# SOLAR SALES ON YOUR TRIP TO MARS

Marco Praderio Bova, Eneko Martin Martinez,
& Maria dels Àngels Guinovart Llort
Team # 744
Problem A

**Abstract**

We study Logarithmically Spiral Trajectories and, in particular, we look for a solution to minimize the transit time of a Spacecraft propelled by a Solar Sail, while simultaneously minimizing the area of the Solar Sail, which would allow us to carry more payload on board. We start by analyzing the forces that act on the Spacecraft taking into account that its propellant is a Solar Sail; we use the studied forces to deduce the motion equations. We then solve this motion equation with a Runge-Kutta 4 method and transform the problem of minimizing time and area to a Non-linear Optimization problem. When solving the NLP we also try to minimize the relative final speed of th spacecraft with the destination planet in order to guarantee the possibility of a safe landing on its surface. The model improves when an angle parameter $\alpha$ (describing the angle formed by the Solar Sail with the colliding photons) is defined as a piecewise constant function and optimized whose values are optimized in every interval to minimize transit time and Area. Using the developed model to optimize the trajectory to be followed for sending from Earth to Mars a $2000\,kg$-spacecraft propelled by a Solar Sail, subject to the condition that at trajectory start Mars and Earth were at their closest approach, and the Arrival Relative Velocity is less than $9\,km/s$, give us a minimal transit time of $500\,days$ and a minimal area for the Solar Sail of $183158\,m^2$, meaning that the maximal payload would be $718\,kg$. Compared with different number of partitions of $\alpha$, the optimum stays stable. This gives a solid optimal trajectory and a great result for the numerical method used. Actually, waiting until the best moment to throw the Spacecraft, id est, Mars is at $1.14\,radians$ respectively to Earth initial position, the minimal sail area $145950\,m^2$ and, therefore, ables to transport until $978\,kg$ of payload with the same transit time. In addition and to conclude we tried the model to optimize the inverse trajectory.

# Contents

**1 Introduction** **3**

**2 Non Solar Sail models** **4**

2.1 Hohmann Transfer . . . . 4

2.2 Gravity Assist Encounters . . . . 5

**3 The model** **6**

3.1 Theoretical introduction . . . . 6

3.1.1 Logarithmic Spiral Trajectories . . . . 6

3.1.2 SRP Force . . . . 7

3.1.3 Sun Gravity Force . . . . 9

3.1.4 Differential Equation System . . . . 10

3.2 Numerical model . . . . 10

3.2.1 $\alpha$ constant . . . . 11

3.2.2 $\alpha = \alpha(t)$ . . . . 12

**4 Results** **13**

4.1 First results . . . . 13

4.2 Mass-Time optimization . . . . 16

**5 Further Analysis** **17**

**6 Strengths and Weaknesses** **19**

**7 Future Work** **20**

**8 Conclusions** **20**

**Appendices** **22**

A Deducing motion equations for Hohmann transfer . . . . 22

B Codes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

## 1 Introduction

Ever since the first success of Mariner 4, flying by Mars on November 1964, and the first success of Mariner 9, orbiting Mars on November 1971, many spatial missions have been conducted towards the red planet and its moons with mainly three different aims; Flying by the planet, entering its orbit and landing on the surface.

Most missions among the last decades were conducted using fuel propellant based spaceships but in the meanwhile an alternative was growing in strength. Solar sails had in fact been proposed for the first time in the 1920s, when Konstantin Tsiolkovsky and astronautical engineer Friedrich Zander had the original idea of taking advantage of the solar wind to navigate space. Later in the 1960s this idea was rediscovered and further investigated by Sands [1] and Fimple [2] who invented the concept of solar sail and theorized how to use them to abandon Earth. Nowadays these ideas have cemented into successful spacecrafts which have begun to pave the way for major investments into their research and use. The Japan Aerospace Exploration Agency (JAXA) launched its IKAROS spacecraft in 2010. NASA also deployed its NanoSail-D2 in early 2011 and The Planetary Society (TPS) successfully launched its first of a series of solar sail-propelled CubeSats called the LightSail [3]. The main difference between solar sails and the more famous chemical, ion, and nuclear rockets, is the fact that solar sails do not need to carry any fuel in order to produce thrust and accelerate [4]. Instead, they use large sails (from which they derive their name) made of very thin aluminized BoPET (Mylar) or aluminized Kapton film that reflect photons colliding on them. Thus doing, according to the momentum conservation law, they acquire the same momentum that the reflected photon has lost when being reflected. This transfer of momentum (see Figure 4) results in the acceleration of the sail away from the light source (usually the Sun) where the colliding photon was generated.

The growing importance solar sails have been achieving is motivated not only by the economical that can be derived from the usage of free fuel (the solar wind) but mostly from the capacity of continuing accelerating (even at a slow rate) for long periods of time making them ideal for long, relatively fast, trips.

The Goal of this paper is to design a model capable of, given certain information such as departure planet, arrival planet, maximum desired arrival speed and the angle formed by the vectors connecting the Sun to both planets, it could describe the optimal trajectory for a space sail in order to go from the first planet to the second minimizing the Sail area and travel time.

We will later use that model in a particular scenario in which we will find the characteristics of a solar sail based trip from Earth to Mars under the conditions of reaching the planet with a small relative speed (less than 9 Km/s) and that the planets are initially aligned with the Sun. We will also briefly compare this trip with a similar trip using a fuel based Rocket and the Hoffman maneuver. Finally, we will use our model to study solar sail based travel to larger distant regions of our solar system and trips in the opposite direction which go against the solar wind.

# 2 Non Solar Sail models

Before starting with the description of our model we will make a brief recap of the most common models used to plan space trajectories in order to achieve a global vision on the matter. The non interested reader skip this section and go straight to section 3 where we start with our real model. On the other hand for the interested reader we can provide a computer simulation for this models that we had developed before the contest beginning and that can be found in the following github link[1] that we have created for this sole purpose in order to maintain our identities anonymous as the contest rules require.

## 2.1 Hohmann Transfer

Hohmann Transfer method is not a viable method in this case due to its necessity to change fast the object velocity in an specific point of an orbit, but it has some interest in order to compare the travel time with the mentioned method.
This method it is an orbital changing method, very useful to place an object at an specific orbit, which could be an a proximate planet orbit as Mars is. Suppose that we have an object orbiting in an circular orbit, as the Earth orbit in Figure 1. The process to deduce the motion equations is similar than before and you can find explicitly in Appendix A and it gives us the following equations:

$$\frac{d^2r}{dt^2} - r\left[\frac{d\theta}{dt}\right]^2 = -\frac{\mu}{r^2} + \beta\frac{\mu}{r^2}cos^3\alpha \tag{1}$$

$$r\frac{d^2\theta}{dt^2} + 2r\left[\frac{dr}{dt}\right]\left[\frac{d\theta}{dt}\right] = \beta\frac{\mu}{r^2}cos^2\alpha sin\alpha \tag{2}$$

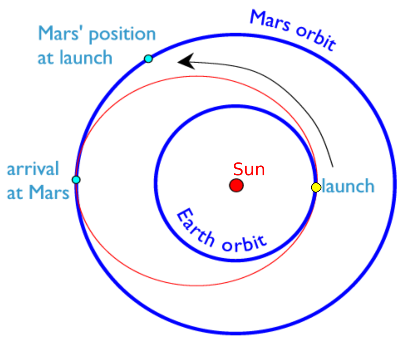

Figure 1: Hohmann Transfer trajectory.

We want to change objects orbit to one bigger, as Mars orbit in this case, in order to arrive to the planet. So when the object is passing by the yellow dot we will give him more tangential speed in order to change his orbit to an elliptical orbit, which will have the perigee at this point and the apogee coinciding

[1]https://github.com/team744UphysicsC/RocketTrajectory

with the finally orbit that we want. Once the object ends half the orbit, it will be at the apogee and we will be able to give him one more time the needed velocity in order to change his orbit to the circular Mars orbit, we will also be able to land the object in Mars at these punt.
Comparing this method with Logarithmic Spiral Trajectory for solar sail, about which we will talk about later, we see the differences shown in table 1 taken from [5].

Table 1: Hohmann Transfer & Logarithmic Spiral Trajectory for Earth to Mars

| | Hohmann Transfer | Solar Sail Logarithmic Spiral Trajectory | | | | | |
|---|---|---|---|---|---|---|---|
| $\beta$ | – | 0.05 | 0.10 | 0.125 | 0.15 | 0.20 | 0.30 |
| $a_0\,(mm/s)$ | – | 0.27 | 0.54 | 0.67 | 0.80 | 1.07 | 1.61 |
| $\alpha\,(deg)$ | – | 34.84 | 34.39 | 34.15 | 33.91 | 33.40 | 32.26 |
| $\gamma\,(deg)$ | – | 2.27 | 4.66 | 5.91 | 7.19 | 9.85 | 15.63 |
| $Time\,(days)$ | 259 | 875 | 431 | 342 | 283 | 209 | 135 |
| $\Delta v_1\,(km/s)$ | 2.94 | 1.24 | 2.52 | 3.18 | 3.85 | 5.22 | 8.12 |
| $\Delta v_2\,(km/s)$ | 2.65 | 1.00 | 2.04 | 2.57 | 3.12 | 4.23 | 6.58 |
| $\Delta v_T\,(km/s)$ | 5.59 | 2.24 | 4.56 | 5.75 | 6.96 | 9.45 | 14.69 |

We can check easily how Hohmann transfer is faster than Logarithmic needing less velocity difference in order to arrive faster.

