AERO 423 FINAL REPORT B

By Jonathan Jermstad

NOVEMBER 27, 2023

1 CONTENTS

2 PROBLEM DEFINITION

The objective of this problem is to identify the optimal departure date and time of flight for an Earth-Mars transfer orbit, minimizing the transfer cost function J. The transfer cost is defined as the sum of velocity differences at departure and arrival between the heliocentric departure from Earth and heliocentric arrival at Mars.

The following data was given for this project:

- Departure dates: June 1, 2026, to March 1, 2027
- Departure epoch: November 14, 2026
- Time of flight: 100 to 750 days in 1-day steps
- Earth circular parking altitude: 690 km
- Mars elliptical periapsis altitude: 360 km
- Mars elliptical apogee altitude: 6350 km

The problem will be solved using the patched conics approach and will utilize Lambert's problem to solve for the transfer orbit parameters. For each unique combination of departure times and time of flight, a cost will be calculated and used to select ideal conditions to minimize the delta v needed.

Some constraints for the problem, is that the analysis will not be exact, as it will utilize a patched conics approach, which is a good approximation and initial guess. However, for full accuracy, an additional higher fidelity simulation will be required. An additional assumption is that the earth and mars are going to be assumed to be in the same orbit plane, which, as can be seen in **Figures 1 and 2**, does not perfectly reflect reality.

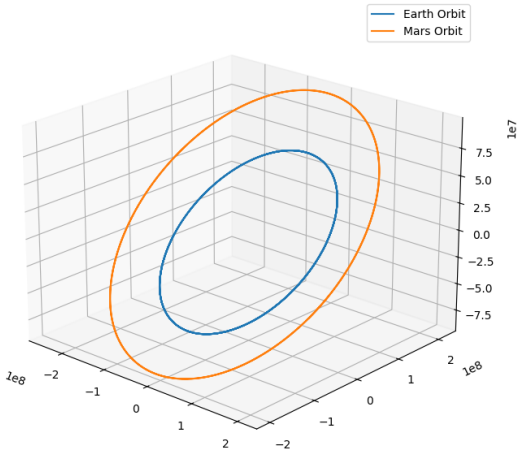


Figure 1. 3d Earth and Mars Orbit Plot

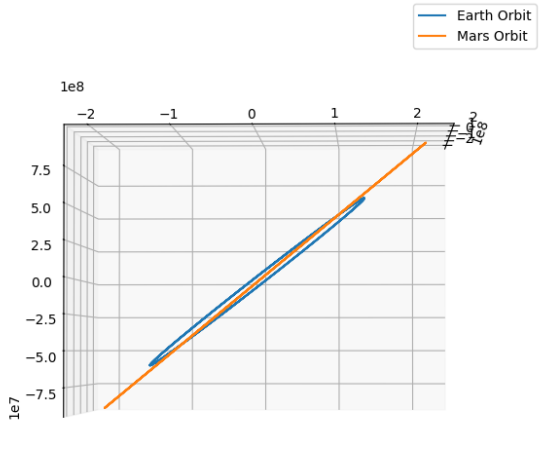


Figure 2. 3d Earth and Mars Orbit Plot (Side View)

3 HELIOCENTRIC DESIGN PHASE

This phase of the project details how the cost (J) of the mission was minimized. To do that the major problem that had to be solved was Lambert's Problem, which is concerned with the determination of an orbit from two position vectors and a time of flight. The following diagram in **Figure 3** describes how Lambert's problem was solved.

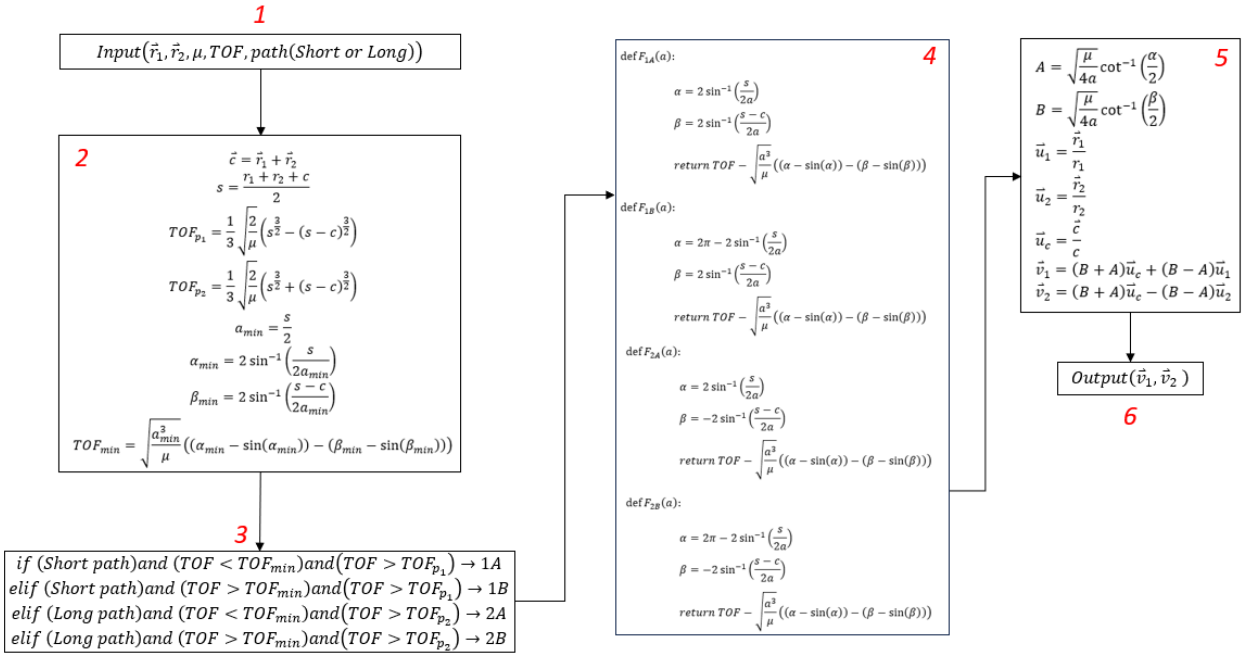


Figure 3. Lambert's Problem Solver Diagram

The Lambert Solver starts with box #1, and it can be seen how variables are inputted into the program. The program accepts both a starting ($\vec{r}_1$) and ending position ($\vec{r}_2$), the standard gravitational parameter (μ), a Time of Flight (TOF), and whether the path taken is the short (1) or long (2) path of solutions. Moving into the next box #2, it can be seen how geometric properties of the orbit are found, and then parabolic and minimum elliptical transfer orbits are found. In box #3, the solution type is determined, if $path == Short\ path$, then the solution is of type 1, and if $path == Long\ path$,then the solution is of type 2, which can be observed in **Figure 4**. Additionally, the behavior of the transfer orbit can be determined, by compared the inputted TOF to the TOF_p, which is the parabolic time of flight. A TOF greater than TOF_p indicates an elliptical orbit, and a TOF less than TOF_p indicates a hyperbolic orbit, which can be ignored. The last solution type indicator is with the comparison between TOF and TOF_{min}, which is the minimum elliptical time of flight. A TOF greater than TOF_{min}, indicates a category B solution, and a TOF less than TOF_{min} indicates a category A solution, as can be observed in **Figure 5**. In box #4, depending on the solution type, the semi parameter, alpha, and beta are found using numerical methods. In box #5, velocity at both positions on the transfer orbit are found, and they are outputted in box #6.

