Interplanetary Trajectory Design for a Jupiter Mission between 2025 to 2040

Loo Ting Tan
University of Colorado Boulder, Boulder, CO 80302, USA

A direct ballistic trajectory from Earth to Jupiter is designed between 2025 to 2040, to serve as a potential follow-up scientific mission after Juno's mission and Europa Clipper's planned launch in 2020. The detailed computation framework is illustrated and the feasibility of the launch and arrival opportunities for Jupiter's mission is obtained and discussed. It is shown that a direct trajectory from Earth to Jupiter is feasible between 2030 and 2032 in 893 days, which is within the time of flight requirement for Europa Clippers (3 years) and the characteristic energy for launch is also within the capability of the planned Space Launch System propulsion technology for Europa Clippers. The porkchop plots and trajectory solutions are presented and also validated.

I. Nomenclature

r	=	magnitude of position vector
v	=	magnitude of velocity vector
θ^*	=	true anomaly of an orbit
$C3$	=	characteristic energy of transfer
e	=	eccentricity
p	=	semi-latus rectum
a	=	semi-major axis
μ	=	gravitational parameter
TOF	=	time of flight
f, g, f', g'	=	Lagrange coefficients

II. Introduction

The exploration of Jupiter is a significant motivation as large planet plays an important role in the planet formation due to their influence on the orbits of other celestial objects within the planetary system. Studying and understanding its origin could potentially allow us to discover how Earth is form and how other giant planets behave around other stars. In addition, Europa, one of the moons of Jupiter, was speculated to have oceans of twice the volume of Earth's ocean beneath their outer ice shell due to certain magnetic characteristics detected by Galileo spacecraft during its 12 close flybys from 1995 to 2003, as well as satellite images showing the lack of craters, hypothesized to be warmer ice rising from below and erasing the craters over time. Water is one of the key ingredients for life, and hence, the mission around Jupiter is important to discover extraterrestrial life and understand our place in the universe.

Figure 1: Europa's ice crust in the Conamara region showing the interplay of surface color with ice structures. (image from NASA)

This research project identifies flight opportunities between 2025 to 2040 using Porkchop plots. Porkchop plots are useful contour plots to aid engineers identify the optimal flight opportunities during mission planning. With Juno planned deorbit in mid-2021 and Europa Clipper estimated launch date in 2020 (potentially up to 2025) and last

approximately 3.5 years (orbit around Jupiter and 45 flybys around Europa), this selected exploration window opens up future scientific experiments opportunities that are not included in the Europa Clipper mission and also follow-up missions post-Clipper's discoveries.

III. Problem Overview

In this project, only a direct trajectory has been assessed and analysis of additional trajectories such as flybys were not prioritized. This is in line with Europa Clipper's planned direct transfer within 3 years. The analysis will approximate the heliocentric orbits to be circular and coplanar among the planets. The feasibility of direct trajectory will be investigated and briefly discussed based on the current propulsion technology available for launch vehicle. The mass of the spacecraft is also assumed insignificant to influence the computations compared to the celestial objects in discussion (Jupiter, Mars, Earth etc.). In addition, the study limits the trajectory design up to the Jupiter's orbit insertion and rendezvous at a desired location. Subsequent orbit or fly-bys around Jupiter for observation will not be investigated. This project assumes two body problem and will not include effects of additional perturbation for initial calculations. In reality, the large gravitational sphere of influence of Jupiter (6% of its semi-major axis around the Sun) is likely to influence the outcome but will be assumed insignificant for preliminary investigation. The feasibility and limitations of the trajectory will be investigated and discussed.

The time system in this research project will utilize the modified Julian Date (MJD), Julian Date -2400000.5
In the consideration of a feasible transfer trajectory, the mission is assumed to be an unmanned mission, which limits the mission planning to a one-way trajectory from Earth to Jupiter as deep space exploration satellite in general deorbit into the mission atmosphere as the mission ends. In addition, there are no feasible launch vehicles to deliver humans to even Mars due to the significant amount of ECLSS and logistics required by the long time of flight.

IV. Methodology

The project implemented a Python 3.7 algorithm to calculate the optimal direct trajectory and to generate the porkchop plots. Several common add-on libraries from poliastro, astroquery, astropy and built-in libraries such as math, numpy and matplotlib are used to simplify the algorithm. Add-on libraries do not come with the standard Python libraries and are required to be installed separately. The numerical codes are included in Appendix.

The mission design involves two maneuvers: (i) spacecraft escapes from the orbit around Earth and (ii) Jupiter orbit insertion that puts the spacecraft in Jupiter's orbit. The mission assumes that mid-course corrections are not required for the preliminary investigation.

The overview of the computation process of the direct trajectory is illustrated in the Fig 2 below.

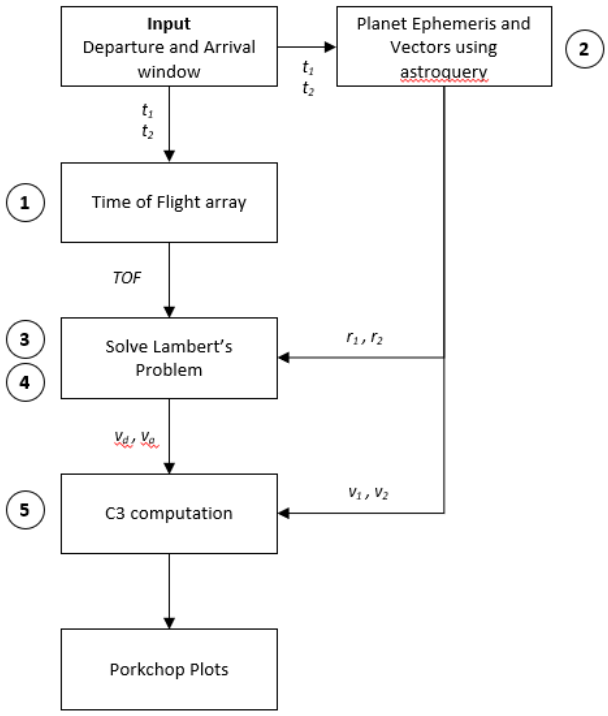

Figure 2: Workflow of the computation of porkchop plots

1) Selection of the trajectory dates (departure, arrival) and time of flight (TOF) arrays are computed based on the trajectory dates

2) Extraction of planet ephemeris and vectors relative to the heliocentric ecliptic frame of reference from JPL Horizon database or JPL ephemeris.

3) Vectors of the launch and target planet are inputs to solve Lambert's Problem

4) Determination of the orbital elements of the transfer orbit (semi-latus rectum, eccentricity, $\Delta\theta$* and f,g functions to obtain velocity at departure from Earth and arrival at Jupiter)

5) Solving of Hyperbolic Excess Velocity to obtain minimum energy (C3) characteristics for departure and arrival

Planet ephemerides and Vectors

Extraction of JPL horizon data can be done using both Astropy.coordinates or Astroquery.jplhorizons python library.

1) Using astropy.coordinates, the function get_body_barycentric_posvel(body, time, ephemeris) could be used to generate the position and velocity vectors with ephemeris. Setting 'jpl' specify explicitly the usage of JPL ephemeris.

   from astropy.coordinates import solar_system_ephemeris
   solar_system_ephemeris.set('jpl')

2) Using Astroquery's module, the inputs to generate simply follows the format used in JPL Horizon's web interface where id refers to the planet or celestial object (399 for Earth, 599 for Jupiter), location refers to the frame of observation (500@10 refers to sun-centered frame), epochs refers to the start, end date and the step size in between (indicated as 10d for 10 days, 30d for 30 days)

   src=Horizons(id='399',location='500@10', epochs={'start':time_start,'stop':time_end,'step':stepsize})

   To obtain planet vectors and ephemerides, we used vectors() and ephemerides() respectively to access the object.

vec_src = src.vectors()
eph_src = src.ephemerides()

The individual variables are simply vec_src[‘*variable name’*] where variable name is x,y,z,vx,vy,vz. Examples are shown below:
r1v= np.array([vec_src['x'][i],vec_src['y'][i],vec_src['z'][i]])
v1iv=np.array([vec_src['vx'][i],vec_src['vy'][i],vec_src['vz'][i]])

Units are expressed in AU and AU/day. Conversion are applied to convert to km and km/s

Time t_1 and t_2 can simply be accessed via the object *vec_src* via *vec_src[‘datetime_jd’]*.

Lambert’s equation
The solution of Lambert’s equation here used follows the expression presented in Curtis [2005] which is similar to Prussing and Conway [1993] where Lagrange coefficients are obtained to determine v_1 using two position vectors given a known TOF. In Curtis [2005], the Lagrange coefficients are further re-expressed using universal anomaly variable χ and the algorithm is presented in the material. In this project, bisection method is used. Derivation is presented below. In addition, the code based on Prussing and Conway [1993] is also included in the Appendix as a check for the solution.