## 2.2 Gravity Assist Encounters

A gravity assist, or hyperbolic, encounter is one where the ship will swing around a planet or satellite, using the gravitational pull of the body to alter the direction of motion of the ship. Mission planners use gravity assists because they allow the objective to be accomplished with much less fuel (and hence with a much smaller, cheaper rocket) than would otherwise be required; but in Solar Sails this fuel saving doesn't bring any new advantage, as our fuel resource is the radiation pressure exerted by sunlight [6].

A hyperbolic gravity assist of our Moon can be used to aid departures or arrivals. A ship spiraling out from Earth can direct its trajectory around the Moon on the trailing side, opposite from the Moon's direction of motion. This adds energy to the ship's trajectory which will both reduce the time to escape and give to the ship a greater speed relative to Earth when it does break free and enters interplanetary space. An arriving ships would swing around the leading side of the Moon.

The gains from using lunar gravity assists are modest, but may be sufficient to justify the added operating complexity. To use the gravity assists, the departures and arrivals must be timed such that the Moon will be in satisfactory position, placing a major constraint on the interplanetary trajectory. It is possible that the constraint might cause a time delay greater than the savings resulting from the gravity assist [7].

# 3 The model

## 3.1 Theoretical introduction

### 3.1.1 Logarithmic Spiral Trajectories

When the solar sail thrust is orientated at a fixed pitch angle to the Sun-line, with a low sail performance, we obtain a logarithmic spiral trajectory [8] [5]. The radial component of the sail thrust reduces the effective gravitational force on the sail as for zero pitch, however the component of thrust in the transverse direction acts to increase (or decrease) the orbital angular momentum of the orbit.
It should be reiterated here that the solar sail can spiral away from the Sun by directing a component of thrust along the velocity vector in order to increase the orbital angular momentum and so spiral away from the Sun. Similarly, negative pitch angle can direct a component of the thrust against the velocity vector, so decreasing the orbital angular momentum as so spiral towards the Sun.
For fixed sail pitch angle in a two-dimensional planar case, a particular solution to the polar equations of motion in Eqs. 32 and 33, from appendix, can be found to Eq. 3.

$$r(\theta) = r_0\, exp(\theta\, tan\gamma) \tag{3}$$

Where $r_0$ is the initial distance from the Sun, and $\gamma$ is the spiral angle, the angle between the transverse direction and the velocity. For a specific logarithmic spiral, the spiral angle will be fixed, as shown in Figure 2. If we differentiate the solution and substitute, we can obtain the velocity magnitude in Eq. 4, where it can be seen that the local solar sail speed is always less than the local circular orbit speed.

$$v(r) = \sqrt{\frac{\mu}{r}}\, \left[1 - \beta\, cos^2\alpha\, (cos\alpha - sin\alpha\; tan\gamma)\right]^{1/2} \tag{4}$$

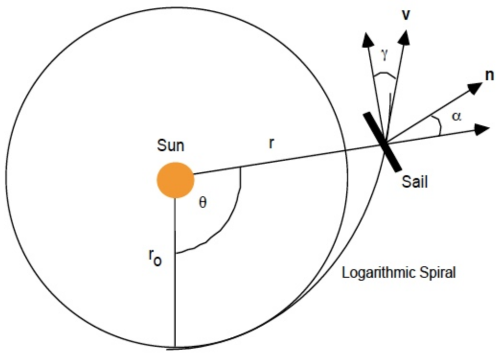

Figure 2: Logarithmic spiral trajectory.

This means that coplanar transfer between two circular orbits cannot be achieved without applying a hyperbolic excess at launch to place the solar sail onto the logarithmic spiral, and circularizing the orbit on arrival at the final circular orbit. These discontinuities pose problems in the practical application of logarithmic spirals to orbit transfers. However, the logarithmic spiral often provides a good first guess for a more advanced control law that can deal with this two-point boundary value problem.

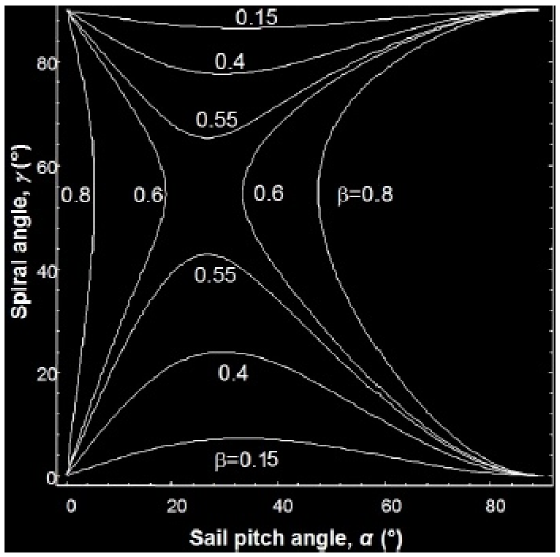

Figure 3: Implicit relationship between sail pitch angle and logarithmic spiral angle, for contours of sail lightness number.

By combining the radial and transverse components of velocity, and substituting these into the equations of motion, it can be shown that an implicit relationship exists between the sail pitch angle and spiral angle, as in Eq. 5, the result of this is depicted in Figure 3. For the lower branch of solutions, it is observed that there appears to be an optimal sail pitch angle that maximizes the spiral angle for each sail lightness number. This leads into the definition of the optimal pitch angle derived from optimal control theory [8] [5].

$$\frac{sin\gamma\, cos\gamma}{2 - sin^2\gamma} = \frac{\beta\, cos^2\alpha\, sin\alpha}{1 - \beta\, cos^3\alpha} \tag{5}$$

### 3.1.2 SRP Force

Sunlight force acting at the solar sail is a result of the Sun Radiation Pressure (SRP), given by:

$$P = P_0 \left(\frac{r_0}{r}\right)^2 \tag{6}$$

where $P_0 = 4.563\,\mu N/m^2$ is the solar radiation pressure at $r_0 = 1\,Au$, and $r$ is the distance from the Sun. If we have an ideal model, the force given by SRP is normal to the surface of the sail. Let be $\alpha$ angle of the solar sail with the photons beam. The SRP impacting with the sun reflection depend on the orientation of the sail, so if $A$ is the area of the sail, we should consider the effective area as $A\,\cos(\alpha)$. If we use a Cartesian coordinate system which has the x axis tangential to the sail surface and the y axis parallel to the surface normal, a photon particle linear momentum will be given by $\overrightarrow{p}_0 = p\left(\sin(\alpha)\overrightarrow{i} + \cos(\alpha)\overrightarrow{j}\right)$, after reflected, and his momentum will be given by $\overrightarrow{p}_f = p\left(\sin(\alpha)\overrightarrow{i} - \cos(\alpha)\overrightarrow{j}\right)$, so we conclude that the total momentum change of the photon will be:

$$\Delta\overrightarrow{p} = -2\,p\,\cos(\alpha)\,\overrightarrow{j} \tag{7}$$

Using the linear momentum conservation law, we have that the momentum wined by the sail will be $-\Delta\overrightarrow{p}$. Using the SRP expression and taking into account the effective area, we conclude that the force acting at the sail will be:

$$\overrightarrow{F}_0 = 2\,P_0\left(\frac{r_0}{r}\right)^2 A\,\cos^2(\alpha)\,\overrightarrow{j} = 2\,P_0\left(\frac{r_0}{r}\right)^2 A\,\cos^2(\alpha)\,\overrightarrow{n} \tag{8}$$

Were $\overrightarrow{n}$ is the unitary normal vector to the surface. We can see this in the following figure:

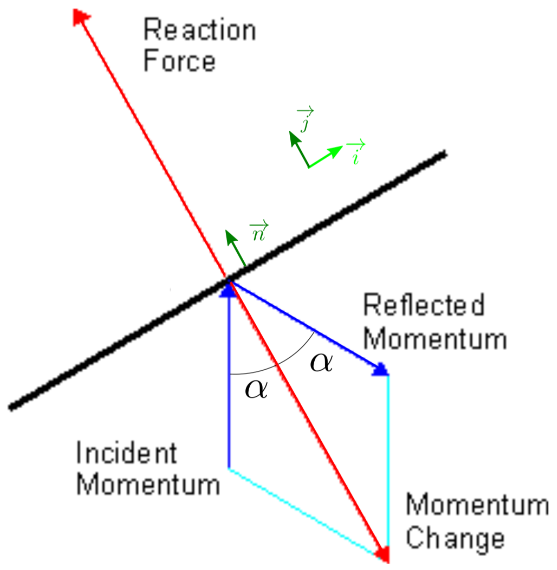

Figure 4: Incident photons momentum change.