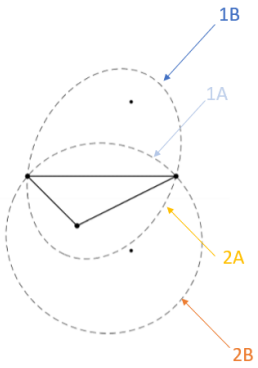


Figure 4. Transfer Orbit Solution Type, Based on Path

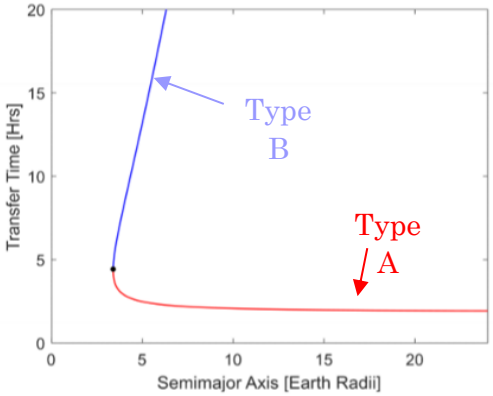


Figure 5. Transfer Orbit Solution Type, Based on Comparison to TOF_p

This approach of using the Lambert Solver was used for a range of initial values $\vec{r}_1$ $(June\ 1{,}2026 - March\ 1{,}2027)$, and a range of departure values $\vec{r}_2$ $(100 - 750\ days\ after\ the\ inital\ values)$. Both positional values were found from an Ephemeris data sheet and resulted in 178,374 unique combinations. For each unique combination, Lamber's problem was solved using the solver defined in **Figure 3**, and the velocity of earth $\vec{v}_{earth}$ and mars $\vec{v}_{mars}$ were pulled from the data sheet as well. The cost was then computed using **Equation 1**:

$$J = |\vec{v}_1 - \vec{v}_{earth}| + |\vec{v}_2 - \vec{v}_{mars}| \quad (1)$$

Any combination of values resulting in a cost > 15 km/s was ignored, and the resulting contour plot was made, shown in **Figure 6**. The values that resulted in a minimum delta v requirement were leaving on 1-Nov-26, and arriving on 7-Sep-27, with a TOF of 310 days, and a minimum cost of 5.1628 km/s.

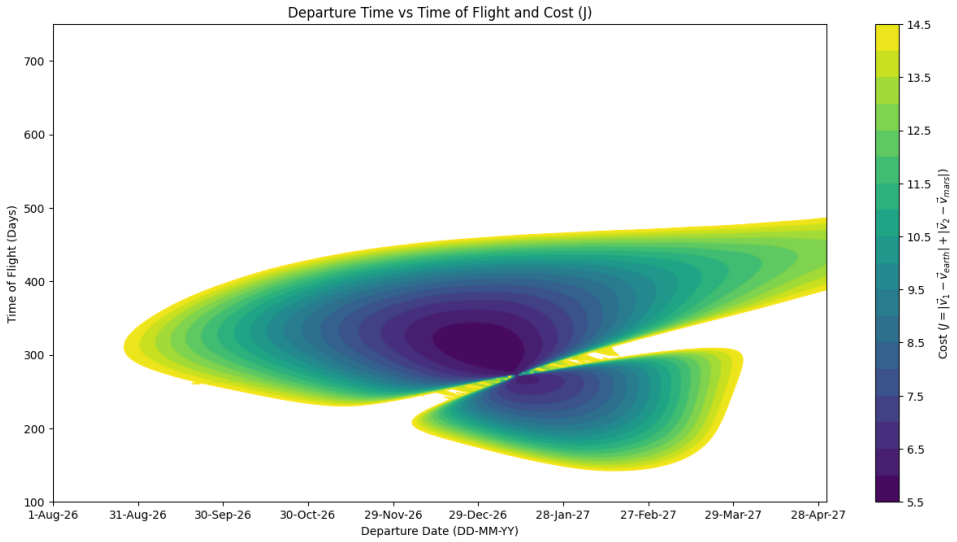


Figure 6. Porkchop Plot Showing the Ideal Launch Conditions

4 HELIOCENTRIC SELECTION AND PATCHED CONIC TRAJECTORY

Now that the transfer orbit has been constructed by minimizing delta v, we can continue with the patched conic approach, by constructing our Earth and Mars hyperbolic orbits. We can do this by finding $\vec{r}_1, \vec{r}_2, \vec{v}_1, \vec{v}_2, \vec{v}_{earth}, \vec{v}_{mars}$ in the same method as they were calculated in **Section 3**. From those values, we can find where the S/C must arrive, either behind or in front of mars, by projected $\vec{v}_2$ onto $\vec{v}_{mars}$, and comparing their magnitudes. What we see from this is that the spacecraft has a larger magnitude of velocity, so the **S/C must arrive behind Mars**. The next step, the hyperbolic velocities for both the earth and mars can be calculated using **Equations 2 and 3**.

$$\vec{v}_{\infty/earth} = \vec{v}_1 - \vec{v}_{earth} \tag{2}$$

$$\vec{v}_{\infty/mars} = \vec{v}_2 - \vec{v}_{mars} \tag{3}$$

The delta v will occur tangentially at the point of perigee on both orbits, and the velocity at the point of perigee for both hyperbolas can be computed from **Equations 4 and 5**, and they can be visualized in **Figures 9-12**.

$$v_{in/earth} = \sqrt{v_{\infty/earth}^2 + \frac{2\mu_{earth}}{r_{p_{earth}}}} \tag{4}$$

$$v_{in/mars} = \sqrt{v_{\infty/mars}^2 + \frac{2\mu_{mars}}{r_{p_{mars}}}} \tag{5}$$

The velocities of the S/C before the delta v burn on earth and after the delta v burn on mars are given by **Equations 6 and 7**.

$$v_{p_{earth}} = \sqrt{\frac{\mu_{earth}}{r_{p_{earth}}}} \tag{6}$$

$$v_{p_{mars}} = \sqrt{2\mu_{mars}\left(\frac{1}{r_{p_{mars}}} - \frac{1}{r_{p_{mars}} + r_{a_{mars}}}\right)} \tag{7}$$

Since delta v is occurring tangentially, delta v is as simple as taking the difference between the two known velocities for both earth and mars, as seen in **Equations 8 and 9**.

$$\Delta v_1 = v_{in/earth} - v_{p_{earth}} = \sqrt{v_{\infty/earth}^2 + \frac{2\mu_{earth}}{r_{p_{earth}}}} - \sqrt{\frac{\mu_{earth}}{r_{p_{earth}}}} \tag{8}$$

$$\Delta v_2 = v_{in/mars} - v_{p_{mars}} = \sqrt{v_{\infty/mars}^2 + \frac{2\mu_{mars}}{r_{p_{mars}}}} - \sqrt{2\mu_{mars}\left(\frac{1}{r_{p_{mars}}} - \frac{1}{r_{p_{mars}} + r_{a_{mars}}}\right)} \tag{9}$$

Total delta v is found by adding the two resultant burns, as shown in **Equation 10**.

$$\Delta v = \Delta v_1 + \Delta v_2 \tag{10}$$

Using this equation, we can iterate through a range of TOF's with the lone departure epoch of 14-Nov-26, and find the minimum delta v. This analysis is represented in **Figure 7**, and the minimized delta v is 5.0148 km/s with a TOF of 303 days, arriving on 13-Sep-27.

