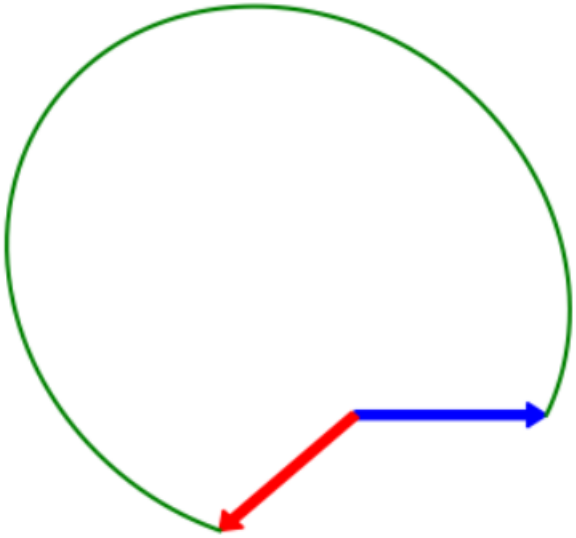
OZBAY Yasin | LHOEST Simon 09/01/2024

Course Project
Solving the Lambert problem



Summary

Introduction

In our modern era, space holds a central fascination for our population. Accessing space allows us to develop new tools and technologies that aid us in our daily lives. Satellites and communications stand out as its primary assets. Exploration and observation assist us in understanding the world and its origins. Achieving space travel is an incredible feat. Simply putting a rocket up to speed is insufficient for reaching space and other celestial objects. Meticulous calculations must be performed to ensure precise launches and the ability to reach any target, even for placing satellites into the desired orbit.

In this project, we tackle the Lambert problem. The solution to this problem involves finding a trajectory that connects two position vectors, $\mathbf{r_1}$ and $\mathbf{r_2}$, given a time-of-flight Δt. It finds applications in interplanetary transfers, orbital determination, orbital transfer, and ballistic missile targeting or rendezvous. Since Johann Henrich Lambert introduced this problem in the 18th century, more than 70 authors have proposed formulations and solutions. NASA currently employs the Gooding solution proposed in 1990, while ESA uses the Izzo solution formulated in 2015. Classical methods, such as the original Gauss method (1809) or its improved version (1971), also exist. For this project, we adopt the Simó method from 1973, leveraging Newton's gravitational equation of motion for the two-body problem, this method regularizes the equation by adopting appropriately selected variables, eliminating singularities.

Our developed tool aims to compute the ΔV budget for different departure and arrival dates. ΔV represents the velocity budget needed to execute the determined transfer orbit. The goal is to minimize ΔV, thereby reducing the required fuel necessary to fly the trajectory, leading to less weight for fuel and more for payload. We can visualize the ΔV budget by plotting a Pork Chop Plot (PCP), which displays the results depending on different launch dates and times of flight.

In this document, we will explain how we developed the Simó method to solve the Lambert problem. We will focus on the main function that our program contains. Subsequently, we will validate our code with examples of elementary cases. Following that, we will provide a complete PCP and ΔV budget for every planet, aiming to find the optimal time of launch in a window between 2020-2025. Finally, we will conclude by discussing what we have learned from the project and suggesting future improvements for the program.

Methodology

The Simó method Lambert problem solver is implemented in Python. All code is home-made except for the use of numpy for arrays and matplotlib for plotting results. The program is divided in 3 parts : The Lambert class file and its functions, the PlanetPosition file and the Project file, the main file used to execute the solver.

1. The Lambert class and its functions

The Lambert.py file represents the core of the Simó method resolution. It is in this file that all calculations are made. We take us of the class method built in Python to store data and apply function to it at the same time. Our Lambert object will have every variable registered to it and will use it in each function that we tell it to apply. We initialize a Lambert object by giving the Δt, the angle $\Delta\theta$, the number of revolutions, the Long or Short choice (LoS), the transfer direction, the standard gravitational parameter, and the two vector $\mathbf{r_1}$ and $\mathbf{r_2}$. When initializing the object, the program also processes the constant A, B , C, P and Q that are derived from the parameters given to the class.