Lagrange f and g coefficient and their derivations:

$$f = 1 - \frac{r_1}{p}(1 - cos\Delta\theta^*)$$

$$g = \frac{r_1 r_2}{\sqrt{\mu p}}(sin\Delta\theta^*)$$

$$\dot{f} = \sqrt{\frac{\mu}{p}}\tan\left(\frac{\Delta\theta^*}{2}\right)\left(\frac{1 - cos\Delta\theta^*}{p} - \frac{1}{r_1} - \frac{1}{r_2}\right)$$

$$\dot{g} = 1 - \frac{r_1}{p}(1 - cos\Delta\theta^*)$$

Using Stumpff functions C(z) and S(z) where $z = \frac{\chi^2}{a}$, we can re-expressed the Lagrange coefficients in the following for all cases for parabolic and hyperbolic. For full derivation, refer to Curtis [2005]:

$$f = 1 - \frac{y(z)}{r_1}$$

$$g = A\sqrt{\frac{y(z)}{\mu}}$$

$$\dot{f} = \frac{\sqrt{\mu}}{r_1 r_2}\sqrt{\frac{y(z)}{C(z)}}\chi[\mathrm{zS(z)} - 1]$$

$$\dot{g} = 1 - \frac{y(z)}{r_2}$$

Where y and A are:

$$y = r_0 + r + A\frac{zS(z) - 1}{\sqrt{C(z)}}$$

$$A = \sin(\Delta\theta^*)\sqrt{\frac{r_1 r_2}{1 - cos\Delta\theta^*}}$$

In this elliptical case we consider *z >0*, Stumpff function is as follow:

$$S(z) = \frac{\sqrt{z} - sin\sqrt{z}}{\sqrt{z}^3}$$

$$C(z) = \frac{1 - cos\sqrt{z}}{z}$$

The algorithm to calculate the value of z can be expressed and solved using an iterative method: $z_{i+1} = z_i - \frac{f(z_i)}{f'(z_i)}$

Alternatively, formulation based on Izzo [2014] with Householder method has shown to be less computationally intensive while delivering accurate results and could potentially be investigated for more complicated trajectories.

Hyperbolic Excess Velocity and C3

$$V_{\infty\, departure} = v_d - v_{Earth}$$
$$V_{\infty\, arrival} = v_a - v_{jupiter}$$
$$C_{3\, departure} = v_{\infty\, departure}^2$$
$$C_{3\, arrival} = v_{\infty\, arrival}^2$$

Validation of code

As Jupiter's porkchop plot data are not available from NASA, the code is validated using Mars' data (Fig. 4) for launch window between 30 April 2005 and 7 Oct 2005 and arrival window between 16 Nov 2005 and 21 Dec 2006. Qualitatively, using max C_3 of launch at 45, the general features are captured correctly.

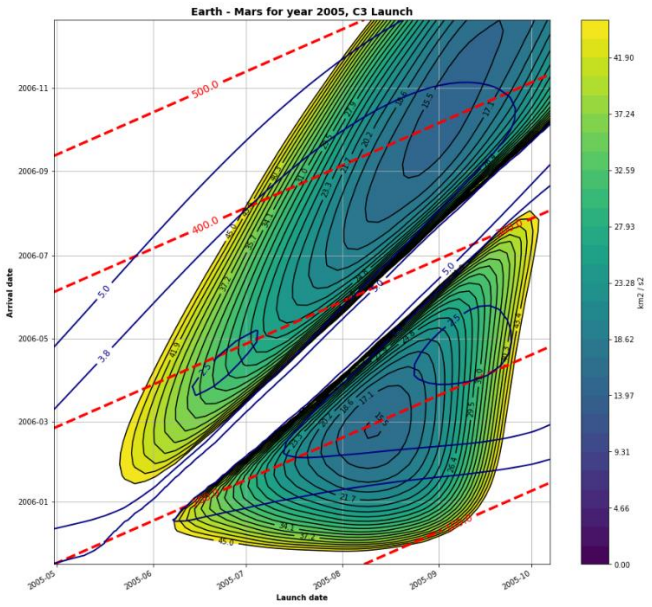

Figure 3: Porkchop plot of Mars generated using Python

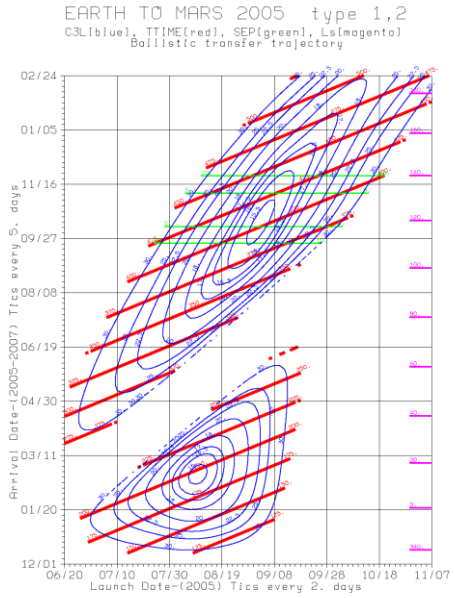

Figure 4: Porkchop plot of Mars obtained from NASA

V. Results and Analysis

A. Criteria for Mission Feasibility

Using Europa Clipper's mission as a reference, Europa Clipper has a planned launch mass of approximately 6000kg in comparison to Juno's launch mass of approximately 5000kg. Clipper is also tentatively planned to be launched with Space Launch System (SLS) Block 1 in order to arrive at Jupiter on a direct trajectory in less than 3 years.

Based on Fig. 5, SLS capability reported in 2014 (Creech [2014]) that a suitable launch characteristic energy for a 6 metric ton payload is approximately 80km^2/s^2 with the 70t SLS configuration. Using the option of 130t configuration, the feasible characteristic energy could increase up to 115 km^2/s^2. In anticipation of further propulsion technology advancement between 2025 to 2040, a max C_3 of 150 km^2/s^2 are plotted for comparison and it is observed that the features become larger as there are better technology to support higher launch characteristic energy (Fig. 6).

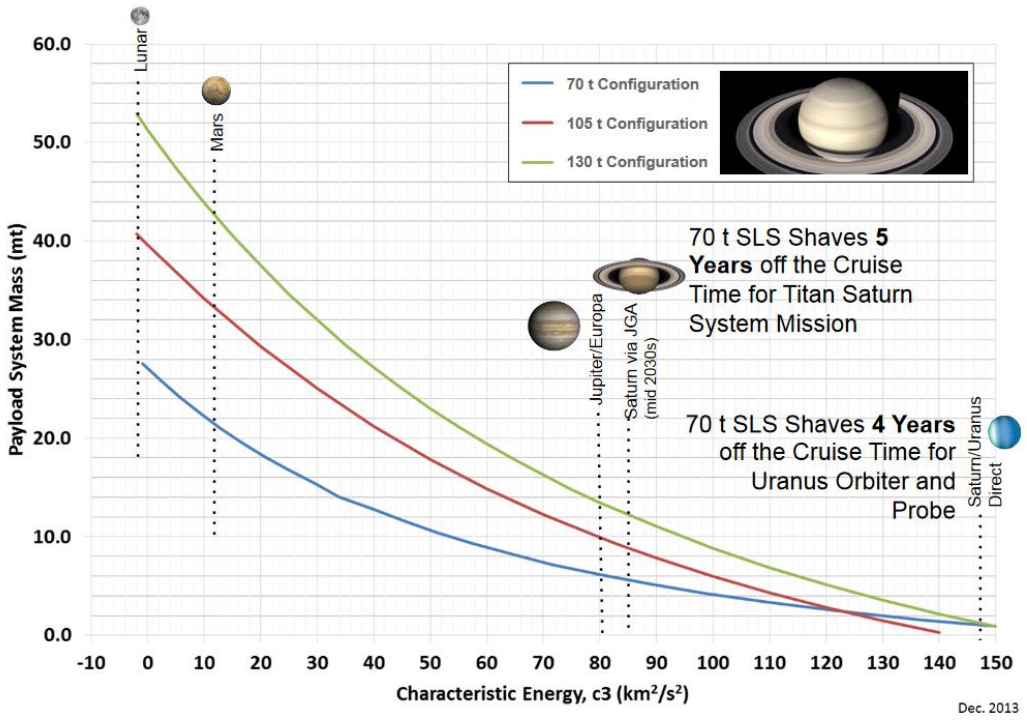

Figure 5: Performance of Space Launch System (Creech [2014])

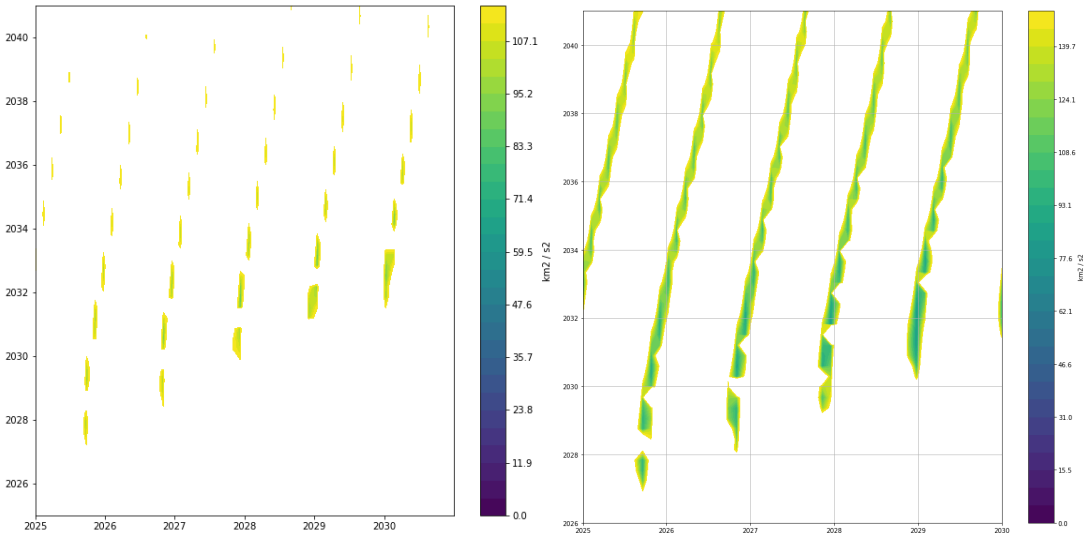

Figure 6: Comparison of Porkchop plot of max characteristic energy of 115 and 150km^2/s^2

B. Porkchop Plot Analysis

In Fig. 7, we observed a similar feasible launch period repeating approximately every 1 year. The maximum C_3 is filtered up to 115km^2/s^2 based on the estimation of a feasible launch vehicles presented above. It is observed that there are 12-13 repeating launch window within our launch time span of interest. The porkchop plot is decomposed into the individual cycles for comparison in terms of the duration of the flight and characteristic energy. 5 windows of the lowest C_3d have been selected and summarized in Table 1 below. All plots are included in appendix for reference.