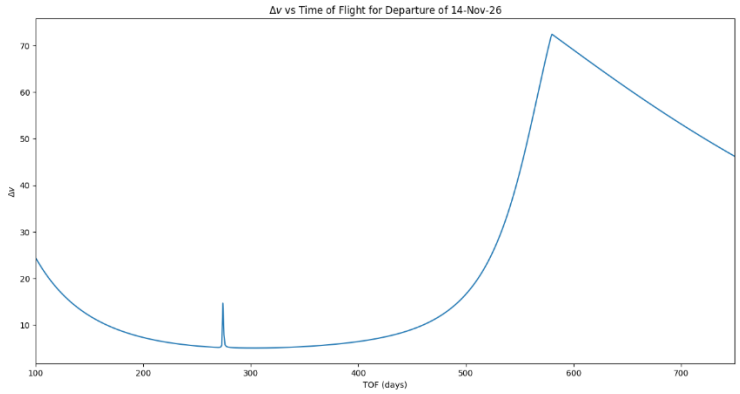


Figure 7. Delta v vs TOF for 14-Nov-26

After finding all the minimum delta v and TOF, the trajectories were plotting in both 2d and 3d, and represented in **Figures 8-13**.

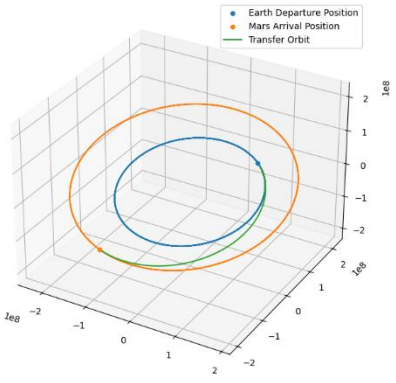


Figure 8. 3D Transfer Orbit Plot (Sun Centered Frame)

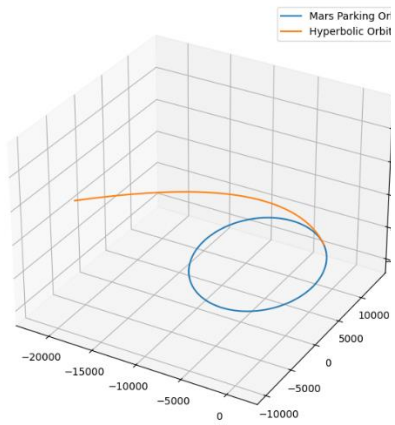


Figure 9. 3D Mars Hyperbolic Orbit (Mars Centered)

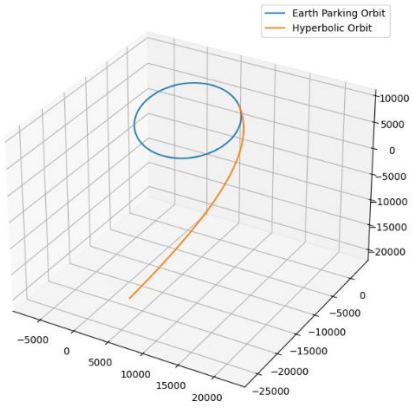


Figure 10. 3D Earth Hyperbolic Orbit (Earth Centered)

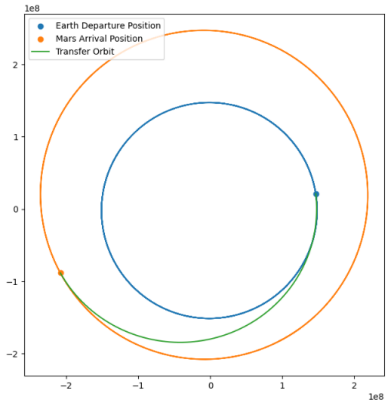


Figure 11. 2D Transfer Orbit Plot (Sun Centered)

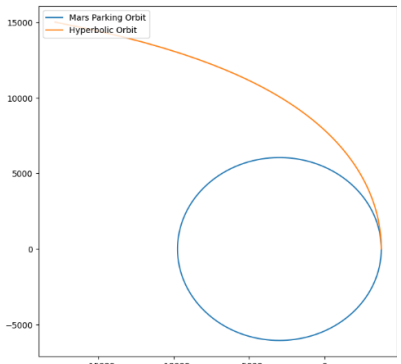


Figure 12. 2D Mars Hyperbolic Orbit (Mars Centered)

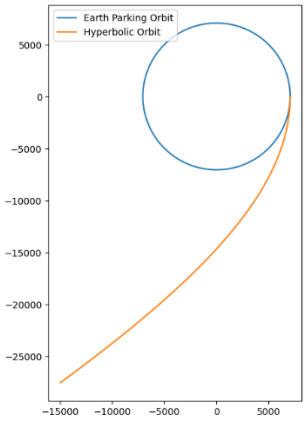


Figure 13. 2D Earth Hyperbolic Orbit (Earth Centered

5 CONCLUSIONS

In conclusion, the comprehensive study of Earth-Mars transfer opportunities undertaken in this project has yielded valuable insights into optimal departure dates and time of flight scenarios. The contour plot generated showcases the interplay between departure dates, time of flight, and the associated transfer costs, revealing distinct regions of efficiency in the Earth-Mars transfer trajectory. The color-coded contour lines provide a visual representation of how the cost of a mission changes. Notably, the results highlight specific departure dates and time of flight combinations that offer low-cost transfer options, laying the foundation for strategic mission planning. Specifically, the date found as the most optimal day to leave, was 1-Nov-26, with an associated arrival on 7-Sep-27, a TOF of 310 days, and a minimum cost of 5.1628 km/s.

Additionally, the application of the patched conics approximation, coupled with considerations of orbital parameters and fuel efficiency, adds a realistic dimension to the study, ensuring the practicality of the identified transfer opportunities. With the epoch date of 14-Nov-26, the ideal TOF was found to be 303 days, arriving on 13-Sep-27, with a minimized delta v of 5.0148 km/s.

The solution we found using patched conics, is by no means exact, it is a good approximation, but will not be accurate enough to fully drive a mission. To get more exact results, additional fidelity will be required, and other factors need to be considered, such as 3rd-body Perturbations and atmospheric Perturbations. Additionally, this approach assumes that all burns all instantaneous, whereas this is not possible, so finding a good way to model this would be a great way to increase fidelity of this problem.

