrange points 4 and 5 of the Sun-Jupiter system were simulated in two dimensions. A linear multistep method was used to solve the equation of motion. The stability of the region surrounding each point was assessed by observing how far an asteroid would 'wander' from its starting position. An asteroid placed at the point itself was found to have a range of wander of 0.14 au; it travelled in a 'tadpole' orbit because of the Coriolis effect. The stable region lies along and just outside Jupiter's orbit, with a width on the order of 1 au. A quadratic empirical relation between the planet's mass and asteroid range of wander was derived. For planet/sun mass ratios greater than 0.02 no stability at the Lagrange points was seen. This is consistent with previous analytical results.

1 Introduction

The Greeks and Trojans are two groups of asteroids observed to lead and trail Jupiter's orbit of the Sun (Figure 1). They lie around the L_4 and L_5 Lagrange points respectively and inherit their names from heroes of the mythological war.

In a two body system, the five Lagrange points are those where the gravitational pull of the bodies is precisely equal to the centripetal force required for a third small mass to move with them. That is, a small mass placed exactly at one of the points should remain in stationary equilibrium with respect to the other two bodies. However, examining the effective potential (in the frame rotating about the barycentre — Figure 2) we see that these all occur at maxima and thus are not expected to be stable.

So why, then, do L_4 and L_5 harbour asteroids in the Sun-Jupiter system? The answer is the Coriolis effect. As an object falls away from one of these points, the Coriolis force curves its path. If the ratio of the two large masses is much greater than 0.04 then this deflection is actually significant enough to prevent the object from escaping [1].

The project described in this paper is a numerical simulation of the behaviour of the Trojan asteroids, confirming that they are indeed trapped about the 4th and 5th Lagrange points by the Coriolis effect. We will examine the

Figure 1: The Greek and Trojan asteroids (green). [3]

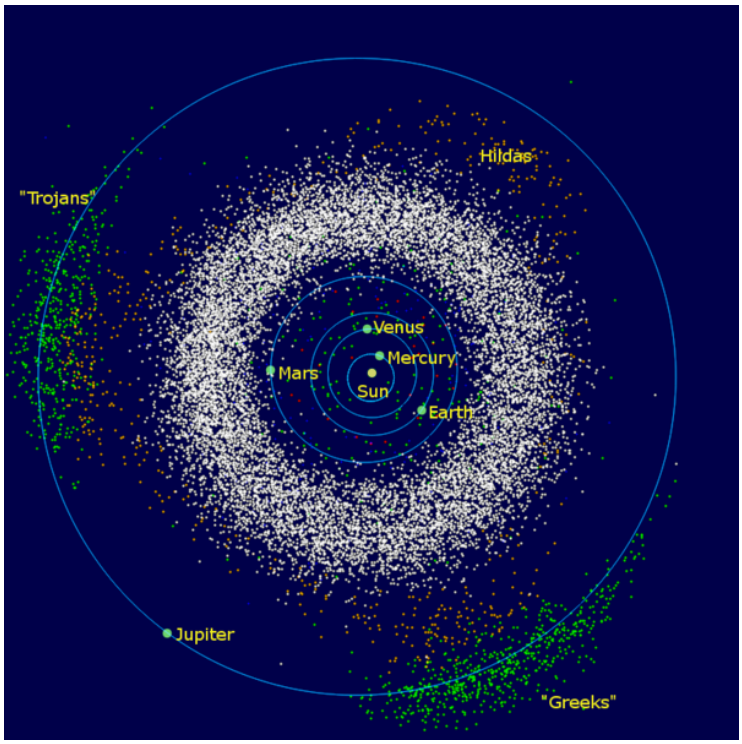


Figure 2: Effective potential (rotating frame) of two bodies in orbit. The five Lagrange points all occur at maxima. [2]