However, we shall consider also the absorbed photons by the sail, which give their lineal moment to totally to the sail, so if an absorbed photon haves a moment $\overrightarrow{p} = p\,\hat{r}$ where $\hat{r}$ is the unitary radial vector considering our coordinates system centered at the Sun. So, let be $\rho$ be the reflection coefficient and $s$ the specular reflection factor, we should first correct the force given by reflected photons,

$$\overrightarrow{F}_1 = \rho\,s\,\overrightarrow{F}_0 \tag{9}$$

Then, we calculate the force given by absorbed photons:

$$\overrightarrow{F}_2 = (1-\rho\,s)\;P_0\left(\frac{r_0}{r}\right)^2 A\,\cos(\alpha)\,\hat{r} \tag{10}$$

We should also consider the diffusely reflected photons which, will be proportional to the reflection coefficient times the non-specular reflected photons, $1-s$, times non-Lambertian coefficient of the front side, $B_f$, which gives us the angular distribution of the diffusely reflected photons. So we have

$$\overrightarrow{F}_3 = \rho\;(1-s)\;B_f\,P_0\left(\frac{r_0}{r}\right)^2 A\,\cos(\alpha)\,\overrightarrow{n} \tag{11}$$

We finally considered the linear momentum added by emitted photons, we know that emitted photons quantity will be proportional to the absorbed ones, but we may consider also the proportion of the ones emitted by the front side and the ones emitted by the back side, the first ones will accelerate our system and the seconds will brake it. Let be $\epsilon_f$ and $\epsilon_b$ the front and back emitted photons coefficients, the ratio of the photons emitted in the front will be

$$r_f = \frac{\epsilon_f}{\epsilon_f + \epsilon_b} \tag{12}$$

And the ratio of the emitted ones at the back side

$$r_b = \frac{\epsilon_b}{\epsilon_f + \epsilon_b} \tag{13}$$

Taking $B_b$ as the back non-Lambertian coefficient, we can express the force given by photons emissions as

$$\overrightarrow{F}_4 = (1-\rho)\ (r_f\, B_f - r_b\, B_b)\ P_0 \left(\frac{r_0}{r}\right)^2 A \cos(\alpha)\, \overrightarrow{n} \tag{14}$$

We finally can express the total SRP originated force given by

$$\overrightarrow{F}_{SRP} = \overrightarrow{F}_1 + \overrightarrow{F}_2 + \overrightarrow{F}_3 + \overrightarrow{F}_4 = 2\, P\, A\, \cos(\alpha)\, (b_1\, \hat{r} + (b_2\, \cos(\alpha) + b_3)\, \overrightarrow{n}) \tag{15}$$

were $b_1$, $b_2$ and $b_3$ are characteristic constants of our sail

$$\begin{aligned} b_1 &= \frac{1-\rho\, s}{2} \\ b_2 &= \rho\, s \\ b_3 &= \frac{1}{2}\left(B_f\, \rho\, (1-s) + (1-\rho)\ \frac{\epsilon_f\, B_f - \epsilon_b\, B_b}{\epsilon_f + \epsilon_b}\right) \end{aligned} \tag{16}$$

In order to facilitate our work we consider all in polar coordinates, changing $\overrightarrow{n} = \cos(\alpha)\hat{r} + \sin(\alpha)\hat{\theta}$, and considering the force per mass unit, taking $a_c = \frac{2\, P_0\, A}{m}$, where $a_c$ represents the acceleration experienced by an ideal solar sail oriented perpendicular to the Sun, $\overrightarrow{n} = \hat{r}$, at $r = r_0$. So we have

$$\overrightarrow{f}_{SRP} = a_c \left(\frac{r_0}{r}\right)^2 \cos(\alpha) \left(\left(b_1 + b_2\, \cos^2(\alpha) + b_3\, \cos(\alpha)\right)\, \hat{r} + (b_2\, \cos(\alpha) + b_3)\, \sin(\alpha)\, \hat{\theta}\right) \tag{17}$$

### 3.1.3 Sun Gravity Force

As we know the gravitational force per mass unit is given by:

$$f_G = -\frac{G\, M}{r^2}\hat{r} \tag{18}$$

where $G$ is the gravitation universal constant and $M$ the mass of the Sun. We can know consider a for variable vector

$$\overrightarrow{x} = \begin{bmatrix} r \\ \theta \\ u \\ v \end{bmatrix} \tag{19}$$

where $u$ is the radial velocity and $v$ the tangential velocity. So if we only consider the Sun's gravity effects we can use the following differential system:

$$\dot{\overrightarrow{x}} = \begin{bmatrix} \dot{r} \\ \dot{\theta} \\ \dot{u} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} u \\ \frac{v}{r} \\ \left(\frac{v^2}{r} - \frac{G\, M}{r^2}\right) \\ -\frac{u\, v}{r} \end{bmatrix} \tag{20}$$

### 3.1.4 Differential Equation System

We can add to the last system the force given by the SRP, and we will finally have the differential equation system needed in order to calculate the trajectory. So we have

$$\dot{\vec{x}} = \begin{bmatrix} u \\ \frac{v}{r} \\ a_c \left(\frac{r_0}{r}\right)^2 \cos(\alpha) \left(b_1 + b_2 \cos^2(\alpha) + b_3 \cos(\alpha)\right) + \frac{v^2}{r} - \frac{G\,M}{r^2} \\ a_c \left(\frac{r_0}{r}\right)^2 \cos(\alpha) \sin(\alpha) \left(b_2 \cos(\alpha) + b_3\right) - \frac{u\,v}{r} \end{bmatrix} \tag{21}$$

## 3.2 Numerical model

Once even though equation 21 describes uniquely the motion of a Solar Sail under the effects of the Sun (solar gravity field and solar wind) it has an inconvenience, it is not analytically solvable. Therefore a numerical model is needed if we want to proceed. Before going on with the description of this numerical model we will have to stop a moment to enumerate a set of assumptions.

- **The orbits of all planets are exactly circular:** This is a standard assumption that supposes little decrease in precision given that the orbits of the solar system planets are all approximately circular.
- **The gravitational fields produced by celestial bodies other than the Sun are negligible:** This assumption is correct almost during almost all the trip except during the processes of escaping the starting planet's orbit and when entering the destination planet orbit. However, since this processes are realized with auxiliary propulsion methods other than solar wind propulsion, these will not be considered in this paper.
- **The Sun and the planets orbiting can be projected in a 2 dimensional field without loss of generality:** This assumption is quite reasonable since all the planets in the solar system have almost coplanar orbits.
- **The Sun and planets are modeled as spherical objects:** This assumption will help us define a condition of reaching a planet as well as applying the known formula of the gravitational force exerted by a spherical object.
- **Atmospheric drag and shadowing effects of Earth and other planets are neglected:** This assumption is based on the fact that most of the trip is done while away from any planet and under this condition shadowing effects are minimal and would unnecessarily difficult the model.
- **We consider that a Solar Sail has reached a destination planet when the distance between the solar sail and the planet is lower than the planet's radius:** It is necessary to make this assumption since it is virtually impossible to find a trajectory that goes through the exact center of the destination planet and thus we cannot consider it as a dimensionless point.
- **Equation 21 is valid and the values of $b_1$, $b_2$ and $b_3$ are $0.0864$, $0.8277$ and $-5.43 \cdot 10^{-3}$, respectively:** The validity of equation 21 was already deduced in the previous section as for the values of the dimensionless constants where extracted from [9].
- **We can give to our solar sail an initial velocity relative to Earth at most equal to Earth's escape velocity:** This is a necessary condition that must be satisfied to abandon the planet.

- **We can control the angle that the initial speed relative to Earth forms with the line tangent to Earth's orbit:** This assumption is perfectly reasonable since it would only be necessary to start the process of leaving the planet at a different point of it in order to change this angle.

Once clarified these points, we can proceed with the model's explanation. Let's first study the simplest case where the $\alpha$ value that appears in equation 21 is constant during all the trajectory.