6 Appendix

```
# Function Definitions

import datetime
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy
from scipy.spatial.transform import Rotation as R
import math

def mag(x):
    return (sum(i**2 for i in x))**.5

def M_to_E(M,e):
    E = M
    while True:
        E_i = E-(E-e*np.sin(E)-M)/(1-e*np.cos(E))
        if abs(E_i - E) < 10**-6:
            return E_i
        else:
            E = E_i

def E_to_M(E,e):
    return E-e*np.sin(E)

def E_to_F(E,e):
    return 2*np.arctan2((1+e)**.5*np.tan(E/2),(1-e)**.5)

def F_to_E(f,e):
    return 2*np.arctan2((1-e)**.5*np.tan(f/2),(1+e)**.5)

def time_convert(date):
    fmt = "%m.%d.%Y"
    date = date.replace(" ",".")
    if "Jan" in date:
        date = date.replace("Jan","01")
    elif "Feb" in date:
        date = date.replace("Feb","02")
    elif "Mar" in date:
        date = date.replace("Mar","03")
    elif "Apr" in date:
        date = date.replace("Apr","04")
    elif "May" in date:
        date = date.replace("May","05")
    elif "Jun" in date:
        date = date.replace("Jun","06")
    elif "Jul" in date:
        date = date.replace("Jul","07")
    elif "Aug" in date:
        date = date.replace("Aug","08")
    elif "Sep" in date:
        date = date.replace("Sep","09")
    elif "Oct" in date:
        date = date.replace("Oct","10")
    elif "Nov" in date:
        date = date.replace("Nov","11")
    elif "Dec" in date:
        date = date.replace("Dec","12")
    t = datetime.datetime.strptime(date, fmt)
```

```
    return pd.Timestamp(year=t.year, month=t.month, day=t.day, hour = t.hour, minute = t.minut
e ,second=t.second, microsecond=t.microsecond)

# Constants

AU = 1.495978707*10**8

mu_earth = 398600.44
mu_sun = 1.3271244*10**11
mu_mars = 42828.374

Earth_Radius = 6378
Sun_Radius = 695700
Mars_Radius = 6792/2

rc_earth = 1 * AU
rc_sun = 0
rc_venus = 0.7233 * AU
rc_mars = 1.524 * AU

date_start = time_convert("Jun 1 2026")

EphemerisData_DF = pd.read_csv("EphemerisData.csv")

Earth_X = EphemerisData_DF['Earth X'].values
Earth_Y = EphemerisData_DF['Earth Y'].values
Earth_Z = EphemerisData_DF['Earth Z'].values

Earth_Position = np.transpose(np.array([Earth_X,Earth_Y,Earth_Z]))

Mars_X = EphemerisData_DF['Mars X'].values
Mars_Y = EphemerisData_DF['Mars Y'].values
Mars_Z = EphemerisData_DF['Mars Z'].values

Mars_Position = np.transpose(np.array([Mars_X,Mars_Y,Mars_Z]))

Earth_Xdot = EphemerisData_DF['Earth Xdot'].values
Earth_Ydot = EphemerisData_DF['Earth Ydot'].values
Earth_Zdot = EphemerisData_DF['Earth Zdot'].values

Earth_Velocity = np.transpose(np.array([Earth_Xdot,Earth_Ydot,Earth_Zdot]))

Mars_Xdot = EphemerisData_DF['Mars Xdot'].values
Mars_Ydot = EphemerisData_DF['Mars Ydot'].values
Mars_Zdot = EphemerisData_DF['Mars Zdot'].values

Mars_Velocity = np.transpose(np.array([Mars_Xdot,Mars_Ydot,Mars_Zdot]))

def Lamberts_Problem(r1,r2,mu,t,Short_Path=True):
    c = mag(r2-r1)
    s = (mag(r1)+mag(r2)+c)/2
    if Short_Path==True: # Type 1
        TOF_p = 1/3*(2/mu)**.5*(s**(3/2)-(s-c)**(3/2))
    else: # Type 2
        TOF_p = 1/3*(2/mu)**.5*(s**(3/2)+(s-c)**(3/2))
    if t > TOF_p: # Ellipse
        am = s/2
        alpha_m = 2*np.arcsin((s/2/am)**0.5)
        beta_m = 2*np.arcsin(((s-c)/2/am)**0.5)
        TOF_min = (am**3/mu)**.5*(alpha_m-beta_m-(np.sin(alpha_m)-np.sin(beta_m)))
        if Short_Path==True: # Type 1
            if t < TOF_min: # Type 1A
                def f(x):
```

```
                    alpha0 = 2*np.arcsin((s/2/x)**0.5)
                    beta0 = 2*np.arcsin(((s-c)/2/x)**0.5)
                    return t-(x**3/mu)**0.5*((alpha0-np.sin(alpha0)-(beta0-np.sin(beta0))))
                a = scipy.optimize.fsolve(f,am+10)[0]
                return [a,2*np.arcsin((s/2/a)**0.5),2*np.arcsin(((s-c)/2/a)**0.5)]
            else: # Type 1B
                def f(x):
                    alpha0 = 2*np.pi-2*np.arcsin((s/2/x)**0.5)
                    beta0 = 2*np.arcsin(((s-c)/2/x)**0.5)
                    return t-(x**3/mu)**0.5*((alpha0-np.sin(alpha0)-(beta0-np.sin(beta0))))
                a = scipy.optimize.fsolve(f,am+10)[0]
                return [a,2*np.pi-2*np.arcsin((s/2/a)**0.5),2*np.arcsin(((s-c)/2/a)**0.5)]
        else: # Type 2
            if t < TOF_min: # Type 2A
                def f(x):
                    alpha0 = 2*np.arcsin((s/2/x)**0.5)
                    beta0 = -2*np.arcsin(((s-c)/2/x)**0.5)
                    return t-(x**3/mu)**0.5*((alpha0-np.sin(alpha0)-(beta0-np.sin(beta0))))
                a = scipy.optimize.fsolve(f,am+10)[0]
                return [a,2*np.arcsin((s/2/a)**0.5),-2*np.arcsin(((s-c)/2/a)**0.5)]
            else: # Type 2B
                def f(x):
                    alpha0 = 2*np.pi-2*np.arcsin((s/2/x)**0.5)
                    beta0 = -2*np.arcsin(((s-c)/2/x)**0.5)
                    return t-(x**3/mu)**0.5*((alpha0-np.sin(alpha0)-(beta0-np.sin(beta0))))
                a = scipy.optimize.fsolve(f,am+10)[0]
                return [a,2*np.pi-2*np.arcsin((s/2/a)**0.5),-2*np.arcsin(((s-c)/2/a)**0.5)]
    else: # Hyperbolic
        return [False]

def Position_and_TOF_to_Velocity(r1,r2,TOF,mu,path):
    ans = Lamberts_Problem(r1,r2,mu,TOF,Short_Path=path)
    if len(ans) > 1:
        a = ans[0]
        alpha = ans[1]
        beta = ans[2]
        c = mag(r2-r1)
        A = (mu/4/a)**.5/np.tan(alpha/2)
        B = (mu/4/a)**.5/np.tan(beta/2)
        u1 = r1/mag(r1)
        u2 = r2/mag(r2)
        uc = (r2-r1)/c
        v1 = (B+A)*uc+(B-A)*u1
        v2 = (B+A)*uc-(B-A)*u2
    else:
        return False,False
    return v1,v2

# Calculating optimal cost (J) for a range of departure dates and TOFs

days = 273
start_day = 61
Departure_Day = np.linspace(start_day,days+start_day,days+1)
TOF = np.linspace(100,750,651)

Departure_Day_list = []
Arrival_Day_list = []
v1_list = []
v2_list = []
TOF_list = []
dv_list = []

for i in Departure_Day:
    # print("Departure_Day:",i)
```

```
    for j in TOF:
        # print("TOF:",j)
        v1s = []
        v2s = []
        J = []
        for Path in [True,False]:
            i = int(i)
            j = int(j)
            r1 = Earth_Position[i] # Earth Departure Position
            r2 = Mars_Position[i+j] # Mars Arrival Position
            ve = Earth_Velocity[i]
            vm = Mars_Velocity[i+j]
            v1,v2 = Position_and_TOF_to_Velocity(r1,r2,j*3600*24,mu_sun,Path)
            if type(v1) != bool and math.isnan(v1[0]) != True:
                v1s.append(v1)
                v2s.append(v2)
            # else:
            #     print("Hyperbole")
            # v1,v2 = lamberthub.izzo2015(mu_sun,r1,r2,j*3600*24,low_path=Path, maxiter=50, at
ol=1e-5, rtol=1e-7, full_output=False)
            J.append(mag(v1-ve)+mag(v2-vm))
        if len(v1s) > 0:
            # print(len(v1s))
            Departure_Day_list.append(i)
            Arrival_Day_list.append(i+j)
            v1_list.append(np.array(v1s)[J.index(min(J))])
            v2_list.append(np.array(v2s)[J.index(min(J))])
            TOF_list.append(j)
            dv_list.append(min(J))
```

```
C:\Users\jjfoo\AppData\Local\Temp\ipykernel_13456\1167677246.py:16: RuntimeWarning: invalid va
lue encountered in arcsin
  alpha0 = 2*np.arcsin((s/2/x)**0.5)
C:\Users\jjfoo\AppData\Local\Programs\Python\Python312\Lib\site-packages\scipy\optimize\_minpa
ck_py.py:177: RuntimeWarning: The iteration is not making good progress, as measured by the
  improvement from the last ten iterations.
  warnings.warn(msg, RuntimeWarning)
C:\Users\jjfoo\AppData\Local\Temp\ipykernel_13456\1167677246.py:38: RuntimeWarning: invalid va
lue encountered in arcsin
  alpha0 = 2*np.pi-2*np.arcsin((s/2/x)**0.5)
C:\Users\jjfoo\AppData\Local\Temp\ipykernel_13456\1167677246.py:23: RuntimeWarning: invalid va
lue encountered in arcsin
  alpha0 = 2*np.pi-2*np.arcsin((s/2/x)**0.5)
C:\Users\jjfoo\AppData\Local\Temp\ipykernel_13456\1167677246.py:31: RuntimeWarning: invalid va
lue encountered in arcsin
  alpha0 = 2*np.arcsin((s/2/x)**0.5)
```

```
min_index = np.where(dv_list == min(dv_list))[0].data[0]
print("Minimizing Delta V Spending:")
print("Leaving at:",EphemerisData_DF["Epoch"][Departure_Day_list[min_index]],"number=",Departu
re_Day_list[min_index])
print("Arriving at:",EphemerisData_DF["Epoch"][Arrival_Day_list[min_index]],"number=",Arrival_
Day_list[min_index])
print("TOF: ",TOF_list[min_index])
print("Delta v:",dv_list[min_index])
```