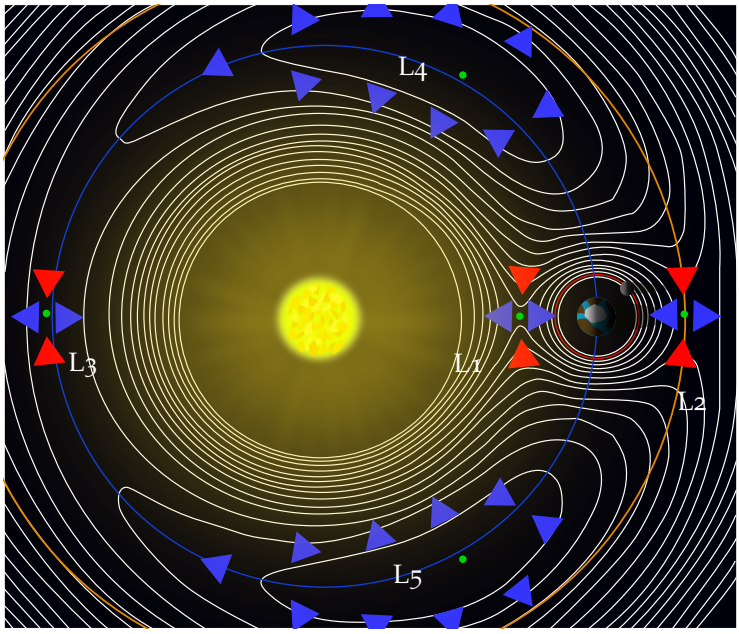


Table 1: Solar system units.

Quantity	Unit
Length	au
Mass	$M_\odot$
Time	yr

range over which they deviate (or 'wander') from the equilibrium point and thus infer the size and shape of the 'stable' region around the Lagrange points. We will also study the relationship between the planet's mass and the asteroids' range of wander.

2 Physical model

From a theoretical standpoint the dynamics of L_4 and L_5 are identical [4], so we need only focus on one of them. To simplify the problem we place the Sun and Jupiter in circular orbits about their barycentre such that they are at rest in our rotating reference frame (angular frequency ω). Their separation is fixed at $R = 5.2\,\text{au}$ with the other orbital parameters derived from this. In the direction perpendicular to the orbital plane, it can be shown that *all* of the Lagrange points are unconditionally stable [1]. Thus we restrict our attention to the two dimensions of the plane itself in order to investigate the particular overall stability of L_4 and L_5.

With these considerations, our asteroids obey the Newtonian equation of motion

$$\ddot{\mathbf{r}} = -\frac{Gm_s}{\left((r_s + r_x)^2 + r_y^2\right)^{\frac{3}{2}}}\begin{pmatrix} r_x + r_s \\ r_y \end{pmatrix} - \frac{Gm_p}{\left((r_p - r_x)^2 + r_y^2\right)^{\frac{3}{2}}}\begin{pmatrix} r_x - r_p \\ ry \end{pmatrix} + 2\omega\begin{pmatrix} \dot{r}_y \\ -\dot{r}_x \end{pmatrix} + \omega^2\mathbf{r} \quad (1)$$

where r_s, m_s, r_p and m_p are the distances from the barycentre and masses of the Sun and Jupiter. The first two terms are the gravitational attraction of the bodies; the last two are the Coriolis and centrifugal forces respectively. The Lagrange point itself is given by $\mathbf{r}_0 = (r_p - R/2, \sqrt{3}R/2)$.

3 Computer implementation

To keep numerical values in the simulation reasonable we work in the 'solar system units' specified in Table 1. This diminishes any potential for arithmetic under/overflows.

The equation of motion (1) may be re-written as four coupled first-order

ordinary differential equations (ODEs):

$$\dot{r}_x = v_x \tag{2a}$$
$$\dot{r}_y = v_y \tag{2b}$$
$$\dot{v}_x = \dot{v}_x(\mathbf{r}, v_y) \tag{2c}$$
$$\dot{v}_y = \dot{v}_y(\mathbf{r}, v_x) \tag{2d}$$

We now have form which can be solved by standard numerical techniques. In particular we employ a linear multistep method[1] using the ODEPACK library routine LSODA [5].

As well as solving for the asteroid's trajectory we also compute its 'wander'. This is the furthest distance it travels from the Lagrange point over the course of the simulation. It gives an indication of how tightly bound the asteroid is, i.e. its stability.