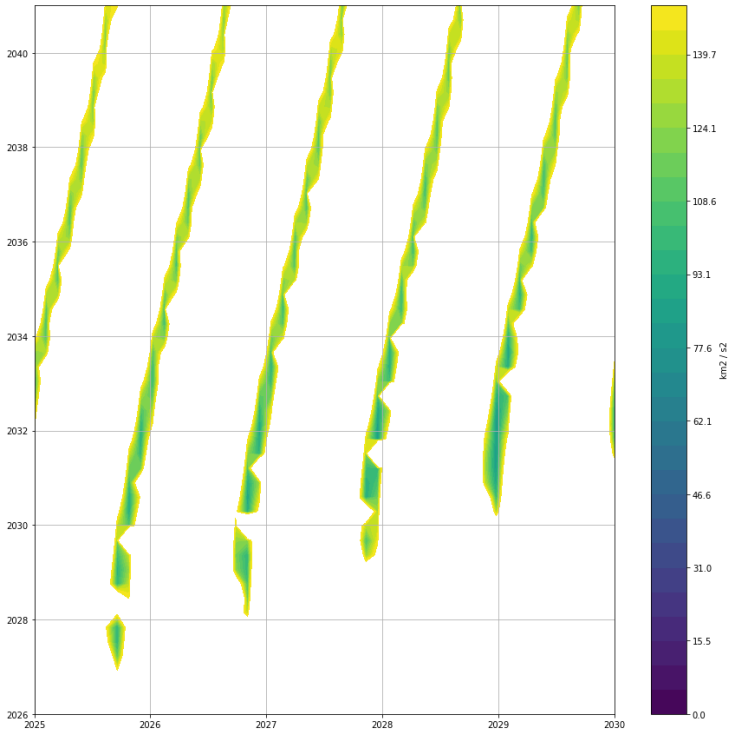

Figure 7: Porkchop plot of Earth to Jupiter (Launch date from 2025 to 2030 and arrival date from 2026 to 2040)

Each window is magnified for higher resolution for more detailed analysis. In Fig 8, two "craters" are observed for each window; the lower and upper correspond to type I (<180° or short trajectory) and type II (>180° or long trajectory) transfers respectively. Type I trajectory is shown to be between 500 to 1000 days and for type II trajectory are often 1000 days and beyond. If time of flight is of top priority, type I is the ideal transfer trajectory to be considered for the mission. In this project, we do not consider the return of the spacecraft as we assumed it is an unmanned mission (for instance satellites), and often deorbits into planets' atmosphere at the end of the mission.

For the values of C_{3d}, the windows presented in Table 1 has shown the lowest minimum local C_{3d} among all launch window based on qualitative observation.

Table 1: Summary of 5 launch window between 2025 to 2040 of lowest estimated$C3_d$

Window	Time Span	Type of Transfer	Min C3 Departure (km^2/s^2)	Min C3 Arrival* (km^2/s^2)	Duration (days)
3	11-2027 to 09-2028	II	~83.3	~31.7	1250-1500
4	11-2028 to 09-2029	II	~79.3	~35.7	1000-1200
5	12-2029 to 09-2030	I	~79.3	~37.7	900-1000
6	02-2031 to 09-2031	I	~83.3	~39.7	750-1000
8	04-2033 to 09-2033	II	~83.3	>43.6	1250-1500

*Corresponds to the approximate time of min C3 departure

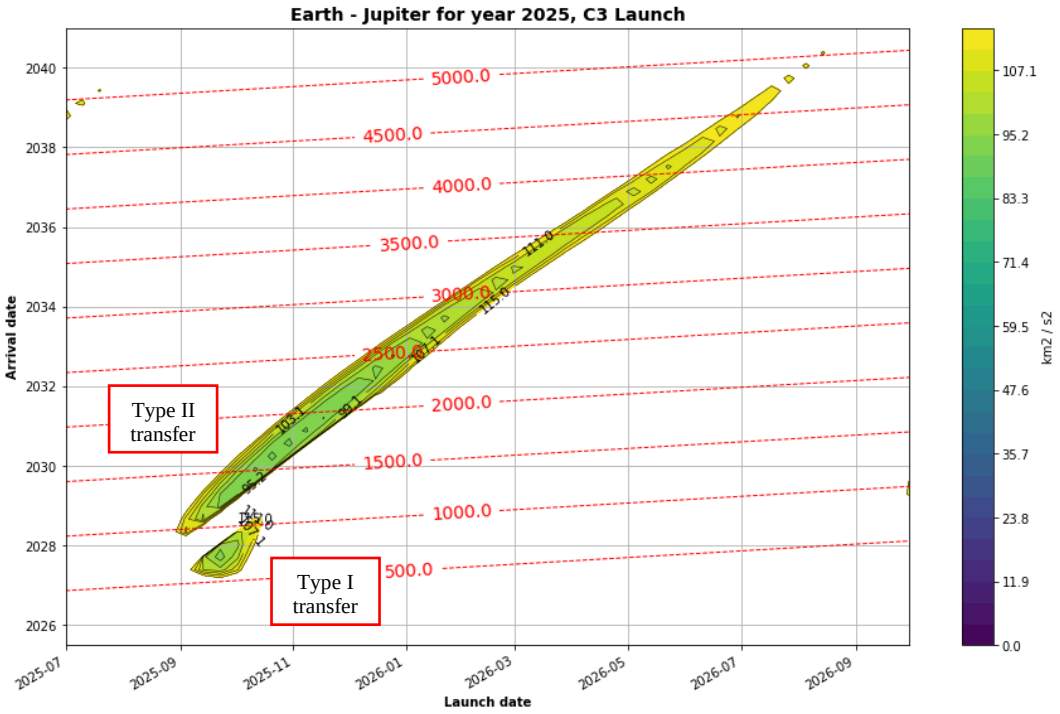

Figure 8: Porkchop plot of a single cycle between Jul 2025 to Sep 2026. The upper and lower craters correspond to type II and type I trajectories respectively.

On the other hand, the transfer with the lowest C_3 for departure may not necessarily have the lowest C_3 for arrival. A comparison of both should be conducted to identify the optimal trajectory. For window 5 in Fig 9, it is observed that the C_3 local minima for departure are in close proximity to the C_3 minima for arrival (in between 39.7 and 35.7km^2/s^2) which is desirable. For window 4 (Fig. 10), in contrast, it has lower min C_3 arrival opportunity compare to window 5 given the same min C_3 departure at the expense of the higher time of flight. The optimal selection would ultimately be determined based on mission objective.

The impulse maneuvers and C_3 for this type I trajectory in window 5 are shown below and is very close to our initial qualitative estimation:

$$\Delta V_{departure} = 9.07 km\, s^{-1}$$
$$\Delta V_{arrival} = 6.12 km\, s^{-1}$$
$$C_{3\,departure} = 82.3 km^2 s^{-2}$$
$$C_{3\,arrival} = 37.5 km^2 s^{-2}$$