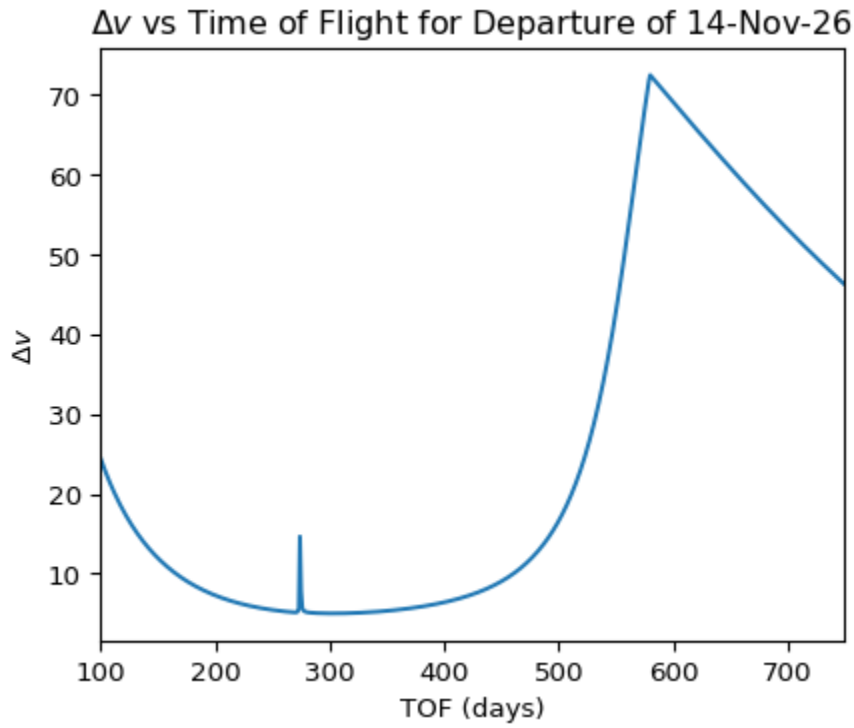
```
Minimizing Delta V Spending:
Leaving at: 1-Nov-26 number= 214
Arriving at: 7-Sep-27 number= 524
TOF:  310
Delta v: 5.612823726806169
```

```
# Calculating optimal dv for a departure date of Nov-14-26

days = 273
```

```
start_day = 61
Departure_Day = [227]
TOF = np.linspace(100,750,651)

x = 24
y = 14

rc_earth = 10*x+450+Earth_Radius
rp_mars = 15*y+150+Mars_Radius
ra_mars = 275*y+2500+Mars_Radius

Departure_Day_list = []
Arrival_Day_list = []
v1_list = []
v2_list = []
TOF_list = []
dv_list = []

for i in Departure_Day:
    # print("Departure_Day:",i)
    for j in TOF:
        # print("TOF:",j)
        v1s = []
        v2s = []
        dv1_list = []
        dv2_list = []
        for Path in [True,False]:
            i = int(i)
            j = int(j)
            r1 = Earth_Position[i] # Earth Departure Position
            r2 = Mars_Position[i+j] # Mars Arrival Position
            ve = Earth_Velocity[i]
            vm = Mars_Velocity[i+j]
            v1,v2 = Position_and_TOF_to_Velocity(r1,r2,j*3600*24,mu_sun,Path)
            if type(v1) != bool and math.isnan(v1[0]) != True:
                v1s.append(v1)
                v2s.append(v2)
                vinf_e = v1-ve
                vinf_m = v2-vm
                delta_v1 = (mag(vinf_e)**2+2*mu_earth/rc_earth)**.5-(mu_earth/rc_earth)**.5
                delta_v2 = (mag(vinf_m)**2+2*mu_mars/rp_mars)**.5-(mu_mars*(2/rp_mars-2/(rp_ma
rs+ra_mars)))**.5
                dv1_list.append(delta_v1)
                dv2_list.append(delta_v2)
        if len(dv2_list) > 0:
            # print(len(v1s))
            dv_tot = np.array(dv1_list)+np.array(dv2_list)
            Departure_Day_list.append(i)
            Arrival_Day_list.append(i+j)
            index=list(np.where(dv_tot == min(dv_tot))[0])
            v1_list.append(np.array(v1s)[index])
            v2_list.append(np.array(v2s)[index])
            TOF_list.append(j)
            dv_list.append(min(dv_tot))

min_index = np.where(dv_list == min(dv_list))[0].data[0]
print("Minimizing Delta V Spending:")
print("Leaving at:",EphemerisData_DF["Epoch"][Departure_Day_list[min_index]],"number=",Departu
re_Day_list[min_index])
print("Arriving at:",EphemerisData_DF["Epoch"][Arrival_Day_list[min_index]],"number=",Arrival_
Day_list[min_index])
print("TOF: ",TOF_list[min_index])
print("Delta v:",dv_list[min_index])
```

```
Minimizing Delta V Spending:
Leaving at: 14-Nov-26 number= 227
Arriving at: 13-Sep-27 number= 530
TOF:  303
Delta v: 5.014819215708099
```

```
X = Departure_Day_list
Y = TOF_list
Z = dv_list

X_labels = np.array(EphemerisData_DF["Epoch"][Departure_Day_list])[list(np.where(np.array(Z) <
 15)[0])]

dv_lim = 100

levels = np.arange(5.5,dv_lim,0.5)

mycmap = plt.get_cmap('rainbow')

X_plot = np.array(X)[list(np.where(np.array(Z) < dv_lim)[0])]
Y_plot = np.array(Y)[list(np.where(np.array(Z) < dv_lim)[0])]
Z_plot = np.array(Z)[list(np.where(np.array(Z) < dv_lim)[0])]

fig = plt.figure(r"$\Delta v$ vs Time of Flight for Departure of 14-Nov-26")
ax = plt.axes()
plt.plot(Y_plot,Z_plot)
plt.title(r"$\Delta v$ vs Time of Flight for Departure of 14-Nov-26")
plt.xlim([100,750])
plt.xlabel(f"TOF (days)")
plt.ylabel(r"$\Delta v$")
plt.show()
```