Typically we simulate 500 Jupiter orbits, explicitly evaluating the asteroid position $\mathbf{r}$ at 100 points in each. These parameters were chosen to give acceptable precision whilst maintaining good performance. For the longer experiments, multiple processes (each solving a different set of initial conditions) were run in parallel. In some cases intermediate calculations were saved to disk so that subsequent minor tweaks did not necessarily incur a lengthy re-run.

A number of simple test cases were examined in the process of validating the model and fixing bugs in the implementation. These sometimes consisted of fewer bodies, or included a subset of the forces.

Appendix A contains the full source code in Python 3. The core routines are defined in Listing 1. The others are scripts which perform the actual experiments.

4 Experiments and Discussion

Figure 3 shows the result of running the simulation for 5,000 Jupiter orbits. The asteroid was placed exactly (give or take numerical rounding) on the Lagrange point. The trajectory for the first 12 orbits is detailed in Figure 4. The range of wander in this case was 0.14 au.

As expected the asteroid falls away from the Lagrange point; it is not in stable equilibrium. We see it follow a curved path due to the Coriolis effect, which prevents it from becoming free. The motion appears to contain two major components: a high frequency, small radius oscillation combined with a lower frequency, larger radius oscillation. The superposition gives the characteristic 'tadpole' orbit typical of Trojan objects.

It is important to remember that in the inertial frame these trajectories look far less convoluted. They are akin to an ordinary circular orbit which is being slightly perturbed.

Figure 3: Stability over 5,000 Jupiter orbits. The asteroid's trajectory is denoted by the blue line; it remains confined to a small region around L_4 (marked by a cross). Sun and Jupiter not to scale.

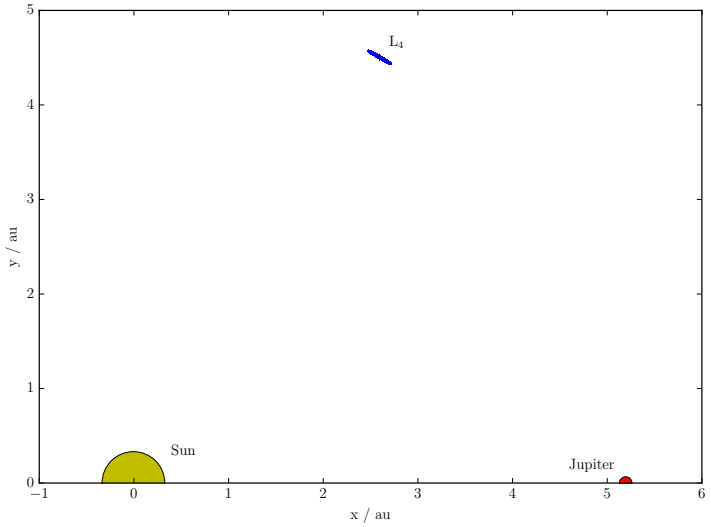


Figure 4: Asteroid trajectory over 12 Jupiter orbits. The starting point, L_4, is marked by a cross.