### 3.2.1 $\alpha$ constant

In this scenario we can use a simple Runge-Kutta 4 algorithm to numerically solve the motion equation and, given initial conditions, find the approximate position and velocity of the Solar Sail at a time $t_f$. Using this result and denoting by $r\,(t_f)$ and $\theta\,(t_f)$ the polar coordinates at time $t_f$ of the destination planet and by $v\,(t_f)$ the tangential velocity of the destination planet at that time $t_f$[2]. We can thus compute the final distance from the Solar Sail and the destination planet and their relative velocities at any time $t_f$ as a function of $t_f$, the parameters appearing in the motion equation 21 and the initial conditions. In other words denoting by $D_p$ the distance between the Solar Sail and the destination planet and by $D_v$ the module of their relative velocity, we can write

$$D_p = D_p\,(\alpha,\, t_f,\, a_c,\, v_0) \qquad\qquad D_v = D\,(\alpha,\, t_f,\, a_c,\, v_0) \tag{22}$$

Where $v_0$ is the initial speed of the Solar Sail. Since $a_c$ is defined as a function of solely the surface area of the sail $(A)$, we can rewrite

$$D_p = D_p\,(\alpha,\, t_f,\, A,\, v_0) \qquad\qquad D_v = D\,(\alpha,\, t_f,\, A,\, v_0) \tag{23}$$

If we now denote by $R$ the radius of the destination planet and by $V$ the maximal relative speed with which we want to reach the destination planet, we can define the function

$$f\,(\alpha,\, t_f,\, A,\, v_0) = \max\,(D_p - R,\, 0) + \max\,(D_v - V,\, 0) \tag{24}$$

It is clear that this function is always positive and only equals 0 for those values of $\alpha$, $t_f$, $A$ and $v_0$ for which the conditions of reaching the destination planet at the desired speed are met. Thus the solutions of the non linear optimization problem $(NLP)$ consisting on minimizing $f$ are exactly the same solutions of the our initial (if this last solutions exist) problem which allows us to view the problem of reaching the destination planet at a given speed as something that we can numerically solve quite simply using the Successive Quadratic Programming algorithm $(SQL)$ implemented in the Scipy Python Library. The function provided by this library has the additional feature, which we have used in our program, that we can impose condition on the parameters we use to optimize. This allows us to make sure that the optimal Sail area find by the simulation will never surpass the threshold given by the weight limit or that the time trajectory time will never surpass a threshold set to 1.8 years. We still remain with the problem of finding the best optimal solution of all possible solutions, id est the solution that minimizes $t_f$ and $A$. This can be achieved quite simply by changing the objective function $f$ of the previous $NLP$ for the function

$$g\,(\alpha,\, t_f,\, A,\, v_0) = k_1\, \max\,(D_p - R,\, 0) + k_2\, \max\,(D_v - V,\, 0) + k_3\, t_f + k_4\, A \tag{25}$$

Where $k_i$ are positive constants that are used to give preference to the minimization of certain parameters over others. The introduction of these parameters is necessary since this time we do not have a one

[2]Let us note that according to our assumption of circularity of the planets orbits this velocity is constant in time and equal to $\sqrt{\frac{G\,M}{R}}$ where $G$ is the gravitational constant, $M$ the Sun's mass and $R$ the orbital radius of the destination planet. On the other hand the radial velocity of any planet at any time is 0 and therefore we can deduce very easily the position and velocity of the destination planet at any given time.

to one correspondence between the solutions of the $NLP$ and the solutions of our original problem. In fact, it could be possible for the minimal value of $g\left(\alpha,\, t_f,\, A,\, v_0\right)$ to be achieved for values such that $\max\left(D_p - R,\, 0\right)$, $\max\left(D_v - V,\, 0\right) \neq 0$, but $t_f$ and $A$ are so small that they compensate for it. Taking values $k_1,\, k_2 >> k_3,\, k_4$ we can considerably decrease the possibility of this to happen.

### 3.2.2 $\alpha = \alpha\left(t\right)$

Now that we have a model for the simplest case of $\alpha$ constant we would like to find a solution for any function $\alpha = \alpha\left(t\right)$. In order to do so, we will approximate a general $\alpha\left(t\right)$ by a piecewise function $\alpha_n\left(t\right)$ defined as

$$\alpha_n\left(t\right) = \sum_{k=1}^{n} \alpha_n^k\, \chi_{\left[\frac{k-1}{n}\, t_f;\, \frac{k}{n}\, t_f\right)}\left(t\right) \tag{26}$$

where $\alpha_n^k \in \mathbb{R}$, $t_f$ has the same value as before and $\chi_{\left[\frac{k-1}{n}\, t_f;\, \frac{k}{n}\, t_f\right)}\left(t\right)$ is defined as

$$\chi_{\left[\frac{k-1}{n}\, t_f;\, \frac{k}{n}\, t_f\right)}\left(t\right) = \begin{cases} 1 & \text{if } t \in \left[\frac{k-1}{n}\, t_f;\, \frac{k}{n}\, t_f\right) \\ 0 & \text{if } t \notin \left[\frac{k-1}{n}\, t_f;\, \frac{k}{n}\, t_f\right) \end{cases} \tag{27}$$

Once done this, it is theoretically possible to proceed as before in order to reduce the problem to an $NLP$ but the dependence of the definition of the function $\alpha_n$ on the parameter $t_f$ makes it harder to program the method to solve the differential equation 21 using the Runge-Kutta-4 method and Python's default libraries in order to solve the problem[3]. We can make up for this problem introducing the variable change $s = \frac{t}{t_f}$ this way equation 21can be rewritten as

$$\dot{\overrightarrow{x}} = t_f \begin{bmatrix} u \\ \frac{v}{r} \\ a_c\, \left(\frac{r_0}{r}\right)^2\, \cos(\alpha_n(s))\, \left(b_1 + b_2\, \cos^2(\alpha_n(s)) + b_3\, \cos(\alpha_n(s))\right) + \frac{v^2}{r} - \frac{G\,M}{r^2} \\ a_c\, \left(\frac{r_0}{r}\right)^2\, \cos(\alpha_n(s))\, \sin(\alpha(s))\, \left(b_2\, \cos(\alpha_n(s)) + b_3\right) - \frac{u\,v}{r} \end{bmatrix} \tag{28}$$

Where now $\alpha_n(s)$ does not depend on $t_f$ and is defined by

$$\alpha_n\left(s\right) = \sum_{k=1}^{n} \alpha_n^k\, \chi_{\left[\frac{k-1}{n};\, \frac{k}{n}\right)}\left(s\right) \tag{29}$$

And we can apply the same procedure followed in the case of $\alpha$ constant to reduce our problem to an $NLP$ problem which can be solved using Scipy libraries.

[3]This approach can also be used in the case of $\alpha$ constant and would result in an improvement of the result since the variable integration interval $[0,\, t_f]$ would become the fixed interval $[0,\, 1]$ and this would make it easier to choose an integration step.

# 4 Results

## 4.1 First results

We have tested our model to find the initial conditions, $\alpha(t)$ function, sail size and trajectory time that would minimize sail size and trajectory time while succeeding at bringing a Solar Sail from Earth to Mars reaching Mars with a relative speed lower than 9 Km/s and considering that at initial time Sun, Earth and Mars are aligned. The values of the four $k_i$ constants that appear in equation 25 and that we used to perform this test are

$$k_1 = 1E-4 \qquad k_2 = 5E-3 \qquad k_3 = 0.005555555 \qquad k_4 = 5 \tag{30}$$

Note that the values of $k_1$ and $k_2$ are much smaller than $k_3$ and $k_4$ which, at first sight may seem to contradict our explanation of the model where we said that $k_{1,2} >> k_{3,4}$. This is due to the fact that the variables $D_b$ and $D_v$ are of many orders of magnitude greater than $t_f$ and $A$. Therefore, even if the values of $k_1$ and $k_2$ are, in fact, smaller than $k_3$ and $k_4$, this difference is only apparent, since if we were to scale $D_b$, $D_v$, $t_f$ and $A$ to make them have values of the same magnitude order under normal conditions (for example writing $A$ in $cm^2$ units instead of $km^2$ units and $t_f$ in minutes instead of hours) we would find new $k_i$ values which would follow $k_{1,2} >> k_{3,4}$. With this parameters we have run several simulations where we defined $\alpha$ as a function defined in 10, 15, 20, 25 and 30 different sections. The results obtained can be seen in table 2 and Figures 5 and 6.

Table 2: Results of the simulation for different values of the partitioning of $\alpha$. It is important to remember when reading this table that Mars' radius measures $3396\,Km$. Table with values for different number of partitions and its correspondence with ARV (Arrival Relative Velocity), D (final distance from Mars center) and V (initial speed relative to the Earth)

| #Partitions | time $(days)$ | ARV $(km/s)$ | A $(Km^2)$ | D $(Km)$ | V $(Km/s)$ |
|---|---|---|---|---|---|
| 10 | 518.95 | 7.14 | 0.21 | 3157.40 | 0.08 |
| 15 | 499.79 | 8.03 | 0.19 | 3710.8 | 0.11 |
| 20 | 500.57 | 8.24 | 0.18 | 3524.3 | 0.11 |
| 25 | 500.68 | 8.64 | 0.19 | 1165.27 | 0.12 |
| 30 | 497.93 | 7.32 | 0.20 | 361.18 | 0.17 |

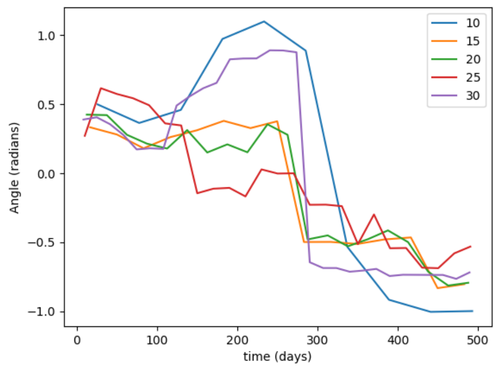

Figure 5: Optimal function $\alpha(t)$ with $\alpha$ variating the number of partitions.