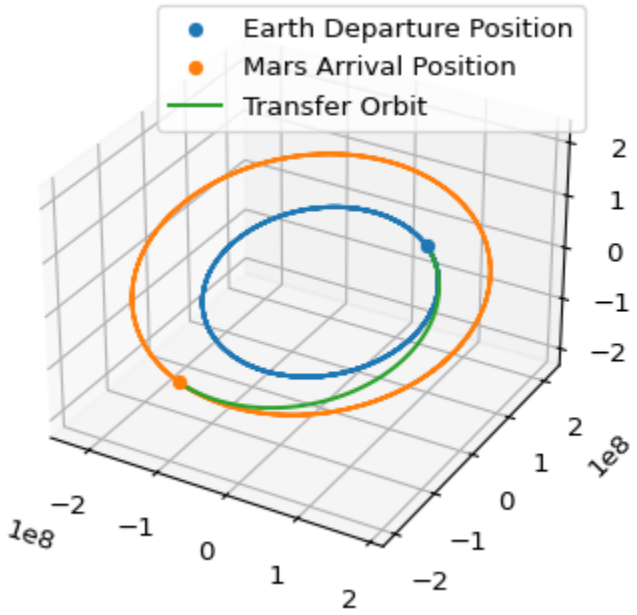


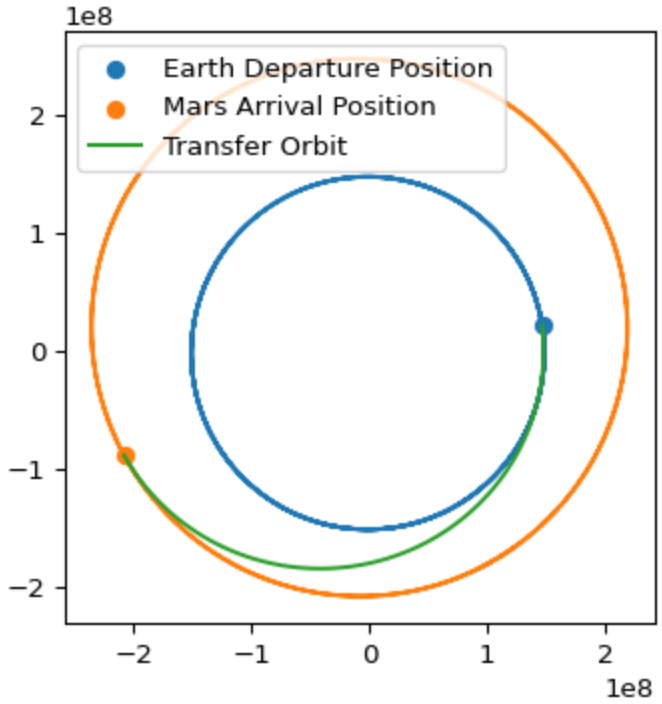

```
r1 = Earth_Position[Departure_Day_list[min_index]]
ve = Earth_Velocity[Departure_Day_list[min_index]]
v1 = v1_list[min_index][0]
r2 = Mars_Position[Arrival_Day_list[min_index]]
```

```
vm = Mars_Velocity[Arrival_Day_list[min_index]]
v2 = v2_list[min_index][0]
vc_earth_mag = (mu_sun/(rc_earth))**.5 #Magnitude of Earth velocity
vc_mars_mag = (mu_sun/(rc_mars))**.5 #Magnitude of Mars velocity

print("r1 =",r1)
print("ve =",ve)
print("v1 =",v1)
print("r2 =",r2)
print("vm =",vm)
print("v2 =",v2)

if np.dot(v2,vm)/mag(v2) > mag(vm):
    print("S/C must arrive behind mars")
else:
    print("S/C must arrive behind mars")

Et = mag(v1)**2/2-mu_sun/mag(r1)
ht = np.cross(r1,v1)
at = -mu_sun/(2*Et)
et = np.cross(v1,ht)/mu_sun-r1/mag(r1)

et_unit = et/mag(et)
ht_unit = ht/mag(ht)
pt_unit = np.cross(ht_unit,et_unit)

f1 = np.arctan2(np.dot(r1,pt_unit),np.dot(r1,et_unit))
f2 = np.arctan2(np.dot(r2,pt_unit),np.dot(r2,et_unit))

E1 = F_to_E(f1,mag(et))
E2 = F_to_E(f2,mag(et))

M1 = E_to_M(E1,mag(et))
M2 = E_to_M(E2,mag(et))

tof = (M1-M2)*(at**3/mu_sun)**.5
print("TOF Calculated =",tof)
print("TOF Expected =",TOF_list[min_index]*24*3600)
print("|%| Difference =",abs(tof-TOF_list[min_index]*24*3600)/(TOF_list[min_index]*24*3600)*10
0)

vinf_e = v1-ve
vinf_m = v2-vm

print("vinf_e =",vinf_e)
print("vinf_m =",vinf_m)

x = 24
y = 14

rc_earth = 10*x+450+Earth_Radius
rp_mars = 15*y+150+Mars_Radius
ra_mars = 275*y+2500+Mars_Radius

delta_v1 = (mag(vinf_e)**2+2*mu_earth/rc_earth)**.5-(mu_earth/rc_earth)**.5
# delta_v2 = (mag(vinf_m)**2+2*mu_mars/rp_mars)**.5-(mu_mars/rp_mars)**.5
delta_v2 = (mag(vinf_m)**2+2*mu_mars/rp_mars)**.5-(mu_mars*(2/rp_mars-2/(rp_mars+ra_mars)))**.
5

total_dv = delta_v1 + delta_v2
print("total_dv =",total_dv)
```

```
r1 = [9.26556836e+07 1.05926643e+08 4.59167382e+07]
ve = [-23.72704244  17.00217341   7.36937508]
v1 = [-25.21595329  19.09795249   9.58274311]
r2 = [-9.14951171e+07 -1.88738688e+08 -8.41035519e+07]
vm = [23.06605482 -6.80126949 -3.74165057]
v2 = [20.72961082 -5.77184433 -3.30395818]
S/C must arrive behind mars
TOF Calculated = 18786807.711606234
TOF Expected = 26179200
|%| Difference = 28.237655422601783
vinf_e = [-1.48891085  2.09577908  2.21336803]
vinf_m = [-2.336444    1.02942516  0.4376924 ]
total_dv = 5.014819215708099

def set_axes_equal(x,y,z,ax):
    """
    Make axes of 3D plot have equal scale so that spheres appear as spheres,
    cubes as cubes, etc.

    Input
      ax: a matplotlib axis, e.g., as output from plt.gca().
    """

    x_limits = [max(x), min(x)]
    y_limits = [max(y), min(y)]
    z_limits = [max(z), min(z)]

    x_range = abs(x_limits[1] - x_limits[0])
    x_middle = np.mean(x_limits)
    y_range = abs(y_limits[1] - y_limits[0])
    y_middle = np.mean(y_limits)
    z_range = abs(z_limits[1] - z_limits[0])
    z_middle = np.mean(z_limits)

    # The plot bounding box is a sphere in the sense of the infinity
    # norm, hence I call half the max range the plot radius.
    plot_radius = 0.5*max([x_range, y_range, z_range])

    ax.set_xlim3d([x_middle - plot_radius, x_middle + plot_radius])
    ax.set_ylim3d([y_middle - plot_radius, y_middle + plot_radius])
    ax.set_zlim3d([z_middle - plot_radius, z_middle + plot_radius])

from scipy.spatial.transform import Rotation as R

Initial_x = EphemerisData_DF['Earth X'][Departure_Day_list[min_index]]
Initial_y = EphemerisData_DF['Earth Y'][Departure_Day_list[min_index]]
Initial_z = EphemerisData_DF['Earth Z'][Departure_Day_list[min_index]]

Initial_pos = np.array([Initial_x,Initial_y,Initial_z])

Final_x = EphemerisData_DF['Mars X'][Arrival_Day_list[min_index]]
Final_y = EphemerisData_DF['Mars Y'][Arrival_Day_list[min_index]]
Final_z = EphemerisData_DF['Mars Z'][Arrival_Day_list[min_index]]

Final_pos = np.array([Final_x,Final_y,Final_z])

i = np.arccos(ht_unit[2])
w = np.arctan2(ht_unit[0],ht_unit[1])
o = np.arctan2(et_unit[2],-pt_unit[2])

# fig = plt.figure()
# ax = plt.axes(projection='3d')
```