Then, we add all the calculation functions to our file. We add every Stumpff functions from 0 to 5. We could add only one Stumpff function that process each by adding a number variable to get the desired one, but for reading ease we use 5 functions that follows :

$$c_{k+2}(x) = \frac{\frac{1}{k!} - c_k(x)}{x}$$

Afterwards, the formula to compute the transfer time and its derivative uses all our previous variable and function. The Lambert function *_getz()* makes use of the Newton first oder iterative method to find the closest matching z that matches the desired Δt. Having found z, we can compute the orbit parameters : semi-major axis a, eccentricity e and the semi latus rectum p. With the orbit parameters calculated we can proceed and compute the ΔV for the points r_1 and r_2.

With all this data we can make our plots. We redefine the print function of our Lambert object by defining the function *__str__()*. We do two different plots. The first one is the plot of the computed trajectory, linking the points r_1 and r_2. The second plot is the z with respect to Δt plot. This second plot also highlights the z value calculated by our program corresponding to the given Δt. The *_print_3d()* function is the one used to print the trajectory in 3d. This function is used when calculating the trajectory in the case of planets. It also highlights the planets orbits.

2. PlanetPosition file

The PlanetPosition file is the file derived from assignment 1 to be used in our project. It computes two planets position, one at the given date and the other one at the date + Δt. The *PP()* function uses all the other function in the file to compute each planet position and velocity. The *GP()* function has been added to retrieve the dictionary containing the orbital elements and their derivative of each planet. This dictionary is used to simply request the data in our main file, making switching between planets easier and clearer.

3. The main file Project

This file is the one that brings the last two together. The only function of this file : *Project()*, is used to get all the data of a Lambert object. The rest of the file are examples on how to set our variables, use the function and print our results. For the 3d printing, the file makes use of PlanetPosition.py, first by getting the desired planets and then by computing their position and velocities. Lastly, the file uses all the functions to calculate the PCP on determined dates and velocities.

Results

In this next part we will assess the results given by the program

1. Elementary cases

First things first, we will put our program to the test with elementary cases. We will use 2d vectors to prove its ability to compute the trajectory between $\mathbf{r_1}$ and $\mathbf{r_2}$. We will compare the tests with the results shown in class, to confirm that the program gives us correct answers.

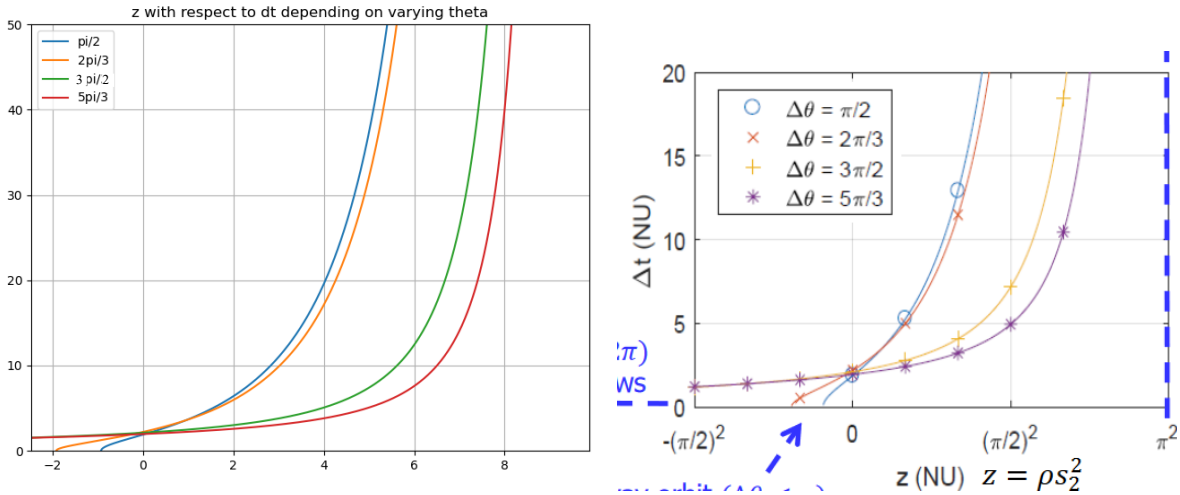


Figure 1

This first test shows us that the program can calculate Δt, so our formula is correct and well implemented. We now see if it handles the revolution :