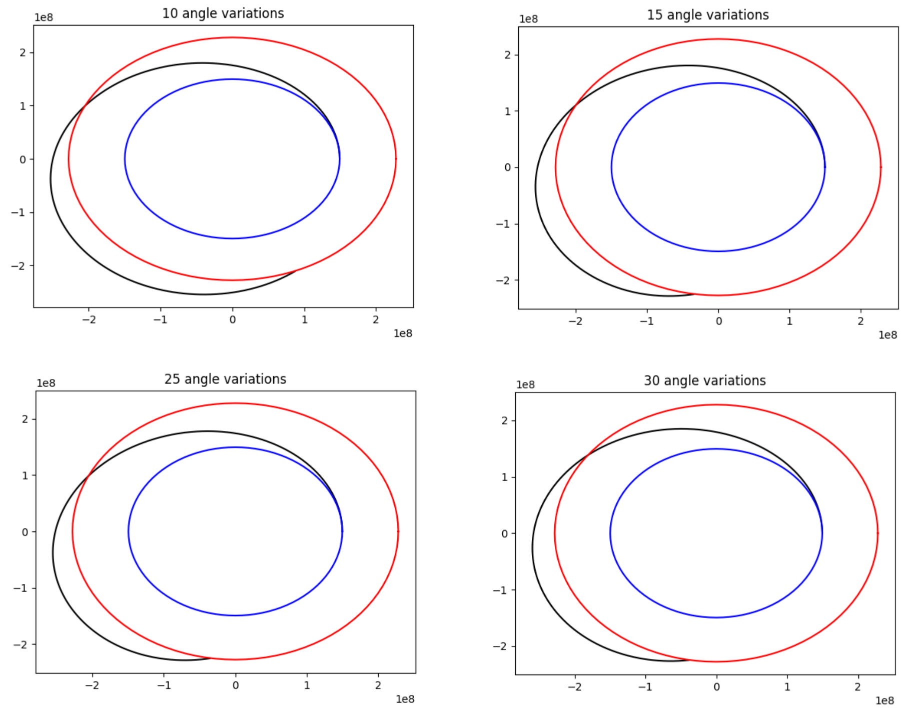

Figure 6: Optimal trajectories found when variating the number of partitions of $\alpha$.

As we can observe from table 2, using these parameters of $k_i$, all our simulations succeeded at reaching Mars with the desired velocity and we obtain a maximal efficiency between 15 and 20 partitions of the function $\alpha$. At first, this result can seem contradictory since a greater number of partitions of $\alpha$ should result in a better approximation of the ideal $\alpha$ function. However, we must keep in mind that, when we increase the number of parameters we use to solve the NLP (which happens when we increase the number of partitions of $\alpha$), we also increase the numerical error in the solving process, which can result in a worsening of the final result. It is also important to notice that, in the last column of the table, where the initial speed with respect to Earth is reported, the fact that this value is different from 0 in every line and it keeps increasing together with the number of partitions. This means that probably the best possible traveling method is neither purely based on Solar wind propulsion nor in fuel propulsion method with only one burst. On the other hand, the fact that this initial value is clearly inferior to the one used for the Hohmann maneuver ($2.49\,Km/s$ as reported in table 1) and, even so, the trajectory described by the Solar Sail easily surpasses Mars orbit, what indicates us the power of this traveling system which is able to reach further destinations with an input energy 10 times lower. Another interesting fact that we can intermediately see by observing Figures 5 and 6 has little effect on the variation of the trajectory which indicates that the obtained solution has a certain degree of stability. In order to further study this degree of stability we have run the simulation multiple times changing a set of initial parameters obtaining a similar result for the optimal trajectory found (we decided not to report these results as images since it would be like copying the already shown ones). This evidences suggest us that the optimal solution found is a very good minimum if not an absolute minimum. Before we conclude, two more things are worth noticing when observing figure 5. The first one is that the solutions obtained with 10 and 30 partitions are very similar, in the sense that both show a sudden increase for the central period of time when the Solar Sail is the furthest away from the Sun. This is probably related with the Sail size, which is greater in these two cases and has probably resulted in the Solar Sail achieving a greater velocity, escaping from the Sun at this point of the trajectory. Therefore, velocity must be compensated with a sharper change of the value $\alpha$, in order to accelerate in the opposite direction, to decrease their speed. Finally, it is worth noticing how the $\alpha$ functions all have the same decreasing behavior, which is translated in an initial increase in velocity which sends the solar ship away from the Sun followed by a decrease of this that helps keeping the Solar Sail closer to the Sun. This behavior can be better comprehended by observing the animation produced by our code, reported in Annex B. To conclude this subsection is worth studying what happens in the case we only realize one, two or three partitions of the function $\alpha$. Under this conditions we obtain once again a trajectory identical to those shown in figure 6 and that, therefore, are not worth showing. Other results can be observed in table 3 and 7.

Table 3: Results of the simulation for different values of the partitioning of $\alpha$. It is important to remember when reading this table that Mars' radius measures 3396 Km. Table with values for different number of partitions and its correspondence with ARV (Arrival Relative Velocity), D (final distance from Mars center) and V (initial speed relative to the Earth)

| #Partitions | time $(days)$ | ARV $(km/s)$ | A $(Km^2)$ | D $(Km)$ | V $(Km/s)$ |
|---|---|---|---|---|---|
| 1 | 502.33 | 4.43 | 0.23 | 2414.46 | 0.11 |
| 2 | 500.04 | 7.75 | 0.19 | 3395.82 | 0.11 |
| 3 | 500.15 | 7.97 | 0.19 | 3395.83 | 0.13 |

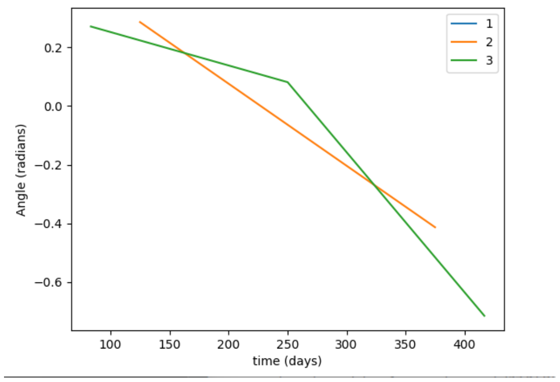

Figure 7: Optimal function $\alpha(t)$ with $\alpha$ variating the number of partitions.

As we can observe from the results shown in the table 3 there is almost no difference with those observed in 2 (except for a greater value on the Sail area when we have just one partition). This fact supposes another evidence on the fact that our solution and, more importantly shows us that it is not necessary to perform many steering maneuvers in order to achieve a good approximation of the optimal trajectory. This is a very important fact since steering maneuvers can result complicate and it may be preferable to keep them to a minimum.

In figure 7 we can once again observe the decreasing tendency of the $\alpha$ function which means that in order to achieve optimal results we must accelerate at the beginning and decrease speed once reached or surpassed the desired orbit.

## 4.2 Mass-Time optimization

Using non-linear optimization at Python programming, we have been able to change the weight for the area/time optimization, so we obtain our results by giving more importance to one of them against the other: We see that the result maintains the Arrival Relative Velocity (ARV) under $9\,km/s$.

Table 4: Area-Time-Mass optimization changes

| Area ($m^2$) | time ($days$) | ARV ($km/s$) | Maximal Payload ($kg$) |
|---|---|---|---|
| 182215 | 508 | 8.49 | 724.5 |
| 183031 | 501 | 8.88 | 718.5 |
| 183158 | 500 | 8.31 | 718.0 |
| 186755 | 500 | 8.13 | 692.7 |
| 189426 | 500 | 8.07 | 674.0 |
| 190419 | 500 | 8.96 | 667.1 |
| 199866 | 500 | 7.82 | 601.0 |
| 200746 | 505 | 7.56 | 594.8 |
| 200958 | 501 | 7.73 | 593.3 |
| 209168 | 501 | 7.51 | 535.8 |

As we see, the dependence on changing weights is not very relevant, so we have the minimal time at $500\,days$. However, having a bigger area does not mean arriving faster, because if the area is too big, we may change the trajectory for a longer one in order to arrive under the required ARV. We can see this effect when the area becomes bigger than $200000\,m^2$. We take as optimal value the one with $183158\,m^2$ and $500\,days$; then with a solar sail mass of $7\,g/m^2$, it gives us a maximal payload charge of $718\,kg$.
As we work with an $1.8\,years$ boundary conditions we may can reduce the needed area by increasing this boundary time. It will depends on which is more relevant to us, increasing the payload quantity or reducing the travel time.