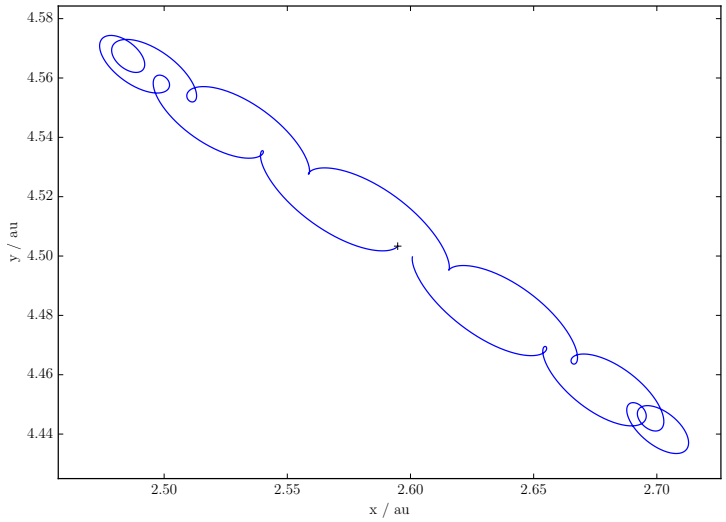
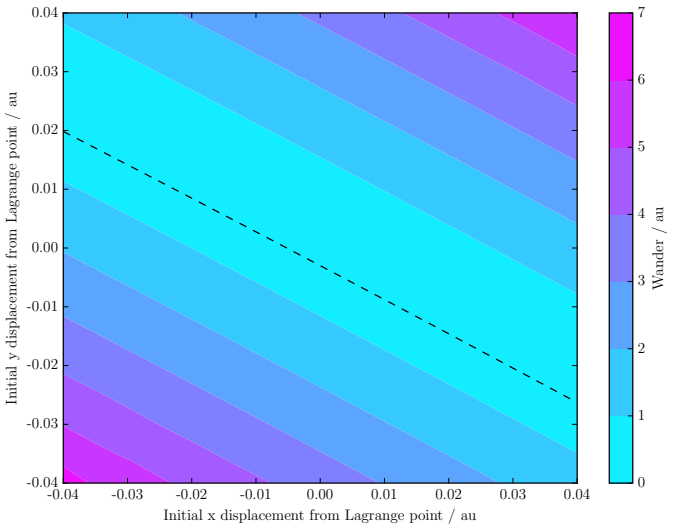


Figure 5: Range of wander as a function of starting position. The dotted line indicates Jupiter's orbit. The most stable region lies along and slightly outside of this.



4.1 Region of stability

In order to investigate the stability of the area around the Lagrange point we run a number of simulations, each using a different asteroid starting point. The range of wander is thus calculated for each position, as shown in Figure 5. This result took 141 s to evaluate on an Intel Core i5 processor; that is 0.6 s per simulation run.

Immediately we notice that the stable region has some shape. Perturbing the initial position along the line of Jupiter's orbit has very little effect, whereas in the perpendicular direction a small offset gives rise to a significant range of wander. Starting the asteroid more than 0.06 au to/from the Sun results in a total loss of stability. The shape here matches up with the asteroids' typical trajectories. Notice also that the most stable region is actually slightly outside of Jupiter's orbit, not at the precise maximum of the effective potential.

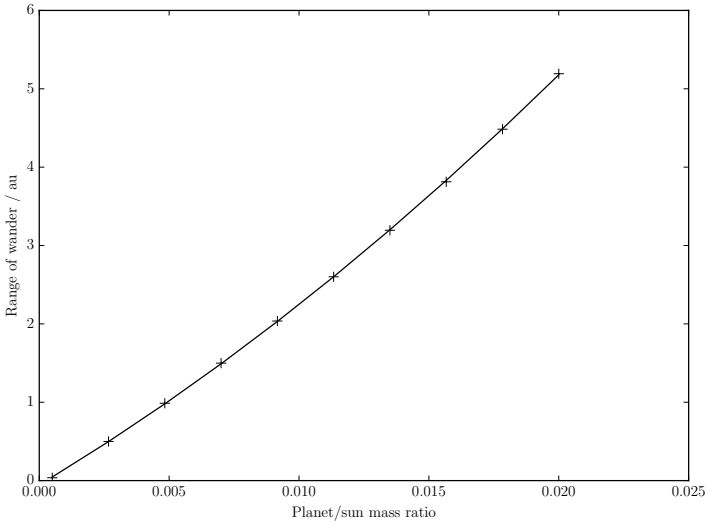
4.2 Planet/sun mass ratio

Ten simulations were run with varying planet masses (all other parameters were kept as for Jupiter). In each an asteroid was placed at the Lagrange point and its range of wander determined after 500 orbits. The resulting trend is displayed in Figure 6.

When the mass ratio exceeded 0.02 the asteroid was no longer bound to the point. This finding is consistent with Greenspan's analytical result that for stability the ratio must be much less than 0.04 [1].