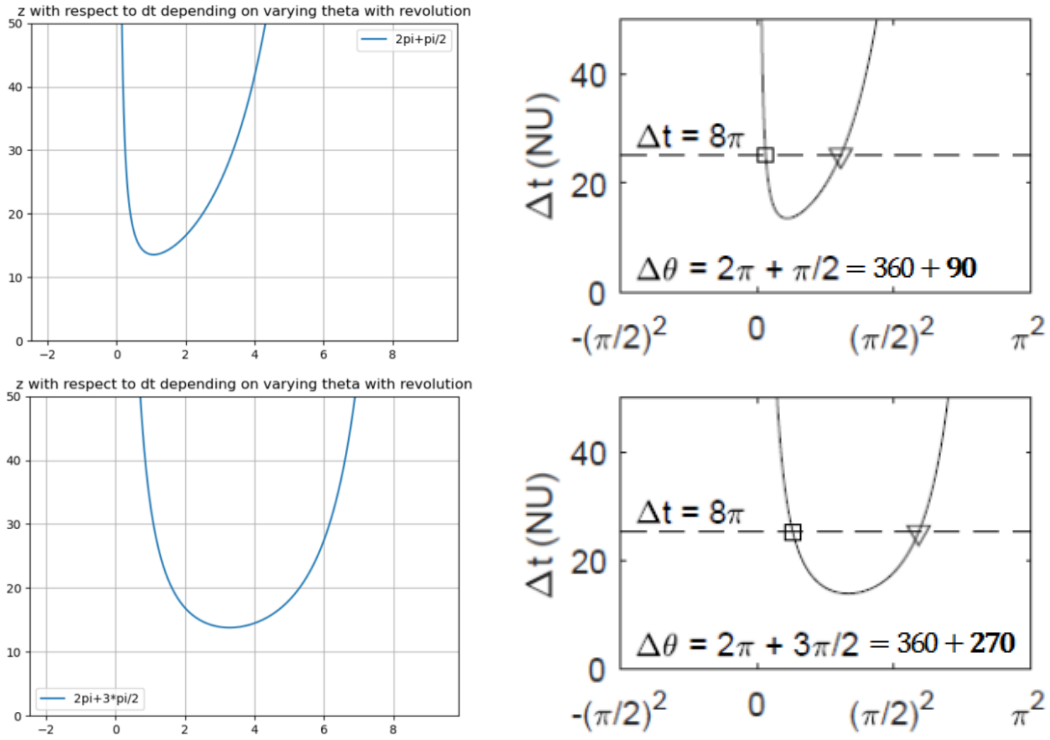


Figure 2

Again, we demonstrate that the program gives us correct calculations, as it is shown by the figure 2.

In the next test, we demonstrate how our program can select between long or short way as well as counterclockwise or clockwise rotation direction. We again compare our results with the ones given in class, first with an angle of 90° then 270° :