## 5 Further Analysis

Now that we have tested our model to learn the optimal trajectory to go from one planet to another one given the initial conditions, we can try to slightly modify the model to optimize the orbit variation. We study to optimize the solution, without considering the initial condition that Mars and Earth are aligned with the Sun. In order to adapt the model to solving this problem, we just have to redefine the parameters $D_p$ and $D_v$ shown in equation 25 as $D_p$ the distance between the orbit radium and the Solar Sail distance from the Sun and $D_v$ the difference between Solar Sail velocity and orbital velocity[4]. Thus doing we obtained that the optimal sail size is 0.15 $Km^2$, which is much better than the ones obtained before, while the transit time remains more or less the same, $499.99\,days$. Even though, according to our model, the Solar Sail spends 500 days to reach Martian orbit and Martian orbital speed. But according to figure 8 (left), the Solar Sail reaches the orbit much before, but it still tries a little while to reach orbital velocity and in the meanwhile it maintains the orbit thanks to the solar wind pressure. So we are able to calculate the Mars-Earth angle, centered at the Sun, when launching in order to arrive to Mars. Mars initial position should be at $1.14\,rad$ ($65.38\ ^o$) respectively to Earth initial position see figure 8 (right) .

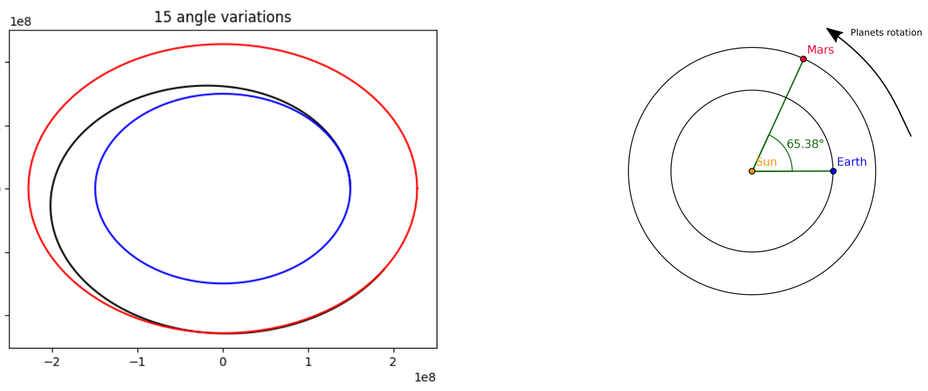

Figure 8: Trajectory followed by a Spacecraft propelled with a Solar Sail to change orbit from Earth to Mars (left). Initial relative positions of the planets in order to arrive to Mars (right).

To conclude this section we will test the model for other two trajectories. The first to study is the trajectory from the Martian orbit to Earth's Orbit and the second one, the trajectory from Earth's orbit to Jupiter's Orbit. Unfortunately, due to errors of overflow, our python script was not able to give a proper result for the Earth-Jupiter trajectory and crushed prematurely. In order to solve these problems it would be necessary to either hope for them to be solved simply changing the dimensions of the values (instead of hours, use days and instead of kilometers, use astronomical units) or change the code to a lower level language as C++ that would allow us to define personalized variables capable of handling the needed calculations without throwing an overflow error. On the other hand, our script was easily able to reach a solution for bringing the Solar Sail from Martian Orbit to Earth orbit. According to the obtained result, the optimal Sail size for this trajectory would be $0.24\,km^2$, while the optimal time remains at 499.98days. It is worth mentioning that, according to this model, the optimal initial velocity for the Spacecraft would be in the opposite direction of Mars' velocity. The graphical results obtained in this model are shown in images 9 and 10. Notice how, contrarily on how happened in the other trajectories the angle value grows

[4]Orbital velocity for an orbit of radium $R$ is defined as the purely tangential velocity of modulus $\sqrt{G\,M/R}$ where $G$ is the mass of the Sun and $M$ Sun's mass.

in time instead of decreasing.

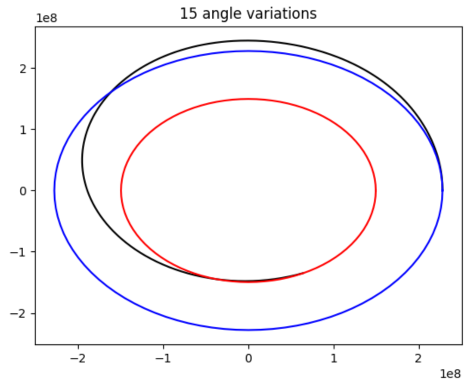

Figure 9: Trajectory followed by a Spacecraft propelled with a Solar Sail to change orbit from Mars to Earth.

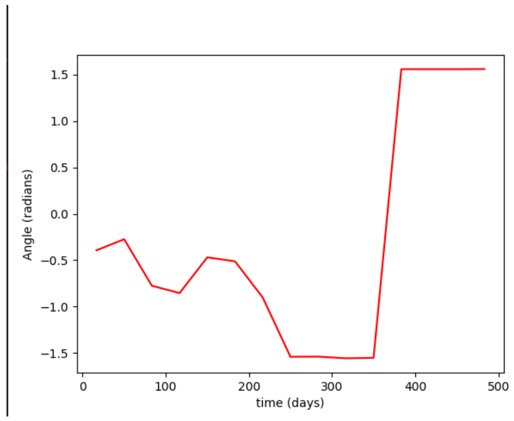

Figure 10: Optimal computed values of $\alpha$ over time for the trajectory from Mars to Earth.

To perform all the simulations realized during this section we have used the same parameters used in section 4.1 and we have kept the number of partitions of the function $\alpha$ to 15.

## 6 Strengths and Weaknesses

On the one hand, we should mention that we have worked with a non-ideal solar sail, given by some specific constants in Eq. 16. Therefore, we are able to adapt our simulation to the material characteristic as we do not use an ideal one. Moreover, our program works fine calculating final relative velocities under $9\,km/s$. Likewise, we have the option of working with different values of the orientation angle $\alpha$ during the travel. We can easily adapt our program in order to calculate the best moment to throw the solar sail, instead of waiting until Earth and Mars are aligned.
On the other hand, we should consider some overflows given sometimes by the program. Unfortunately, we cannot ensure that the found optimal value is not a local minimum. Moreover, we need to do a first manual testing in order to approximate the best values and later use them in order to find the best ones. We also did not consider the possible difficulties to maneuver the sail, i.e. changing the $\alpha$ angle. In addition, not having an analytical result complicates more our work as we do not know an estimation of the expected value.

## 7 Future Work

There have been some extra factors to consider, which would accuracy our model result or, moreover, improve the optimization result. First of all, we are able to improve our simulation results accuracy by adding the gravitational attraction made from Earth and Mars to our sail, at least considering it when the sail is not to much far from one of the planets. Furthermore, we could consider the launch and landing and how the sail is affected by the planets atmospheres, rotation, etc.. However, in order to optimize more the area or time, we can also think about the gravity assist using the Moon, as we mentioned before.
In addition, we have no considered the sail configuration, which will be the best shape for instance, as we can see at [10] [11]. Besides, we should consider that not all the solar sail is a sail, so we may study the effect of the SRP in each part of it, and also think about compound solar sails as [12].
Finally, we shall consider some improvements which would help us going faster or using less area, as different materials or painted materials [13].

## 8 Conclusions

Logarithmic spiral trajectories model have been modeled successfully with our numerical method. The numerical model is solid and always ensures a ARV lower than the security velocity required. It gives stability for our optimal trajectory when varying the number of partitions of $\alpha$. This means that with a lower number of angle changes, the optimal trajectory can be performed. Our objective to reach a method to find the optimum of the trajectory from Earth to Mars starting from an aligned configuration of the planets has been reached. The same numerical method can be used to finding the optimal method for any initial positions of Earth and Mars, and this gives a general scenario where you can choose a better optimal trajectory if the date of launch is variable.

Our optimal values for the parameters of total mass equal to $2000\,kg$, a Solar Sail made of material of mass $7\,g/m^2$ and a security maximal velocity of $9\,km/s$ when arriving on Mars are: the minimal area of the Solar Soil is $183158\,m^2$, the minimal transit time is $500\,days$ and the maximal payload is $718\,kg$.

In our further analysis, we study the optimal trajectory in case that we could choose the day of launching, which means that the initial relative position of Earth and Mars could be different than the proposed one. In this case, the minimal area of the Solar Soil is $145950\,m^2$, the minimal transit time is $500\,days$ and the maximal payload is $978\,kg$.Therefore, we clearly see that the improvement of the maximal payload could be an important motivation to change the initial positions of Earth and Mars, id est, change the launching date. As our Python code has also a parameter defined as the angle between the Earth and Mars, or any two planets, we see that the optimal value for launching the Spacecraft is for an angle of $1.14\,rad$.