[1] Linear multistep methods keep and use information from previous steps; Runge-Kutta methods simply discard such data.

Figure 6: Relationship between the planet's mass and asteroid range of wander. A least squares fit yields the quadratic $3150m^2 + 199m + 0.05$.



For lower values, a strong relationship is evident. The data are well fitted by a quadratic polynomial with a large linear contribution and negligible constant: $3150x^2 + 199x + 0.05$.

5 Conclusions

The simulation was successful in demonstrating the somewhat unexpected binding of objects to the 4th and 5th Lagrange points over many cycles, due to the Coriolis effect. The range of wander for an asteroid placed on the point was found to be 0.14 au.

The stable region was found to lie along and just outside the line of Jupiter's orbit; it's width is of the order 1 au. This matches our observations of the real distribution of Trojan asteroids (Figure 1). Future work could extend the range of initial conditions covered and fully map out the region.

The range of wander of asteroids depended strongly on the mass of the planet. For planet/sun mass ratios greater than 0.02 no stability was observed. This is consistent with earlier analytical findings. The relationship is almost linear, with a smaller quadratic component. It is given empirically by $3150m^2 + 199m + 0.05$ where m is the ratio.

References

[1] T. Greenspan, 2014. 'Stability of the Lagrange Points, L4 and L5'.

[2] NASA / WMAP Science Team, 2012. 'Lagrange Points of the Sun-Earth system, with Gravity Field'. `http://map.gsfc.nasa.gov/media/990529/index.html` [Retrieved 20 April 2015]

[3] Mdf at *English Wikipedia*, 2011. 'The inner solar system'. `https://commons.wikimedia.org/wiki/File:InnerSolarSystem-en.png` [Retrieved 20 April 2015]

[4] F. Marzari, H. Scholl, C. Murray, C. Lagerkvist, 2002. 'Origin and Evolution of Trojan Asteroids'. *Asteroids III*, 725–738.

[5] A. Hindmarsh, 1983. 'ODEPACK, a systematized collection of ODE solvers'. *IMACS Transactions on Scientific Computation*, 1, 55–64.

A Code

Listing 1: `lagrange.py`. Core functionality.

```
import math

import numpy as np
import scipy.integrate

import matplotlib.pyplot as plt

#########################################################
# user defined quantities

G = 4*math.pi**2 # gravitational constant
ms = 1 # solar mass
mp = 0.001
R = 5.2 # astronomical units
T = R**1.5 # jupiter -> 11.8618 years

precision = 100 # evaluation points per orbit

#########################################################
# derived quantities

rs = R * mp/(ms+mp) # distance from origin to sun
rp = R * ms/(ms+mp)
w = 2*math.pi/T # angular velocity of frame

# co-ordinates of lagrange point
lx = rp-R/2
ly = math.sqrt(3)*R/2
```

```
    ########################################################

31
    # equations of motion
    def derivs (y, t):
34    rx, ry, vx, vy = y

      # for convenience
37    dp3 = ((rp-rx)**2 + ry**2)**1.5
      ds3 = ((rs+rx)**2 + ry**2)**1.5

40    return (vx,
              vy,
              -G * (ms*(rx+rs)/ds3 + mp*(rx-rp)/dp3) + 2*w*vy + rx*w**2,
43            -G * (ms*ry/ds3 + mp*ry/dp3) - 2*w*vx + ry*w**2)

    # calculate a trajectory starting from y0
46  def orbit (y0, orbits):
      t = np.linspace(0, orbits*T, orbits*precision)
      y = scipy.integrate.odeint(derivs, y0, t)
49    return np.transpose(y)

    # find furthest distance travelled from lagrange point
52  def wander(y):
      return max([math.sqrt((rx-lx)**2+(ry-ly)**2) for (rx,ry) in zip(y[0],y[1])])
```