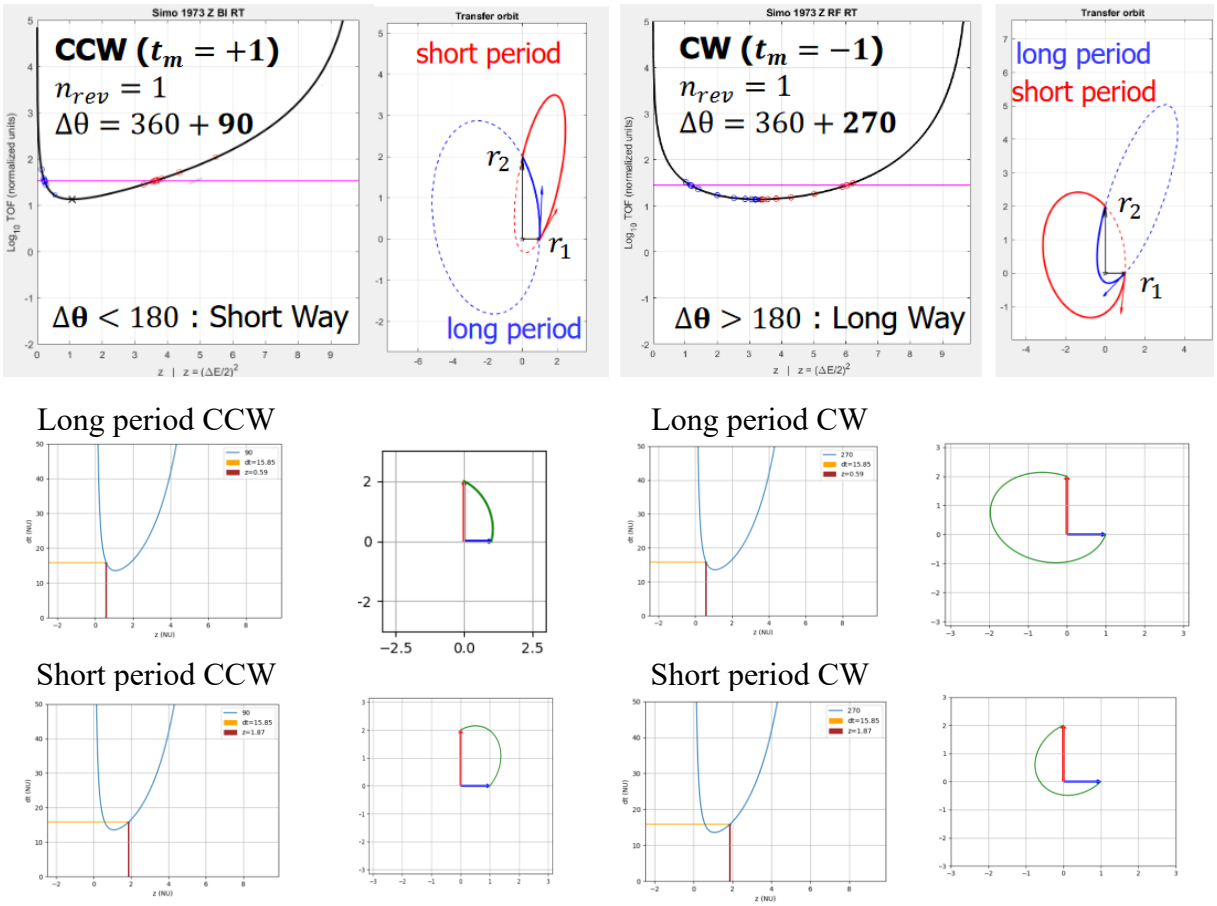
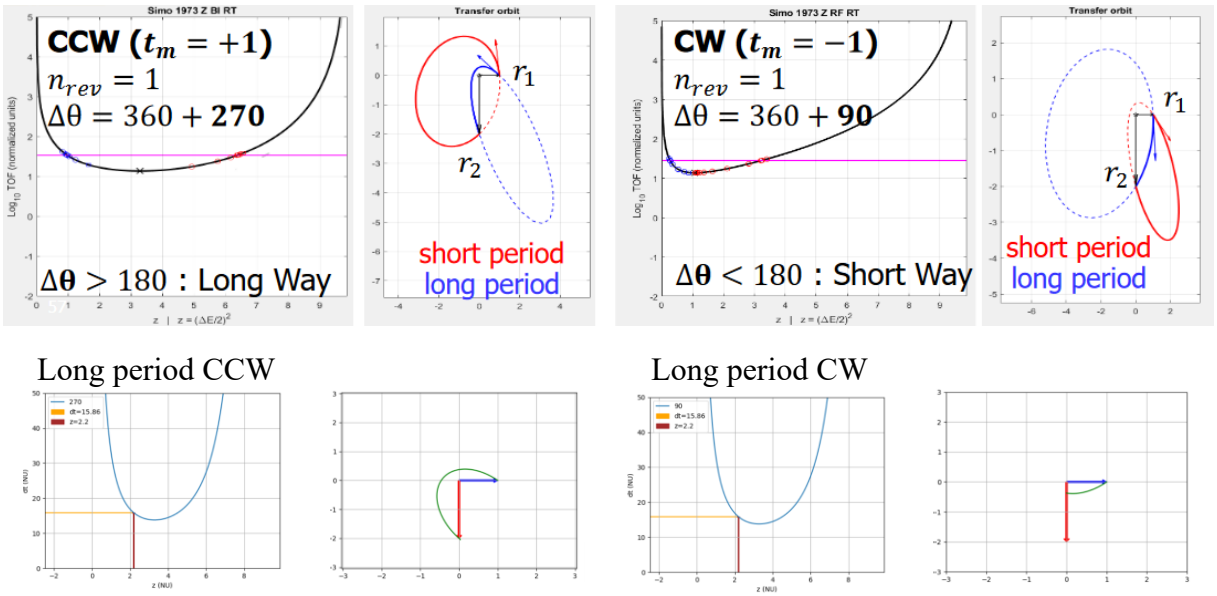
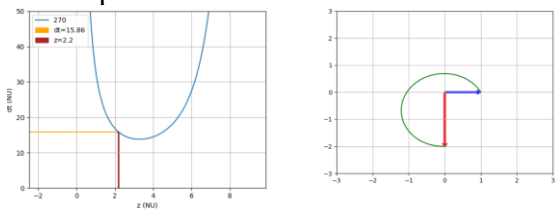