## References

[1] N. Sands, "Generalised three-dimensional trajectory analysis of planetary escape by solar sail," in *American Rocket Society Journal*, no. 31, pp. 527–531, 1962.

[2] W. Fimple, "Escape from planetary gravitational fields by use of solar sails," in *American Rocket Society Journal*, no. 32, pp. 883–887, 1961.

[3] Y. I. H. Furuya and T. Masuoka, "Deployment characteristics of rotationally skew fold membrane for spinning solar sail," in *AIAA/ASME/ASCEA/AHS/ASC Structures, Structural Dynamics and Materials Conference*, vol. 46, pp. 139–143, April 2005.

[4] R. H. Frisbee, "Advanced space propulsion for the 21st century," in *Journal of Propulsion and Power*, vol. 19, pp. 1129–1154, November 2003.

[5] K. Tetzlaff, "Solar Sails: An Analysis of Interplanetary Logarithmic Spiral Trajectories," 2004.

[6] D. Shortt, "Gravity assist," *The Planetary Society*.

[7] J. L. Wright, *Space Sailing*. Gordon and Breach Science Publishers, 1992.

[8] M. . Hughes, "Solar sailing lectures," in *Summer Workshop on Advanced Topics in Astrodynamics*, July 2004.

[9] M. Zhang, G. Wang, and Y. Sun, "Solar sail trajectory optimization for earth-mars mission," in *Proceedings of the 29th Chinese Control Conference*, pp. 139–143, July 2010.

[10] unknown, "Solar Sail Technology Development: Design & Construction." https://web.archive.org/web/20050311004606/http://solarsails.jpl.nasa.gov/introduction/design-construction.html. Accessed: 2017-11-12.

[11] M. Macdonald, *Advances in Solar Sailing*. Springer and Praxis Publishing, 2014.

[12] G. Mengali and A. A. Quarta, "Compound solar sail with optical properties: Models and performance," vol. 43, pp. 239–245, 01 2006.

[13] B. Christensen, "Earth to mars in a month with painted solar sail," 02 2005.

[14] B. Dachwald, "Optimal solar sail trajectories for missions to the outer solar system," in *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, August 2004.

# Appendix

## A Deducing motion equations for Hohmann transfer

In the first instance, the sun-centred orbital dynamics of solar sail spacecraft will be described, since this is traditionally where solar sailing is applied and where the bulk of the mission concepts exist, due to the large $\Delta v$ requirements [14].The vector equation of motion of a solar sail spacecraft moving in a heliocentric orbit is defined by Eq. 31, where $r$ is the position vector of the spacecraft with respect to the Sun, at time $t$. The gravitational parameter of the Sun is defined by $\mu$. For an ideal sail, the thrust vector is aligned along the sail normal direction, $n$.

$$\frac{d^2\overrightarrow{r}}{dt^2} + \frac{\mu}{r^2}\hat{r} = \beta\frac{\mu}{r^2}(\hat{r}\cdot\overrightarrow{n})^2 \tag{31}$$

For the simple analysis to follow, it is appropriate to show how this equation is represented in two-dimensional polar coordinates, shown in Eq. 32 and 33 Here, $\theta$ is the azimuth angle from the x-axis, $r$ is the distance of the spacecraft from the Sun. The pitch or cone angle, $\alpha$ is defined as the angle between the sail normal and the radial vector, as defined in the SRP theoretical model.

$$\frac{d^2r}{dt^2} - r\left[\frac{d\theta}{dt}\right]^2 = -\frac{\mu}{r^2} + \beta\frac{\mu}{r^2}cos^3\alpha \tag{32}$$

$$r\frac{d^2\theta}{dt^2} + 2r\left[\frac{dr}{dt}\right]\left[\frac{d\theta}{dt}\right] = \beta\frac{\mu}{r^2}cos^2\alpha sin\alpha \tag{33}$$

The parameter, $\beta$ is known as the sail lightness number and is defined as the ratio of the local solar radiation pressure acceleration produced by the sail to the local solar gravitational acceleration. This number is a useful parameter since it is independent of the solar distance and defines the performance of the sail. Another useful performance parameter is the sail characteristic acceleration, $a_c$, which is the solar radiation pressure induced acceleration of an ideal sail, pitched face on to the Sun ($\alpha = 0$) at $1AU$ from the Sun. It is easy to calculate that a characteristic acceleration of $1.0mms^{-2}$ corresponds to a dimensionless sail lightness number of 0.1686. As has been discussed the solar sail characteristic acceleration is dependent on the surface area and reflectivity of the sail assembly and the mass of the spacecraft [8].