Listing 2: `single.py`. Calculates and plots the trajectory of an asteroid given its initial state.

```
1   from lagrange import *

    import matplotlib.pyplot as plt
4   from matplotlib import rc

    rx0 = lx
7   ry0 = ly
    orbits = 5000

10  ################################

    y0 = (rx0, ry0, 0, 0)
13  y = orbit(y0, orbits)

    print(wander(y))
16

    # latex font rendering
19  plt.rc('font',**{'family':'serif','serif':['mathpazo']})
    plt.rc('text', usetex=True)

22  plt.plot(y[0], y[1], 'b-')

    plt.text(lx+0.1, ly+0.1, r'L$_4$')
```

```
plt.plot(lx, ly, 'k+')

plt.text(-rs+0.4, 0.3, 'Sun')
plt.plot(-rs, 0, 'yo', markersize=50)
plt.text(rp-0.6, 0.15, 'Jupiter')
plt.plot(rp, 0, 'ro', markersize=10)

plt.xlabel('x␣/␣au')
plt.ylabel('y␣/␣au')

plt.tight_layout()
plt.show()
```

Listing 3: `offset.py`. Concurrently evaluates wander for a grid of starting points and saves result to disk.

```
from lagrange import *

import multiprocessing

# our grid of sample starting points has size deltax * deltax, centred
# on the lagrange point
deltax = 0.08
x = np.linspace(-deltax/2, deltax/2, 16)
xx, yy = np.meshgrid(x, x)


if __name__ == "__main__":
  # nested list comprehension; the outermost one we run in parallel
  # note: unfortunately pool.map() cannot take lambda functions
  # equivalent to:
  #  ww = [[wander(orbit((lx+x[0],ly+y[0],0,0))) for x in xx.T] for y in yy]
  def y(y):
    return [wander(orbit((lx+x[0],ly+y[0],0,0), 500)) for x in xx.T]
  pool = multiprocessing.Pool(processes=4) # 4 cores on my laptop
  ww = pool.map(y, yy)

  np.save("ww.npy", ww)
```

Listing 4: `offset_plot.py`. Plots data (in `ww.npy`) produced by `offset.py`.

```
from offset import *

import matplotlib.pyplot as plt
from matplotlib import rc
from matplotlib import cm

ww = np.load("ww.npy")

fig = plt.figure()
ax = fig.add_subplot(111)
plt.rc('font',**{'family':'serif','serif':['mathpazo']})
```

```
plt.rc('text', usetex=True)

contours = ax.contourf(xx, yy, ww, cmap=cm.cool)
cbar = fig.colorbar(contours)
cbar.set_label('Wander / au')

# jupiter's (circular) orbit
cx = np.linspace(-deltax/2, deltax/2, 100)
cy = np.sqrt(rp**2-(cx+lx)**2)-ly
ax.plot(cx, cy, 'k--')

plt.xlabel('Initial x displacement from Lagrange point / au')
plt.ylabel('Initial y displacement from Lagrange point / au')

plt.tight_layout()
plt.savefig('../figures/offset_plot.pdf')
```

Listing 5: `mass.py`. Evaluate wander for a range of planet masses. Perform a quadratic fit to quantify the relationship.

```
from lagrange import *

import matplotlib.pyplot as plt

import scipy.stats

# masses to simulate
mps = np.linspace(0.0005, 0.02, 10)

wanders = []
for mp in mps:
  ############################################################
  # from lagrange.py; need to redefine in this loop
  rs = R * mp/(ms+mp)
  rp = R * ms/(ms+mp)
  w = 2*math.pi/T

  lx = rp-R/2
  ly = math.sqrt(3)*R/2
  ############################################################

  y0 = (lx,ly,0,0)
  # TODO: parallelise this loop
  wanders.append(wander(orbit(y0, 500)))

fit = np.poly1d(np.polyfit(mps,wanders,2)) # quadratic
print(fit)

plt.rc('font',**{'family':'serif','serif':['mathpazo']})
plt.rc('text', usetex=True)

plt.plot(mps, wanders, 'k+', markersize=8)
```

```
33 plt.plot(mps, fit(mps), 'k-')

   plt.xlabel('Planet/sun␣mass␣ratio')
36 plt.ylabel('Range␣of␣wander␣/␣au')

   plt.tight_layout()
39 plt.savefig('../figures/mass.pdf')
```