Figure 3



Short period CCW



Short period CW

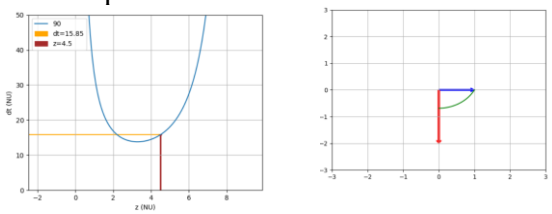


Figure 4

Here we can see that our program is unable to correctly compute the trajectory for Δθ=270°, short period, clockwise rotation. The other cases are well solved by our code.

Next, we can demonstrate the trajectory of a Hohmann transfer with our project. We simply do 2 changes of trajectory. We will transfer from a certain orbit to one 2 times bigger. We do the transfer with Δθ=180°, as it is a Hohmann transfer. With the two transfer we change from one orbit to one that is 2 times bigger.

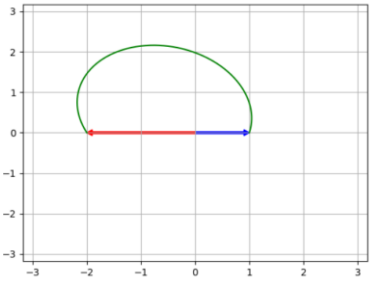


Figure 5 first transfer

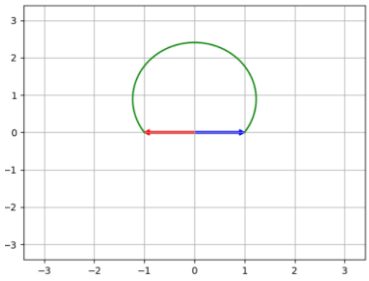


Figure 5 second transfer

The program can do 180° transfers, however it can't process 360° or 0° transfers, as those are conics degeneracies, in case of rectilinear orbit. The program finds a solution for the z value, but it cannot display the trajectory.