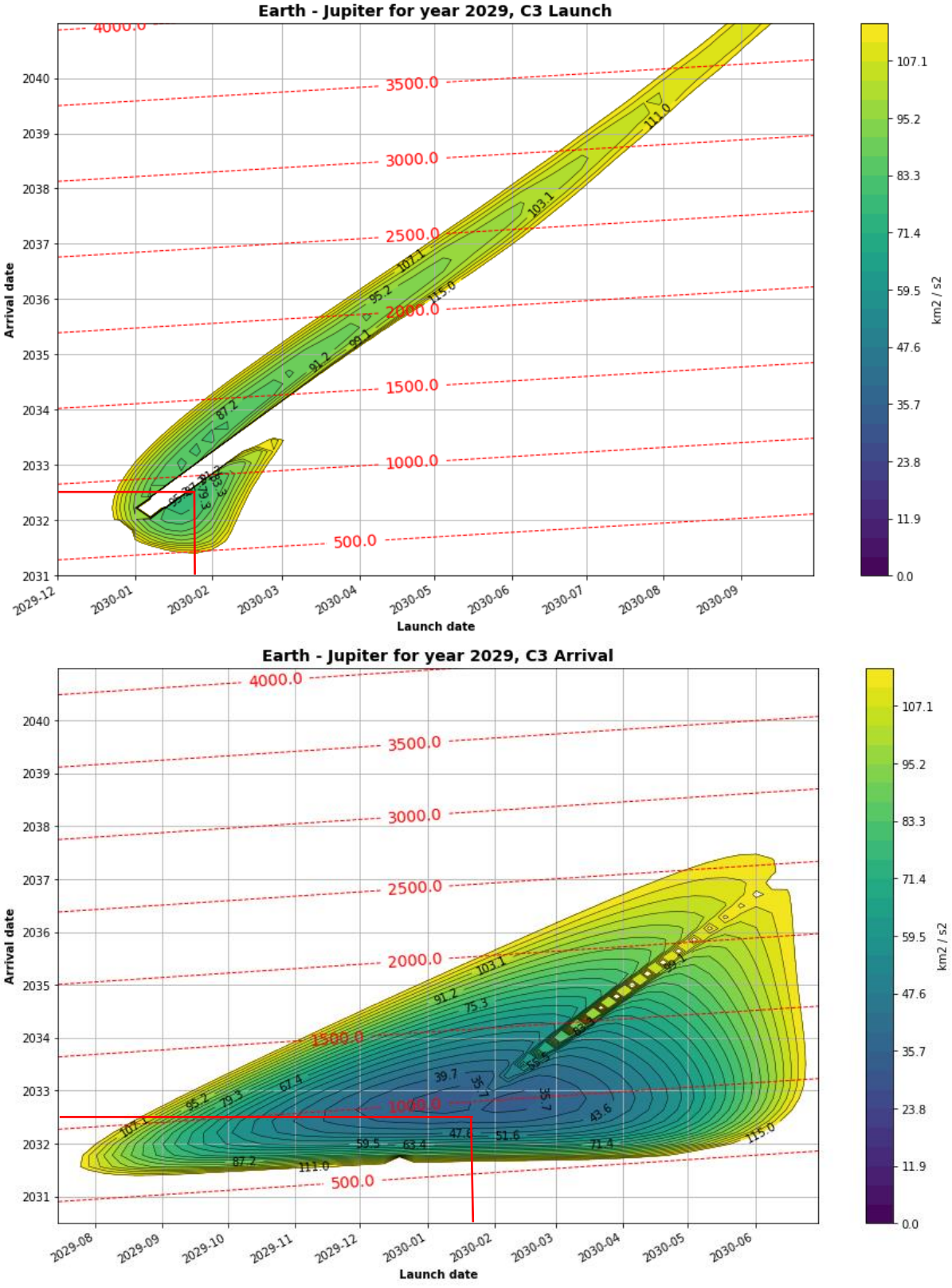

Figure 9: Window 5 (Dec 2029 to 07-2030) – C_3 departure and arrival

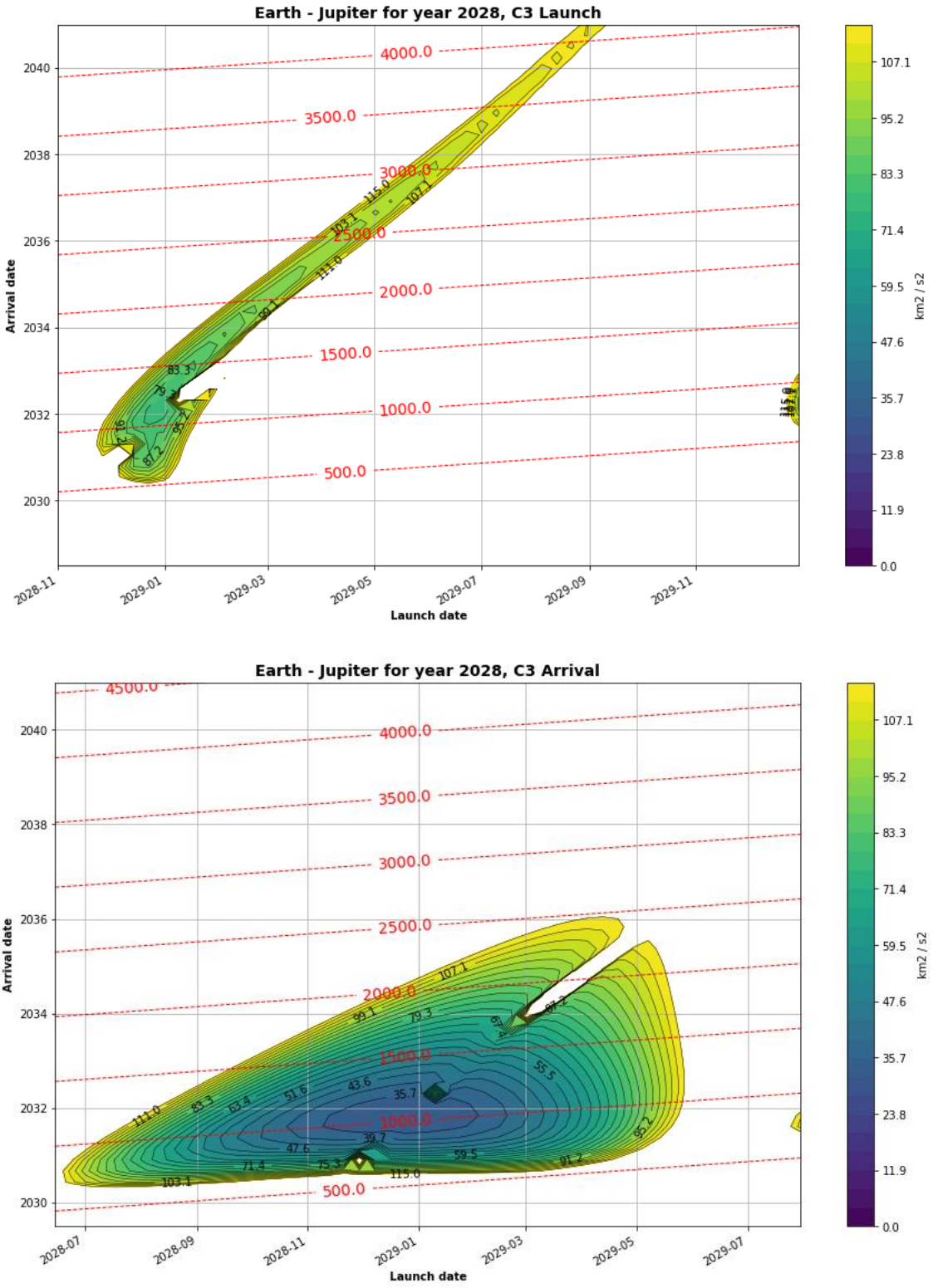

Figure 10: Window 4 (Nov 2028 to Sep 2029) - C_3 departure and arrival

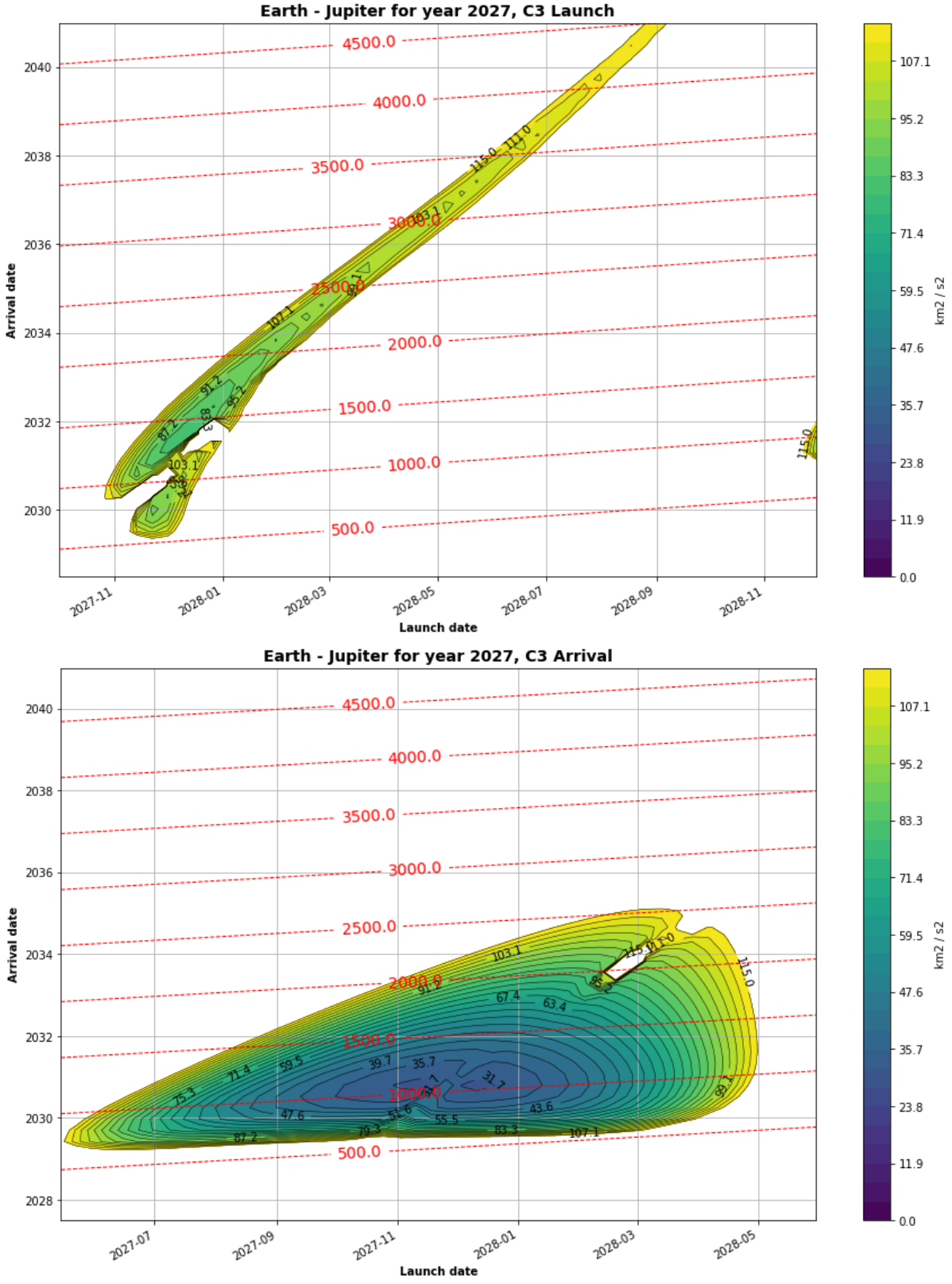

Figure 11: Window 3 (Nov 2027 to Sep 2028) - C_3 departure and arrival

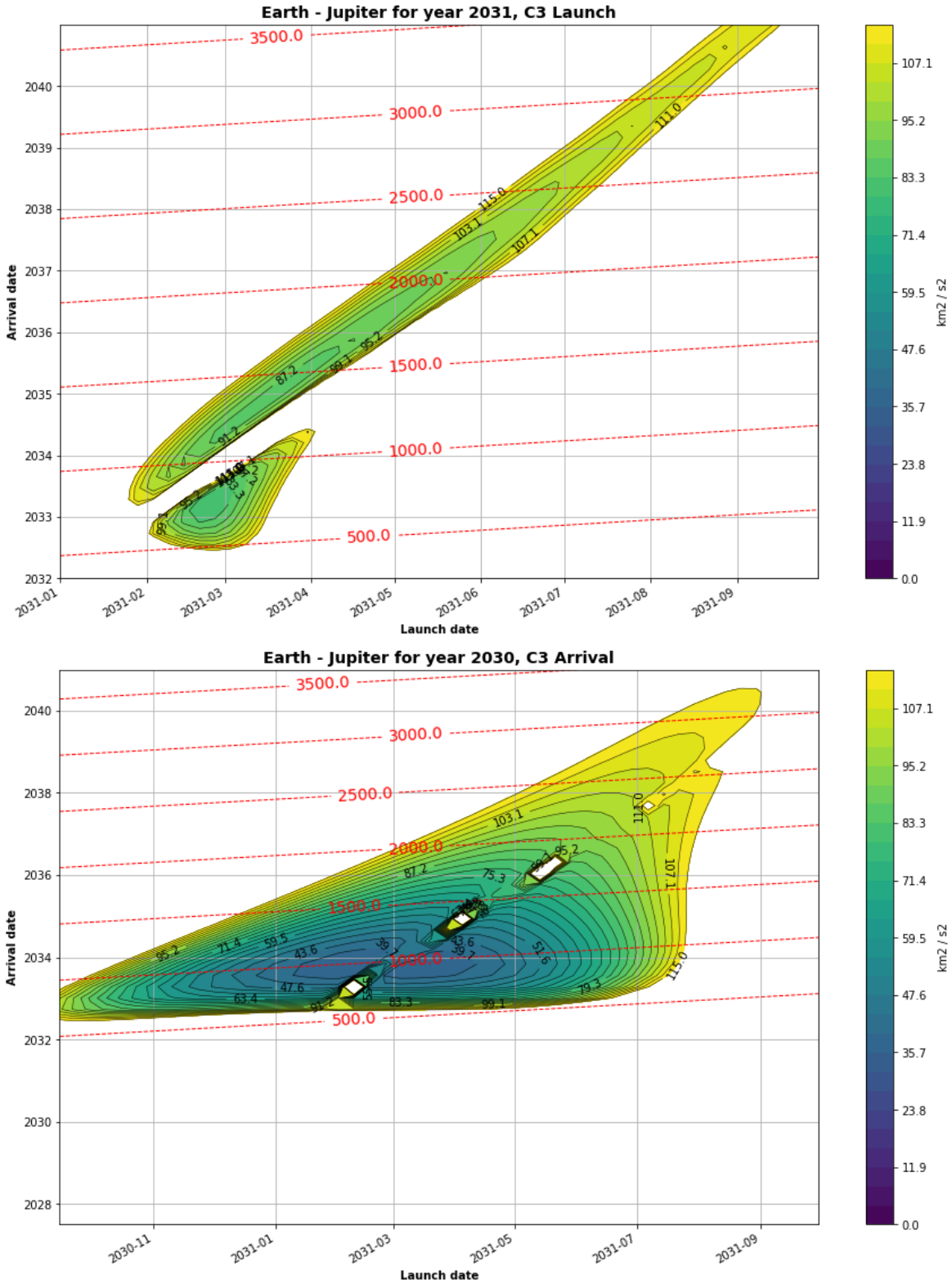

Figure 12: Window 6 (Feb 2031 to Sep 2031) - C_3 departure and arrival

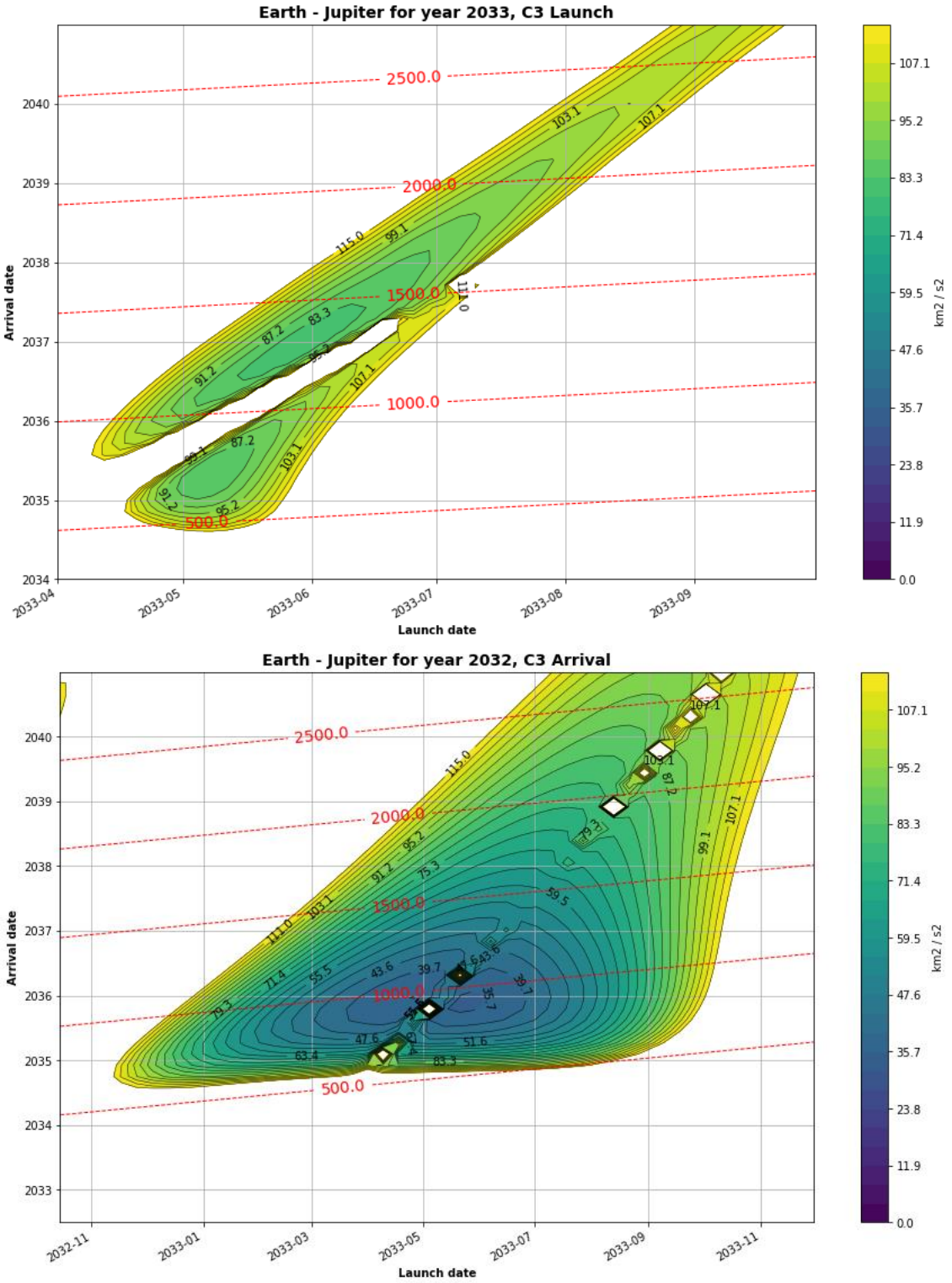

Figure 13: Window 8 (Apr 2033 to Sep 2033) - C_3 departure and arrival

C. Trajectory data

To visualize the trajectory, launch window 5 is selected and analyzed in detail assuming time of flight and launch vehicle limitation are the priority. The launch and arrival date are approximated visually based on the contours from Fig. 9.

The trajectory of the Mars orbital period is approximately 687 days and it is observed that the trajectory will not encounter with the Mars during the transfer. The type I trajectory takes approximately 893 days to insert into Jupiter's orbit.