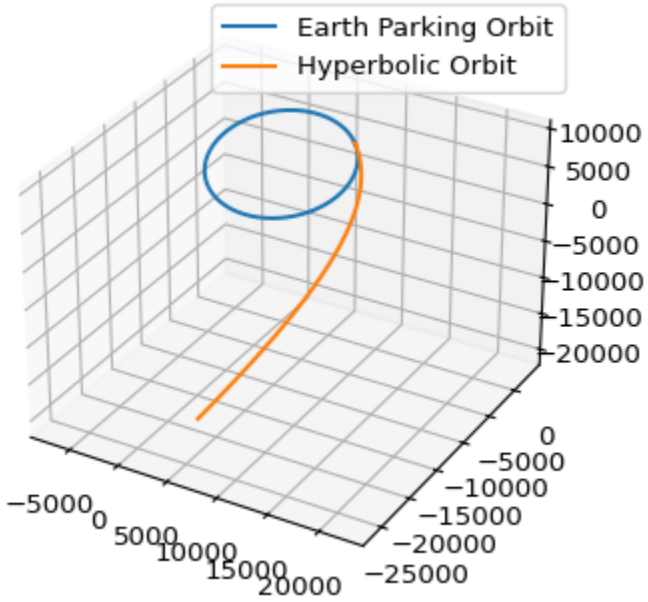
```
T = np.transpose(R.from_euler("ZXZ", [o, i, w], degrees=False).as_matrix())

Initial_pos_equi = np.array(np.linalg.inv(T)@Initial_pos)
Final_pos_equi = np.array(np.linalg.inv(T)@Final_pos)

Earth_Position_equi = np.array([np.linalg.inv(T)@i for i in Earth_Position])
Mars_Position_equi = np.array([np.linalg.inv(T)@i for i in Mars_Position])

pos_icrf = []
fs = np.linspace(f1,f2,100)
for f in fs:
    radius = at*(1-mag(et)**2)/(1+mag(et)*np.cos(f))
    pos_icrf.append(T@(radius*np.array([np.cos(f),np.sin(f),0])))

pos_icrf = np.array(pos_icrf)
pos_equi = np.array([np.linalg.inv(T)@i for i in pos_icrf])

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot(Earth_Position[0:,0],Earth_Position[0:,1],Earth_Position[0:,2])
ax.plot(Mars_Position[0:,0],Mars_Position[0:,1],Mars_Position[0:,2])
ax.scatter(Initial_pos[0],Initial_pos[1],Initial_pos[2],label='Earth Departure Position')
ax.scatter(Final_pos[0],Final_pos[1],Final_pos[2],label='Mars Arrival Position')
ax.plot(pos_icrf[0:,0],pos_icrf[0:,1],pos_icrf[0:,2],label='Transfer Orbit')
set_axes_equal(Mars_Position[0:,0],Mars_Position[0:,1],Mars_Position[0:,2],ax)
ax.legend()
plt.show()

fig = plt.figure()
ax = plt.axes()
ax.plot(Earth_Position_equi[0:,0],Earth_Position_equi[0:,1])
ax.plot(Mars_Position_equi[0:,0],Mars_Position_equi[0:,1])
ax.scatter(Initial_pos_equi[0],Initial_pos_equi[1],label='Earth Departure Position')
ax.scatter(Final_pos_equi[0],Final_pos_equi[1],label='Mars Arrival Position')
ax.plot(pos_equi[0:,0],pos_equi[0:,1],label='Transfer Orbit')
ax.set_aspect('equal', adjustable='box')
ax.legend(loc=2)
plt.show()
```