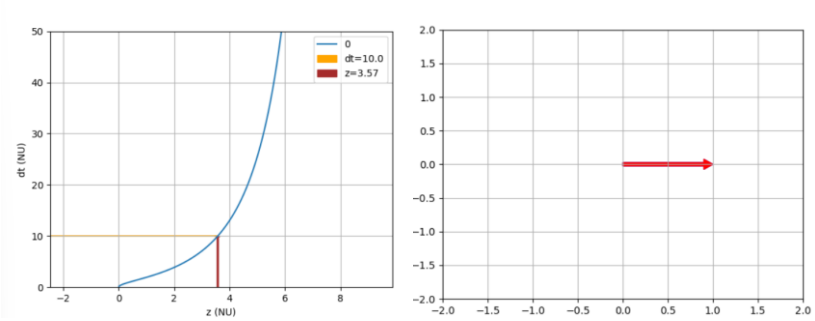


Figure 6 Plots for 0°

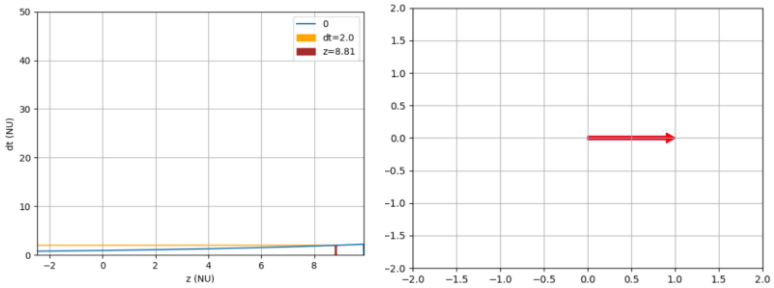


Figure 7 Plots for 360°

2. PCP and ΔV computations

We will now compute the PCP of ΔV for a transfer from Earth to every other planet of the solar system. We will use a range of transfer time Δt=[1,750] days and a range of days since 1st January 2020 equal to [1,1827]. Each of these range are composed of 185 equally spaced values, making it a total of 34,225 values of ΔV. By doing so we can find the minimum ΔV and launch opportunity for each transfer, between 2020 and 2025 excluded.

Here is an example of the PCP for Earth-Mercury transfer. Our program obviously gives us a wrong calculation of the total ΔV budget. Here the velocities of the planets have been reduced as they are approximately 11*10^9 km/s for Earth and 20*10^9 km/s for Mercury. As assignment 1 was another work and we did not receive any correction for it, we minimize its impact by reducing it to around 100 km/s. In comparison, the ΔV for our transfers is around 10 km/s. This is still what we will use for the rest of the report. Anyway, we can see that we have many transfer opportunities, with a minimum ΔV of 131 km/s, found at 368 days after 01/01/22 and with at Δt of 587 days.

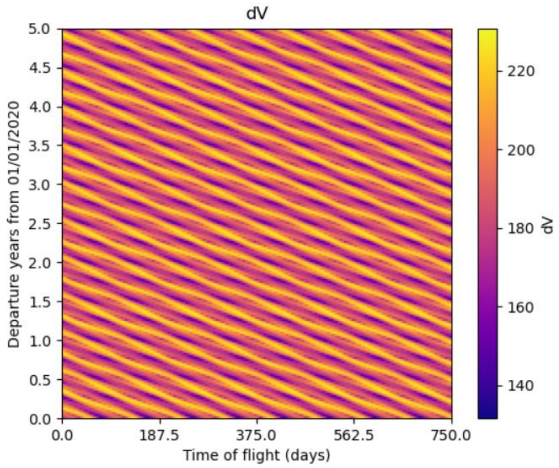


Figure 8 dV for Earth-Mercury transfer CCW

We can assess the influence of the rotation direction, clockwise (CW) or counterclockwise (CCW). Taking the same example of the Earth-Mercury transfer but this time in CW rotation, we find the minimum ΔV to be 130 km/s found at 190 days after 01/01/2022 and with a Δt of 327 days. The CW rotation offers similar ΔV budget but for a sooner date and a shorter transfer time of 260 days.

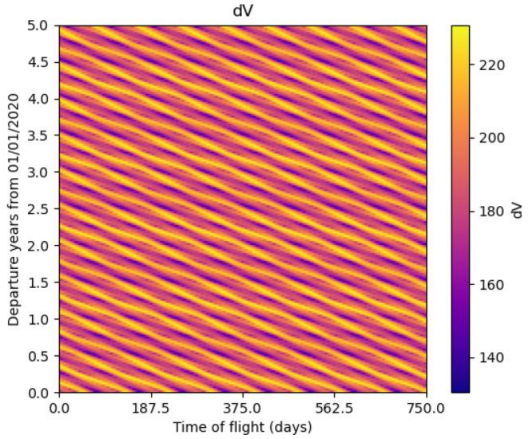


Figure 8 dV for Earth-Mercury transfer CW

In this next table we show what date and transfer time gives the minimum ΔV for each transfer in CCW direction from Earth to the other planets of the solar system.