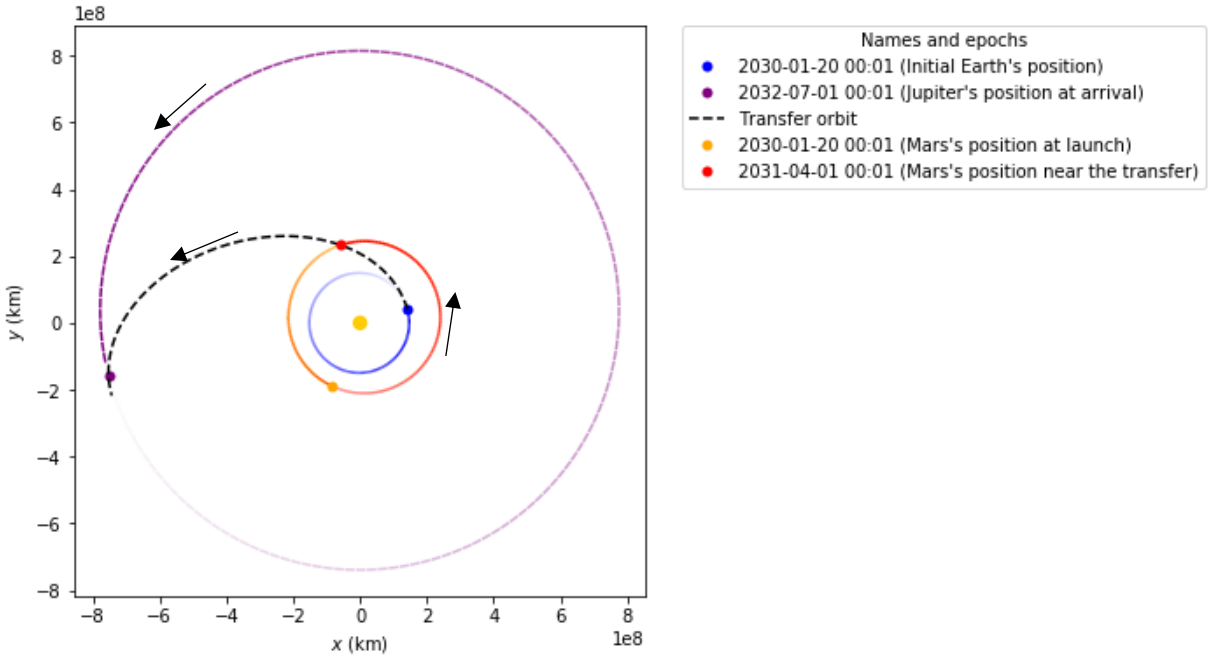

Figure 14: Earth to Jupiter Trajectory with Window 5 launch opportunity. Direction of motion is in anti-clockwise direction

Using the initial and final position and velocity vectors, the General Mission Analysis Tool (GMAT), a NASA mission planning software, is used to calculate the maneuvers given a desired time of flight of 893 days. The impulse maneuvers are computed using the code and input as initial guess in GMAT and the solution converged in 3 iterations given a tolerance of 1km for position vector and 0.0001km/s for velocity vector. Comparing Fig. 15 and Fig. 16, the trajectory generated using the code is qualitatively validated.

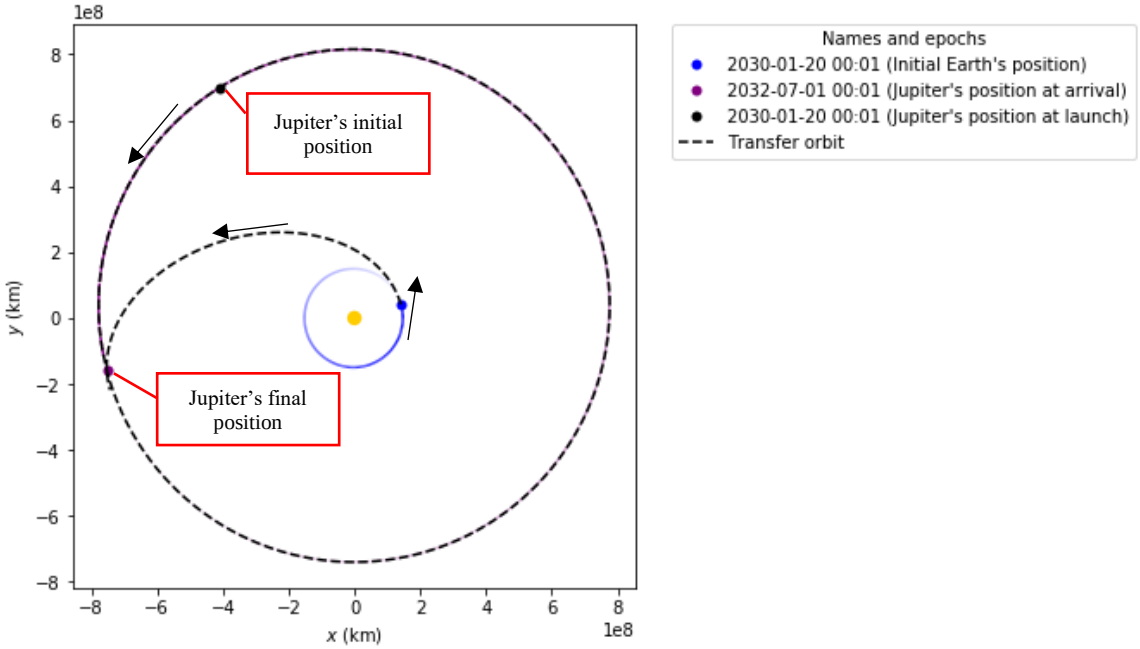

Figure 15: Illustration of Jupiter's orbit during the transfer

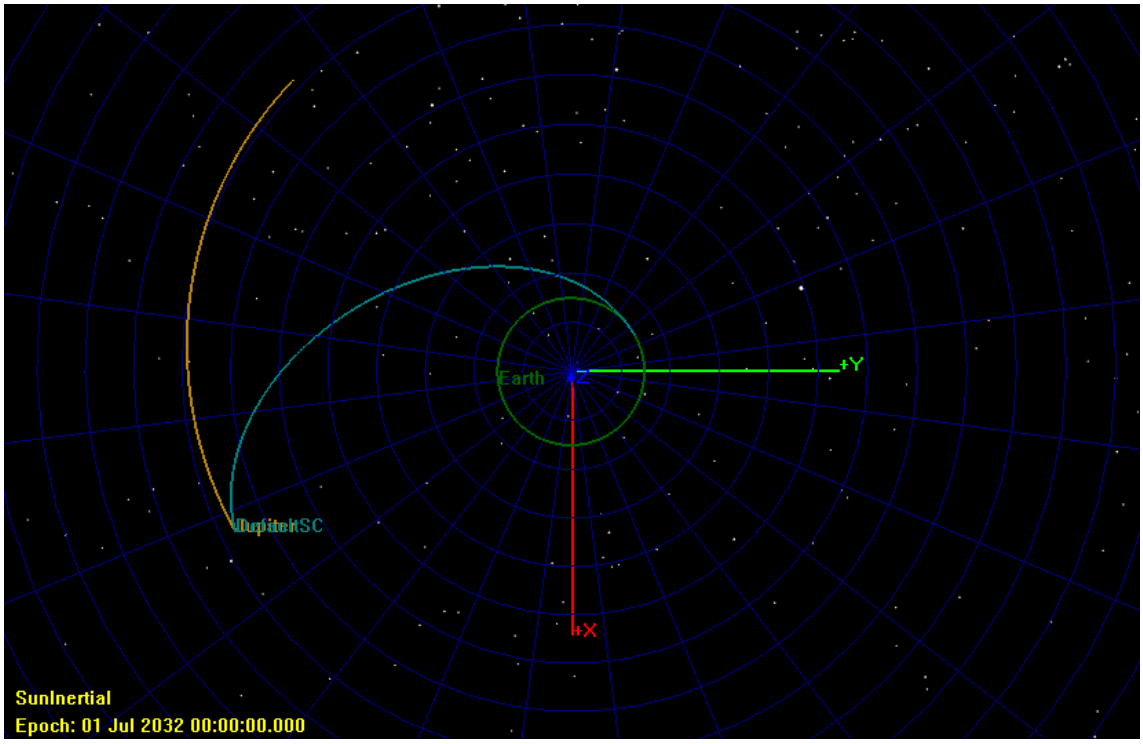

Figure 16: Validation of trajectory using GMAT

VI. Conclusion and Future Work

In this project, the detailed computation framework is illustrated and the feasibility of the launch and arrival opportunities for Jupiter's mission is obtained and discussed. It is shown that a direct trajectory from Earth to Jupiter is feasible between 2030 and 2032 in 893 days, which is within the time of flight requirement for Europa Clipper (3 years) using the planned SLS technology. The porkchop plots and trajectory solutions are also validated using NASA's porkchop plot for Mars and GMAT respectively.

Only a direct ballistic trajectory is considered in this project and it is assumed that the trajectory is co-planar with the planets. Other fly-by opportunities should be considered for a more complete trade studies on time of flight and fuel efficiency. Depending on the mission objective, fly-by opportunities could also be beneficial to achieve the secondary objective of observing other celestial objects. The presence of asteroid belt between Mars and Jupiter is also not considered in this trajectory design and is assumed to be not sufficiently dense to pose a risk to the instruments.