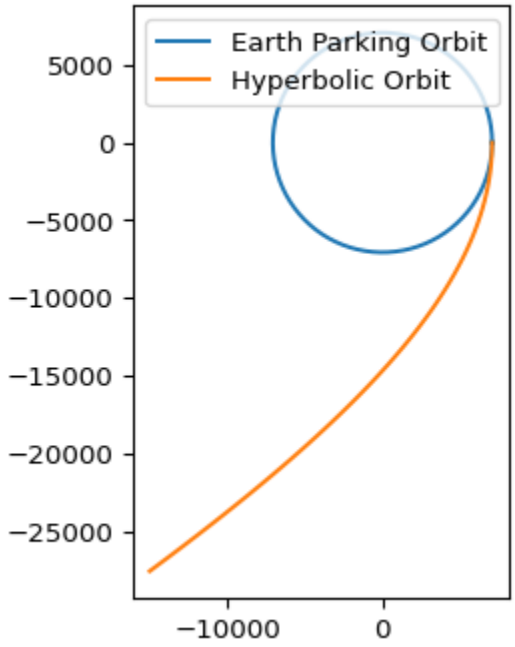





```
ae = rc_earth
ee = 0

ainf_e = -mu_earth/vinf_e**2
einf_e = 1-rc_earth/ainf_e

pe = ae*(1-ee**2)
pos_e_equi = [] ; pos_e_icrf = []
for f in np.linspace(0,2*np.pi,360):
    r = pe/(1+ee*np.cos(f))
    pos_e_equi.append(r*np.array([np.cos(f),np.sin(f),0]))
    pos_e_icrf.append(T@(r*np.array([np.cos(f),np.sin(f),0])))
pos_e_equi = np.array(pos_e_equi)
pos_e_icrf = np.array(pos_e_icrf)

pinf_e = ainf_e*(1-einf_e**2)
pos_inf_e_equi = [] ; pos_inf_e_icrf = []
for f in np.linspace(-120/180*np.pi,0,100):
    r = pinf_e/(1+einf_e*np.cos(f))
    pos_inf_e_equi.append(r*np.array([np.cos(f),np.sin(f),0]))
    pos_inf_e_icrf.append(T@(r*np.array([np.cos(f),np.sin(f),0])))
pos_inf_e_equi = np.array(pos_inf_e_equi)
pos_inf_e_icrf = np.array(pos_inf_e_icrf)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot(pos_e_icrf[0:,0],pos_e_icrf[0:,1],pos_e_icrf[0:,2],label="Earth Parking Orbit")
ax.plot(pos_inf_e_icrf[0:,0],pos_inf_e_icrf[0:,1],pos_inf_e_icrf[0:,2],label='Hyperbolic Orbit
')
set_axes_equal(pos_inf_e_icrf[0:,0],pos_inf_e_icrf[0:,1],pos_inf_e_icrf[0:,2],ax)
ax.legend()
plt.show()

plt.figure()
ax = plt.axes()
plt.plot(pos_e_equi[0:,0], pos_e_equi[0:,1],label="Earth Parking Orbit")
plt.plot(pos_inf_e_equi[0:,0], pos_inf_e_equi[0:,1],label='Hyperbolic Orbit')
```

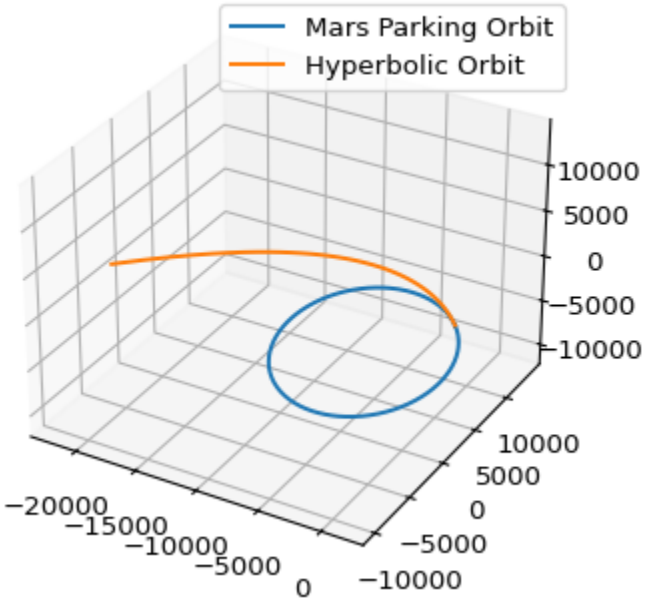
```
ax.set_aspect('equal', adjustable='box')
plt.legend(loc=2)
plt.show()
```

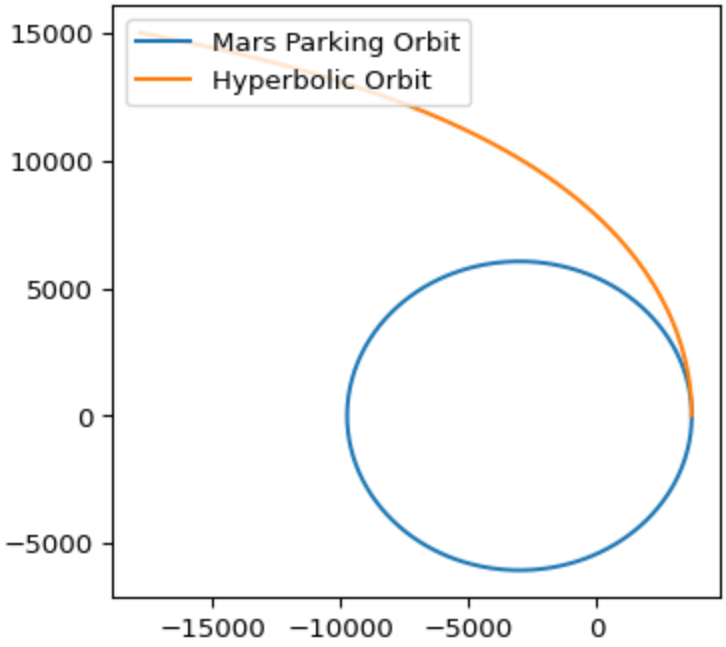





```
am = (rp_mars+ra_mars)/2
em = (ra_mars-rp_mars)/((rp_mars+ra_mars))

ainf_m = -mu_mars/vinf_m**2
einf_m = 1-rp_mars/ainf_m

pm = am*(1-em**2)
pos_m_equi = [] ; pos_m_icrf = []
for f in np.linspace(0,2*np.pi,360):
    r = pm/(1+em*np.cos(f))
    pos_m_equi.append(r*np.array([np.cos(f),np.sin(f),0]))
    pos_m_icrf.append(T@(r*np.array([np.cos(f),np.sin(f),0])))
```

```
pos_m_equi = np.array(pos_m_equi)
pos_m_icrf = np.array(pos_m_icrf)

pinf_m = ainf_m*(1-einf_m**2)
pos_inf_m_equi = [] ; pos_inf_m_icrf = []
for f in np.linspace(0,120/180*np.pi,100):
    r = pinf_m/(1+einf_m*np.cos(f))
    pos_inf_m_equi.append(r*np.array([np.cos(f),np.sin(f),0]))
    pos_inf_m_icrf.append(T@(r*np.array([np.cos(f),np.sin(f),0])))
pos_inf_m_equi = np.array(pos_inf_m_equi)
pos_inf_m_icrf = np.array(pos_inf_m_icrf)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot(pos_m_icrf[0:,0],pos_m_icrf[0:,1],pos_m_icrf[0:,2],label="Mars Parking Orbit")
ax.plot(pos_inf_m_icrf[0:,0],pos_inf_m_icrf[0:,1],pos_inf_m_icrf[0:,2],label='Hyperbolic Orbit
')
set_axes_equal(pos_inf_m_icrf[0:,0],pos_inf_m_icrf[0:,1],pos_inf_m_icrf[0:,2],ax)
ax.legend()
plt.show()

plt.figure()
ax = plt.axes()
plt.plot(pos_m_equi[0:,0], pos_m_equi[0:,1],label="Mars Parking Orbit")
plt.plot(pos_inf_m_equi[0:,0], pos_inf_m_equi[0:,1],label='Hyperbolic Orbit')
ax.set_aspect('equal', adjustable='box')
plt.legend(loc=2)
plt.show()
```




Mars Parking Orbit
Hyperbolic Orbit
15000
10000
5000
0
−5000
−15000
−10000
−5000
0