Destination	Date of launch	Transfer time (days)	ΔV (km/s)
Mercury	03/01/2021	587	131
Venus	07/04/2023	734	119
Mars	07/05/2022	123	97
Jupiter	08/04/2024	746	85
Saturn	12/01/2020	1	80
Uranus	03/10/2024	587	80
Neptune	08/10/2021	497	78
Pluto	08/07/2020	139	84

We observe after calculations that our program takes on average 375 seconds (6 minutes 15 seconds) to find the PCP of each transfer. Those results are obtained considering using a portable computer plugged in for better performance, equipped with an Intel Core i7-8565U. This processor normally runs at 1.8GHz but it reaches 4.6GHz at turbo frequency.

Furthermore, we draw the table for the minimum ΔV budget and orbital elements of the transfers from Earth to all the other planets assuming the project deadline as launch date (11/01/2024 at 11h00 UTC+1). Only a, e and p are shown in this table, since the other elements (Ω, i, ω, θ) might not be correct as we will discuss in the conclusion.

Destination	Transfer time (days)	ΔV (km/s)	Semi-major axis a (AU)	Eccentricity e	Semi latus rectum p
Mercury	188	137	9.67	0.9896	0.2002
Venus	510	124	18.76	0.9941	0.2221
Mars	286	102	12.80	0.9859	0.3589
Jupiter	5	97	-3.30	1.0441	0.2973
Saturn	9	97	-1.33	1.7227	2.6214
Uranus	21	93	-1.63	1.5313	2.1964
Neptune	168	91	15.61	0.9370	1.9049
Pluto	58	86	-6.00	1.0217	0.2637

Conclusion

In conclusion, this project is extremely useful in real life cases and is a good example of what our future job will be about. Despite the mistakes made in this project, we learned a lot about the Lambert problem and more time and research could allow us to properly finish this work.

1. Learnings and difficulties

Throughout this project, we learned a lot about interplanetary transfers and the Lambert Problem. Many difficulties were encountered during the development of the Python program. First problem is the plotting of the trajectory between $\mathbf{r_1}$ and $\mathbf{r_2}$. For it to be correct, the equation of the trajectory should only be plotted between the angles θ1 and θ2. The angle of plotting is :

$$angle = [\theta 1, \theta 1 + \Delta\theta] = [\theta 1, \theta 2]$$

Consequently, the plotting equation must account for the new angles. This means that the equation to compute the trajectory changes :

$$trajectory_{x,y} = \begin{Bmatrix} \frac{p}{(1 + e * \cos(angle)} * \cos(angle - \theta 1) \\ \frac{p}{(1 + e * \cos(angle)} * \sin(angle - \theta 1) \end{Bmatrix}$$

Adding to that, an angle control must be applied to every angle calculated during our process. This applies to every θ, the eccentric anomaly E and the mean anomaly M. To do so we apply a modulo 2pi to our angles, this way they stay between the [0, 360] degrees range.

Leading into the 3d trajectories for planet transfers, a lot of difficulties were faced. Plotting the right direction for the vectors of the planet was a challenge because in our first test, we normalize $\mathbf{r_1}$ as [1,0,0]. For easing our work, we must normalize Earth point and move the other planet with respect to Earth's normalization. As we see in the next figures, we display the before and after normalization to have the correct trajectory.

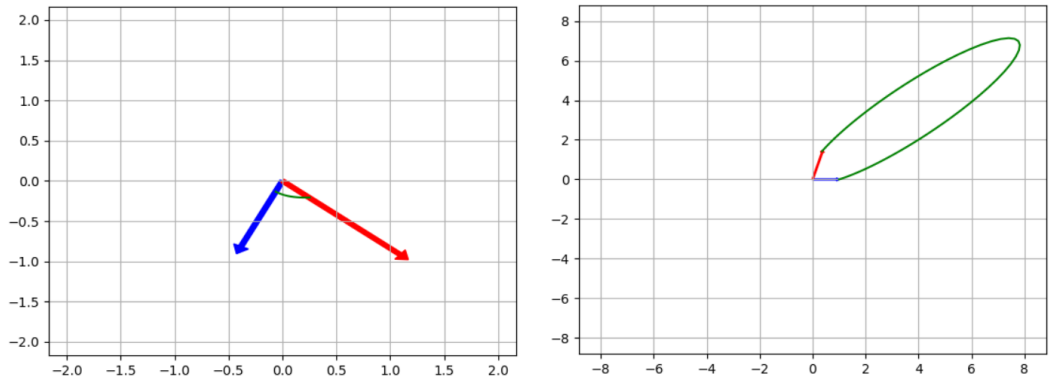


Figure 9 Earth-Mars trajectory before (left) and after (right) normalization.