Appendix

Appendix 1. Source code for the computation of the trajectory:

Generate Coordinates

```
from astropy import coordinates as coord, units as u
from poliastro.bodies import (
    Earth,
    Jupiter,
    Mars,
    Mercury,
    Moon,
    Sun,
)

def _get_state(body, time):
    """ Computes the position of a body for a given time. """

    solar_system_bodies = [
        Sun,
        Earth,
        Mars,
        Jupiter
    ]

    # check if body belongs to poliastro.bodies
    if body in solar_system_bodies:
        rr, vv = coord.get_body_barycentric_posvel(body.name, time)
    else:
        rr, vv = body.propagate(time).rv()
        rr = coord.CartesianRepresentation(rr)
        vv = coord.CartesianRepresentation(vv)

    return rr.xyz, vv.xyz
```

Lambert's problem using Lagrange coefficient with Universal anomaly variable – Curtis [2014]

```
import numpy as np
from numpy import cross, pi
from numpy.linalg import norm

from poliastro.core.hyper import hyp2f1b
from poliastro.core.stumpff import c2, c3

def lambert(k, r0, r, tof, short, numiter, rtol):
    """Solves the Lambert's problem.
    The algorithm returns the initial velocity vector and the final one, these are
    computed by the following expresions:

    # prograde or retrograde trajectory
    if short:
        t_m = +1
    else:
        t_m = -1

    # calculate magnitude of both position vector
```

```
    norm_r0 = np.dot(r0, r0) ** 0.5
    norm_r = np.dot(r, r) ** 0.5
    norm_r0_times_norm_r = norm_r0 * norm_r
    norm_r0_plus_norm_r = norm_r0 + norm_r

    cos_dnu = np.dot(r0, r) / norm_r0_times_norm_r

    A = t_m * (norm_r * norm_r0 * (1 + cos_dnu)) ** 0.5

    if A == 0.0:
        raise RuntimeError("Cannot compute orbit, phase angle is 180 degrees")

    # set initial guess for bisection method
    psi = 0.0
    psi_low = -4 * np.pi
    psi_up = 4 * np.pi

    count = 0

    while count < numiter:
        y = norm_r0_plus_norm_r + A * (psi * c3(psi) - 1) / c2(psi) ** 0.5
        if A > 0.0:

            # Readjust xi_low until y > 0.0
            while y < 0.0:
                psi_low = psi
                psi = (
                    0.8
                    * (1.0 / c3(psi))
                    * (1.0 - norm_r0_times_norm_r * np.sqrt(c2(psi)) / A)
                )
                y = norm_r0_plus_norm_r + A * (psi * c3(psi) - 1) / c2(psi) ** 0.5

        xi = np.sqrt(y / c2(psi))
        tof_new = (xi ** 3 * c3(psi) + A * np.sqrt(y)) / np.sqrt(k)

        # Convergence check
        if np.abs((tof_new - tof) / tof) < rtol:
            break
        count += 1
        # Bisection check
        condition = tof_new <= tof
        psi_low = psi_low + (psi - psi_low) * condition
        psi_up = psi_up + (psi - psi_up) * (not condition)

        psi = (psi_up + psi_low) / 2
    else:
        raise RuntimeError("Maximum number of iterations reached")

    f = 1 - y / norm_r0
    g = A * np.sqrt(y / k)
    gdot = 1 - y / norm_r
    v0 = (r - f * r0) / g
    v = (gdot * r - r0) / g

    return v0, v
```

Lambert's Problem (Alternative method for checking) - Prussing & Conway [1993]

```
import numpy as np
import math
from astroquery.jplhorizons import Horizons
import matplotlib.pyplot as plt

def mag_vec(x):
    # define magnitude of vec
    mag = np.linalg.norm(x)
    return mag.item()

def tof_min(s,c,x,grav_sun):
    # find tof_min
    alf_min = math.pi
```

```
    if x == 'large':
        bet_min = -2*math.asin(math.sqrt((s-c)/s))

    else:
        bet_min = 2*math.asin(math.sqrt((s-c)/s))

    a_min = s/2

    n_min = math.sqrt(grav_sun/(a_min**3))

    tof_min = (alf_min - bet_min -(math.sin(alf_min)-math.sin(bet_min)))/n_min
    return tof_min

def tof_g(s,c,d,x,tof,grav_sun):

    alf_0 = 2*math.asin(math.sqrt(s/(2*d)))
    bet_0 = 2*math.asin(math.sqrt((s-c)/(2*d)))
    n = math.sqrt(grav_sun/(d**3))

    if x == 'large':
        if tof > tof_min(s,c,x,grav_sun):
            alf = 2*math.pi - alf_0
            bet = -bet_0
        else:
            alf = alf_0
            bet = -bet_0
    else:
        if tof > tof_min(s,c,x,grav_sun):
            alf = 2*math.pi - alf_0
            bet = bet_0
        else:
            alf = alf_0
            bet = bet_0
    tof_g= (alf - bet - (math.sin(alf) - math.sin(bet)))*(1/n)  # calculate residual

    return tof_g, alf, bet


def lamb_sol(r1v,v1iv,r2v,v2fv,t1,t2,x,a2):
    grav_sun =  1.32712e11 #3.986004415e5

    tof = (t2-t1)*24*60*60 #seconds
    tol = 0.000001
    if tof<=0:
        return None

    if x =='large': #larger transfer angle
        d_ta= 2 *math.pi - math.acos(np.dot(r1v,r2v)/(mag_vec(r1v)*mag_vec(r2v)))
        c = (math.sqrt(mag_vec(r1v)**2 +mag_vec(r2v)**2 - 2* mag_vec(r1v)*mag_vec(r2v)*math.cos(d_ta)))
        s = 0.5 * (mag_vec(r1v) + mag_vec(r2v) + c) #convert to float

        a1 = s/2
        d= (a1+a2)/2
        #bisection method
        while abs(a2-a1) >=tol or abs(tof_g(s,c,d,x,tof,grav_sun)[0]-tof) >=tol:
            d= (a1+a2)/2
            if tof_g(s,c,d,x,tof,grav_sun)[0]-tof == 0.0:
                return d
            if (tof_g(s,c,d,x,tof,grav_sun)[0]-tof)*(tof_g(s,c,a1,x,tof,grav_sun)[0]-tof)> 0:
                a1 = d
            else:
                a2 = d


    else: #smaller transfer angle
        d_ta = math.acos((np.dot(r1v,r2v))/(mag_vec(r1v)*mag_vec(r2v)))
        c = math.sqrt(mag_vec(r1v)**2 +mag_vec(r2v)**2 - 2* mag_vec(r1v)*mag_vec(r2v)*math.cos(d_ta))
        s = 0.5* (mag_vec(r1v)+mag_vec(r2v) + c)

        a1 = s/2
```

```
        d= (a1+a2)/2
        while abs(a2-a1) >=tol or abs(tof_g(s,c,d,x,tof,grav_sun)[0]-tof) >=tol:
            d= (a1+a2)/2
            if tof_g(s,c,d,x,tof,grav_sun)[0]-tof == 0.0:
                return d
            if (tof_g(s,c,d,x,tof,grav_sun)[0]-tof)*(tof_g(s,c,a1,x,tof,grav_sun)[0]-tof)> 0:
                a1 = d
            else:
                a2 = d

    #calculate semi-perimeter
    p=4*d*(s-mag_vec(r1v))*(s-mag_vec(r2v))*(math.sin((tof_g(s,c,d,x,tof,grav_sun)[1]+tof_g(s,c,d,x,tof,grav_sun)[2])/2))**2/(c**2)
    e=math.sqrt(1-p/d)
    #print(p,e)

    #find true anomaly
    ta1=math.acos((p/mag_vec(r1v)-1)/e)
    ta2=math.acos((p/mag_vec(r2v)-1)/e)
    if abs(d_ta-(ta2-ta1))<0.000001:
        ta2=ta2
        ta1=ta1
    elif abs(d_ta-(-ta2-ta1))<0.000001:
        ta2 =-ta2
        ta1 =ta1
    elif abs(d_ta-(-ta2+ta1))<0.000001:
        ta2=-ta2
        ta1=-ta1
    else:
        ta2=ta2
        ta1=-ta1
    print("true anomaly 1 and 2:", ta1,",",ta2)
    # use fg functions and return vd, va
    f1=1-mag_vec(r2v)*(1-math.cos(d_ta))/p
    g1=mag_vec(r1v)*mag_vec(r2v)*math.sin(d_ta)/math.sqrt(grav_sun*p)
    df1=math.sqrt(grav_sun/p)*(math.tan(d_ta/2))*((1-math.cos(d_ta))/p-1/mag_vec(r1v)-1/mag_vec(r2v))
    dg1=1-mag_vec(r1v)*(1-math.cos(d_ta))/p
    print("Lagrange coefficients (f,g,f'g'): ",f1,g1,df1,dg1)
    print("Planet velocity 1 and 2:",v1iv,v2fv)
    #find departure and arrival velocity within the transfer arc
    v_d=(r2v-f1*r1v)/g1
    v_a=df1*r1v+dg1*v_d
    print("departure and arrival velocity of transfer:",v_d,v_a)
    v_inf_d=v_d-v1iv
    v_inf_a=v_a-v2fv
    print("vinf departure and arrival:",v_inf_d,v_inf_a)
    print(mag_vec(v_inf_d),mag_vec(v_inf_a))
    c3d=mag_vec(v_inf_d)**2
    c3a=mag_vec(v_inf_a)**2
    print("c3d and c3a:",c3d,c3a)

    return c3d

au2km=149597870.7
d2s =24*60*60 #days to seconds

# inputs in sun-centered inertial coordinate
t1 = Time("2030-01-20 00:00", scale="utc").jd
t2 = Time("2032-07-01 00:00", scale="utc").jd

#Extraction from JPL Horizon web interface for comparison
r1v=np.array([-72576391.16328001, 128061475.5880728, -8055.475574925542])
v1iv=np.array([-2.639048456109558E+01,-1.479176337239575E+01,2.010693239588690E-03])
r2v=np.array([3.278496985125721E+08,-6.961872482800779E+08 ,-4.440828489689320E+06])
v2fv=np.array([1.167433018601784E+01, 6.182818012521681E+00,-2.868375221595132E-01])

lamb_sol(r1v,v1iv,r2v,v2fv,t1,t2,'small',1e9)
```