After trying to display the 3d trajectory, we add inclination. We use the state vector to orbital elements algorithm to find i, Ω and ω and then we convert those orbital elements back to update the state vector with the z coordinates. However, the calculation of the orbital parameters is wrong, and we can't display the right transfer. In the next figure, we can see the 2d trajectory in green and the attempt at 3d in yellow. Also, we can see the effect of the normalization of our $\mathbf{r_1}$ and $\mathbf{r_2}$ vectors because they don't point to their respective orbits, which aren't normalized.

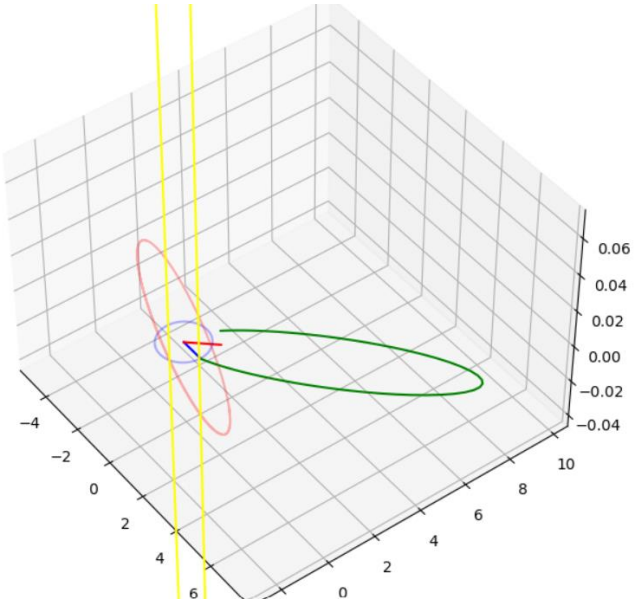


Figure 10 Attempt at modeling the 3d trajectory

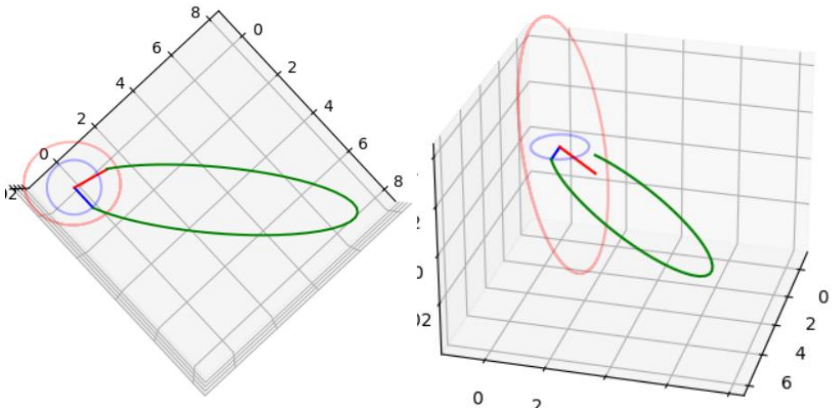


Figure 11 Different angles showcasing the precision problems of the 3d plotting.

Finally, the ΔV calculation should be fixed. The values coming from the PlanetPosition file are wrong as they represent enormous values. This part of the work was not looked upon aside from checking the dictionary values as it is an older work that did not get any correction.

2. Future improvements

In case of future advanced use of our program, it should be upgraded with improvements. Obvious ones are fixing the issues of 3d trajectory displaying. The calculation of the orbital elements should be checked as the issue might be coming from this or from the angle of display of the final 3d trajectory. Furthermore, the program should be able to work without having to normalize the vectors to simplify calculations.

Additionally, there are still elementary cases to fix in our 2d situation. The choice of CW and CCW has issues calculating the trajectory in specific angles and vectors size. Also, we could fix the case of 0° and 360° angle.