Targeting function returns the departure and arrival velocities

```
def _targetting(departure_body, target_body, t_launch, t_arrival):
    """This function returns the increment in departure and arrival velocities."""

    # Get position and velocities for departure and arrival
    rr_dpt_body, vv_dpt_body = _get_state(departure_body, t_launch)
    rr_arr_body, vv_arr_body = _get_state(target_body, t_arrival)

    # Transform into Orbit objects
    attractor = departure_body.parent
    ss_dpt = Orbit.from_vectors(attractor, rr_dpt_body, vv_dpt_body, epoch=t_launch)
    ss_arr = Orbit.from_vectors(attractor, rr_arr_body, vv_arr_body, epoch=t_arrival)

    # Define time of flight
    tof = ss_arr.epoch - ss_dpt.epoch

    if tof <= 0:
        return None, None, None, None, None

    try:
        # Lambert is now a Maneuver object
        man_lambert = Maneuver.lambert(ss_dpt, ss_arr)

        # Get norm delta velocities
        dv_dpt = norm(man_lambert.impulses[0][1])
        dv_arr = norm(man_lambert.impulses[1][1])

        # Compute all the output variables
        c3_launch = dv_dpt ** 2
        c3_arrival = dv_arr ** 2

        return (
            dv_dpt.to(u.km / u.s).value,
            dv_arr.to(u.km / u.s).value,
            c3_launch.to(u.km ** 2 / u.s ** 2).value,
            c3_arrival.to(u.km ** 2 / u.s ** 2).value,
            tof.jd,
        )

    except AssertionError:
        return None, None, None, None, None

targetting_vec = np.vectorize(
    _targetting,
    otypes=[np.ndarray, np.ndarray, np.ndarray, np.ndarray, np.ndarray],
    excluded=[0, 1],
)
```

Porkchop Plot

```
def porkchop(
    departure_body,
    target_body,
    launch_span,
    arrival_span,
    ax=None,
    tfl=True,
    vhp=True,
    max_c3=45.0 * u.km ** 2 / u.s ** 2,
    max_vhp=5 * u.km / u.s,
):
    dv_launch, dv_arrival, c3_launch, c3_arrival, tof = targetting_vec(
        departure_body,
        target_body,
        launch_span[np.newaxis, :],
        arrival_span[:, np.newaxis],
    )

    # Start drawing porkchop

    if ax is None:
```

```
    fig, ax = plt.subplots(figsize=(10, 10))
else:
    fig = ax.figure

c3_levels = np.linspace(0, max_c3.to(u.km ** 2 / u.s ** 2).value, 30)

c = ax.contourf(
    [D.to_datetime() for D in launch_span],
    [A.to_datetime() for A in arrival_span],
    c3_launch,
    c3_levels,
)

line = ax.contour(
    [D.to_datetime() for D in launch_span],
    [A.to_datetime() for A in arrival_span],
    c3_launch,
    c3_levels,
    colors="black",
    linestyles="solid",
    linewidths=0.5,
)

cbar = fig.colorbar(c)
cbar.set_label("km2 / s2")
ax.clabel(line, inline=1, fmt="%1.1f", colors="k", fontsize=10)

if tfl:

    time_levels = np.linspace(100, 500, 5)

    tfl_contour = ax.contour(
        [D.to_datetime() for D in launch_span],
        [A.to_datetime() for A in arrival_span],
        tof,
        time_levels,
        colors="red",
        linestyles="dashed",
        linewidths=1,
    )

    ax.clabel(tfl_contour, inline=1, fmt="%1.1f", colors="r", fontsize=14)

if vhp:

    vhp_levels = np.linspace(0, max_vhp.to(u.km / u.s).value, 5)

    vhp_contour = ax.contour(
        [D.to_datetime() for D in launch_span],
        [A.to_datetime() for A in arrival_span],
        dv_arrival,
        vhp_levels,
        colors="navy",
        linewidths=1.0,
    )

    ax.clabel(vhp_contour, inline=1, fmt="%1.1f", colors="navy", fontsize=12)

ax.grid()
fig.autofmt_xdate()

if not hasattr(target_body, "name"):
    ax.set_title(
        f"{departure_body.name} - Target Body for year {launch_span[0].datetime.year}, C3 Launch",
        fontsize=14,
        fontweight="bold",
    )
else:
    ax.set_title(
        f"{departure_body.name} - {target_body.name} for year {launch_span[0].datetime.year}, C3 Launch",
```

```
        fontsize=14,
        fontweight="bold",
    )

  ax.set_xlabel("Launch date", fontsize=10, fontweight="bold")
  ax.set_ylabel("Arrival date", fontsize=10, fontweight="bold")

  return (
    dv_launch * u.km / u.s,
    dv_arrival * u.km / u.s,
    c3_launch * u.km ** 2 / u.s ** 2,
    c3_arrival * u.km ** 2 / u.s ** 2,
    tof * u.d,
  )
```

Running the Code

```
launch_span = time_range("2025-01-01", end="2026-12-31")
arrival_span = time_range("2025-01-01", end="2040-12-31")

dv_dpt, dv_arr, c3dpt, c3arr, tof = porkchop(Earth, Jupiter, launch_span, arrival_span, tfl=True, vhp=True, max_c3=115 * u.km**2 / u.s**2)
```

Visualisation of the Trajectory

```
import astropy.units as u
from astropy.time import Time
from astropy.coordinates import solar_system_ephemeris

from poliastro.bodies import Sun, Earth, Jupiter, Mars
from poliastro.twobody import Orbit
from poliastro.maneuver import Maneuver
from poliastro.plotting import StaticOrbitPlotter
solar_system_ephemeris.set("jpl")

# Main dates from window 5
date_launch = Time("2030-01-20 00:00", scale="utc")
date_arrival = Time("2032-07-01 00:00", scale="utc")
date_Mars = Time("2031-04-01 00:00", scale="utc")

# Solve for departure and target orbits
ss_earth = Orbit.from_body_ephem(Earth, date_launch)
ss_jupiter = Orbit.from_body_ephem(Jupiter, date_arrival)
# for checking encounter
ss_Mars1 = Orbit.from_body_ephem(Mars, date_launch)
ss_Mars2 = Orbit.from_body_ephem(Mars, date_Mars)
ss_jupiter2 = Orbit.from_body_ephem(Jupiter, date_launch)

# Solve for the transfer maneuver
man_lambert = Maneuver.lambert(ss_earth, ss_jupiter)

# Get the transfer and final orbits
ss_trans, ss_target = ss_earth.apply_maneuver(man_lambert, intermediate=True)
imp_a, imp_b = man_lambert.impulses
print("Initial impulse:", imp_a)
print("Final impulse:", imp_b)

plotter = StaticOrbitPlotter()

plotter.plot(ss_earth, label="Initial Earth's position", color="blue", trail=True)
plotter.plot(ss_jupiter, label="Jupiter's position at arrival", color="purple", trail=True)
plotter.plot(ss_jupiter2, label="Jupiter's position at launch", color="black")
plotter.plot_trajectory(ss_trans.sample(max_anomaly=180*u.deg), color="black", label="Transfer orbit")
#plotter.plot(ss_Mars1, label="Mars's position at launch", color="orange", trail=True)
#plotter.plot(ss_Mars2, label="Mars's position near the transfer", color="red", trail=True)
```

References

Astroquery JPL Horizon Database. Accessed at: https://astroquery.readthedocs.io/en/latest/jplhorizons/jplhorizons.html

Bolton, S. J., Lunine, J., Stevenson, D., Connerney, J. E. P., Levin, S., Owen, T. C., ... Thorpe, R. (2017). The Juno Mission. *Space Science Reviews*, *213*(1-4), 5-37. https://doi.org/10.1007/s11214-017-0429-6

Chen, Y., Baoyin, H., Li, J., (2013) Trajectory Analysis and design for A Jupiter Exploration Mission. Chinese Astronomy and Astroophysics, 37 (1), 77-89. https://doi.org/10.1016/j.chinastron.2013.01.008

Creech, S. (2014) *NASA's Space Launch System: A capability for Deep Space Exploration.* NASA.

Curtis, H. D. (2005). *Orbital mechanics for engineering students*. Amsterdam: Elsevier Butterworth Heinemann.

Izzo, D (2014) *Revisiting Lambert's Problem.* Celestial mechanics and Dynamical Astronomy 121.1 (2014) Accessed at: https://arxiv.org/pdf/1403.2705.pdf

Matousek, S. Sergeyevsky, A (1998) *To Mars and Back: 2002-2020 Ballistic trajectory data for the Mission Architect.* AIAA/AAS Astrodynamics Specialist Conference August 10-12, 1998 Boston MA.

NASA. *"Porkchop" is the First Menu Item on a Trip to Mars.* Accessed at: https://mars.jpl.nasa.gov/spotlight/porkchopAll.html

Prussing, J. E., & Conway, B. A. (1993). *Orbital mechanics*. New York: Oxford University Press.