## B Codes

Codes in Python:

```
from matplotlib import pyplot as plt
from matplotlib import animation
from math import cos, sin, sqrt, pi
import json
from scipy.integrate import ode
from scipy.optimize import fmin_slsqp

H_TO_S = 3600
```

```
S_TO_H = 1 / 3600
M_TO_KM = 1 / 1000
KM_TO_M = 1000
G_TO_KG = 1 / 1000
KG_TO_G = 1000
CM_TO_KM = 1 / 100000
KM_TO_CM = 100000
MICRO_TO_UNI = 1E-6
UNI_TO_MICRO = 1E6

DISTANCE_SUN_EARTH = 149600000 # Km
DISTANCE_SUN_MARS = 227940000 # Km
DISTANCE_SUN_JUPITER = 7786000000 # Km

EARTH_ORBITAL_VELOCITY = 29.8 / S_TO_H # Km/h
MARS_ORBITAL_VELOCITY = 24.1 / S_TO_H # Km/h
EARTH_ANGULAR_VELOCITY = EARTH_ORBITAL_VELOCITY / DISTANCE_SUN_EARTH # rad/h
MARS_ANGULAR_VELOCITY = MARS_ORBITAL_VELOCITY / DISTANCE_SUN_MARS # rad/h

SUN_RADIUS = 695700 # Km
SUN_MASS = 1988500E4 # 10^20Kg

GRAVITATIONAL_CONSTANT = 6.67408 * 1E9 * M_TO_KM ** 3 / S_TO_H ** 2 # 10^(-20) Km^3/(kg*h^2)
GM = GRAVITATIONAL_CONSTANT * SUN_MASS # Km^3/h^2

MAX_SPEED_DIFFER = 9 / S_TO_H # Km/h

MARS_RADIUS = 6792 / 2 # Km
MARS_EARTH_INITIAL_ANGLE_DIFF = 0 # rad
EARTH_ESCAPE_VELOCITY = 11.186 / S_TO_H # Km/h
ODE_SOLVER_STEP = 1 / 500

MAX_WEIGHT = 2E3 # Kg maximum weight we can carry with our sail
SAIL_DENSITY = 7 * G_TO_KG / CM_TO_KM ** 2 # Kg/Km^2
P0 = 4.563 * MICRO_TO_UNI / (M_TO_KM * S_TO_H ** 2) # (Kg/Km*h^2)
r02 = DISTANCE_SUN_EARTH ** 2 # Km
ac = 2 * P0 * r02 / MAX_WEIGHT # Km/h^2
b1 = 0.0864
b2 = 0.8277
b3 = -5.45e-3
N = 15 # Default number of partitions
AREA_RELEVANCE = 1. # default relevance given to area minimization
TIME_RELEVANCE = 1. # default relevance given to time minimization
R_init = DISTANCE_SUN_EARTH
R_fin = DISTANCE_SUN_MARS


# define a function which given a value s between 0 and 1 returns the angle that the sail should
    be
# form with the position vector at the corresponding time.
def alpha(s, alpha_list):
    s = min(s, 0.99)
    return alpha_list[int(s * len(alpha_list))]
```

```
# Define the ordinary differential equation we must solve in order to compute the final position
# and velocity of our solar wind propelled rocket first parameter is radius (from Sun), second is
# angle (respect to initial angle) third is radial speed and last is angular speed. The
    parameters
# consist on a list of N+3 doubles. First N doubles correspond to the angle the sail is forming
    with
# the position vector taking the Sun as coordinates origin. The fourth to last parameter is a
    value
# between 0 and 1 which indicates the initial speed the sail has whith respect to the Earth
# (in proortion to escape velocity). The third to last parameter is the initial
# angle that the ship forms with Earth's orbit. The second to last parameter is the sail area
# and the last parameter is the expected time the trip is going to last.
def f(t, y, params):
    r, theta, u, v = y
    A = params[-2]
    tf = params[-1]
    r_deriv = tf * u
    theta_deriv = tf * v / r
    u_deriv = tf * (v ** 2 - GM / r) / r
    v_deriv = -tf * u * v / r
    a = alpha(t, params[:-4])
    cosalpha = cos(a)
    sinalpha = sin(a)
    aux = tf * ac * A * cosalpha / r ** 2
    u_deriv += aux * (b1 + cosalpha * (b2 * cosalpha + b3))
    v_deriv += aux * sinalpha * (b2 * cosalpha + b3)
    return [r_deriv, theta_deriv, u_deriv, v_deriv]


# define function that solves the differential equation and returns the final values of position
    and speed in
# polar coordinates.
def solve_ode(params):
    # Initialize an object to solve the previous differential equation using Runge-Kutta 4(5)
        method.
    ode_solver = ode(f).set_integrator('dopri5', nsteps=10000)
    # Set initial value
    theta_init = params[-3]
    v_init = params[-4]
    ode_solver.set_initial_value([DISTANCE_SUN_EARTH, 0, sin(theta_init) * v_init,
                                  cos(theta_init) * v_init + EARTH_ORBITAL_VELOCITY],
                                 0)
    ode_solver.set_f_params(params)
    while ode_solver.successful() and ode_solver.t < 1:
        ode_solver.integrate(ode_solver.t + ODE_SOLVER_STEP)

    return ode_solver.y


# define function that solves the differential equation and returns the final values of position
    and speed in
# polar coordinates.
def solve_ode_orbital(params):
    global R_init
    # Initialize an object to solve the previous differential equation using Runge-Kutta 4(5)
```

```
        method.
    ode_solver = ode(f).set_integrator('dopri5', nsteps=10000)
    # Set initial value
    theta_init = params[-3]
    v_init = params[-4]
    ode_solver.set_initial_value([R_init, 0, sin(theta_init) * v_init,
                                  cos(theta_init) * v_init + sqrt(GM/R_init)],
                                 0)
    ode_solver.set_f_params(params)
    while ode_solver.successful() and ode_solver.t < 1:
        ode_solver.integrate(ode_solver.t + ODE_SOLVER_STEP)

    return ode_solver.y


# Our goal is that of maximizing the amount of material we can receive and return per time unit.
def objective(params):
    global AREA_RELEVANCE, TIME_RELEVANCE
    r_fin, theta_fin, radial_speed, theta_speed = solve_ode(params)
    tang_speed_diff = (theta_speed - DISTANCE_SUN_MARS * MARS_ANGULAR_VELOCITY)
    speed_condition = sqrt(max(tang_speed_diff ** 2 + radial_speed ** 2 - MAX_SPEED_DIFFER ** 2,
        0))

    Mars_theta = MARS_EARTH_INITIAL_ANGLE_DIFF + MARS_ANGULAR_VELOCITY * params[-1]
    x_diff = r_fin * cos(theta_fin) - DISTANCE_SUN_MARS * cos(Mars_theta)
    y_diff = r_fin * sin(theta_fin) - DISTANCE_SUN_MARS * sin(Mars_theta)
    pos_condition = sqrt(max(x_diff ** 2 + y_diff ** 2 - MARS_RADIUS ** 2, 0))
    # we square the value to make sure it is not negative.
    # this option results in error, the NLP solver learns to make the value params[-1]
        infinitely small.
    # return -((MAX_WEIGHT - SAIL_DENSITY * params[-2]) / params[-1]) ** 2
    # we add constraints (terms multiplied by 1e8) in order to achieve the desired results that
        being a small
    # final distance to Mars (of the order of Mars radius) and a small relative speed to Mars
        (smaller that 9 Km/s)
    time_condition = TIME_RELEVANCE * abs(params[-1])
    area_condition = AREA_RELEVANCE * params[-2] ** 2
    return pos_condition * 1E-4 + speed_condition * 5E-3 + time_condition / 180 + area_condition
        * 5


# Our goal is that of maximizing the amount of material we can receive and return per time unit.
def orbit_objective(params):
    global R_fin
    r_fin, theta_fin, radial_speed, theta_speed = solve_ode_orbital(params)
    tang_speed_diff = abs(theta_speed - sqrt(GM/R_fin))
    speed_condition = sqrt(max(tang_speed_diff**2 + radial_speed ** 2 - MAX_SPEED_DIFFER ** 2,
        0))

    orbit_condition = abs(r_fin - R_fin)
    # we square the value to make sure it is not negative.
    # we add constraints (terms multiplied by 1e8) in order to achieve the desired results that
        being a small
    # final distance to Mars (of the order of Mars radius) and a small relative speed to Mars
        (smaller that 9 Km/s)
    time_condition = TIME_RELEVANCE * abs(params[-1])
    area_condition = AREA_RELEVANCE * params[-2] ** 2
    return orbit_condition * 1E-4 + speed_condition * 5E-3 + time_condition / 180 +
        area_condition * 5


def find_optimal_values(n=N, verbose=True, area_relevance=AREA_RELEVANCE,
    time_relevance=TIME_RELEVANCE):
    global AREA_RELEVANCE, TIME_RELEVANCE
    AREA_RELEVANCE = area_relevance
    TIME_RELEVANCE = time_relevance
    # Define constraints for the angles values (angles must be in the range [-pi/2, pi/2])
    bounds = [(-pi / 2, pi / 2) for i in range(n)]
    # Define bounds for initial speed respect to Earth (from 0 to escape velocity).
    bounds.append((0, EARTH_ESCAPE_VELOCITY))
    # Define bounds for the initil angle formed with Earth.
    bounds.append((-pi, pi))
    # Define constraints for the Area of the sail (area must be between 0 and maximum
        transportable weight.
    bounds.append((0, 0.28))
    # Define time constraints between 10 hours and 1.8 years
    bounds.append((10, 24 * 1.8 * 365))

    initial = [11 * pi / 180 for i in range(n)] # angles
    initial.append(EARTH_ESCAPE_VELOCITY / 100) # Initial speed relative to Earth
    initial.append(0) # initial angle with Earth orbit
    initial.append(0.14) # Area sail.
    initial.append(500 * 24) # Time.

    if verbose:
        results = fmin_slsqp(objective, initial, bounds=bounds, iprint=2, iter=1000)
    else:
        results = fmin_slsqp(objective, initial, bounds=bounds, iter=1000)

    show_results(results)
    return results


def show_results(params):
    angles = params[:-4]
    speed_init, angle_init, sail_size, time = params[-4:]
    print("\nThe optimal angle values are:")
    for angle in angles:
        print("\t{:.2f} radians".format(angle))

    print("\nThe optimal initial speed respect to Earth is:")
    print("\t{:.2f} Km/s".format(speed_init / H_TO_S))

    print("\nThe optimal initial angle formed with the Earth's orbit is:")
    print("\t{:.2f} radians".format(angle_init))

    print("\nThe optimal sail size is:")
    print("\t{:.2f} Km^2".format(sail_size))

    print("\nThe optimal travel time is:")
    print("\t{:.2f} days".format(time / 24))

    r_fin, theta_fin, radial_speed, tangential_speed = solve_ode(params)
    print("\nSpeed difference between solar sail and Mars is:")
    tang_speed_diff = tangential_speed - DISTANCE_SUN_MARS * MARS_ANGULAR_VELOCITY
    print("\t{:.2f} Km/s".format(sqrt(tang_speed_diff ** 2 + radial_speed ** 2) / H_TO_S))

    print("\nDistance between the center of Mars (with radius 3396 Km) and the solar sail at
        trajectory end is:")
    theta_mars = MARS_EARTH_INITIAL_ANGLE_DIFF + MARS_ANGULAR_VELOCITY * params[-1]
    x_diff = (r_fin * cos(theta_fin) - DISTANCE_SUN_MARS * cos(theta_mars))
    y_diff = (r_fin * sin(theta_fin) - DISTANCE_SUN_MARS * sin(theta_mars))
    print("\t{:.2f} Km".format(sqrt(x_diff ** 2 + y_diff ** 2)))


def show_orbital_results(params):
    angles = params[:-4]
    speed_init, angle_init, sail_size, time = params[-4:]
    print("\nThe optimal angle values are:")
    for angle in angles:
        print("\t{:.2f} radians".format(angle))

    print("\nThe optimal initial speed respect to Earth is:")
    print("\t{:.2f} Km/s".format(speed_init / H_TO_S))

    print("\nThe optimal initial angle formed with the Earth's orbit is:")
    print("\t{:.2f} radians".format(angle_init))

    print("\nThe optimal sail size is:")
    print("\t{:.2f} Km^2".format(sail_size))

    print("\nThe optimal travel time is:")
    print("\t{:.2f} days".format(time / 24))

    r_fin, theta_fin, radial_speed, tangential_speed = solve_ode_orbital(params)
    print("\nSpeed difference between solar sail Orbital speed and Mars is:")
    tang_speed_diff = tangential_speed - sqrt(GM/R_fin)
    print("\t{:.2f} Km/s".format(sqrt(tang_speed_diff ** 2 + radial_speed ** 2) / H_TO_S))

    print("\nDistance between the solar sail and the orbit:")
    print("\t{:.2f} Km".format(abs(r_fin-R_fin)))

    print("\nThe Mars angle respect Earth should be:")
    print("\t{:.2f} radians".format(theta_fin - MARS_ANGULAR_VELOCITY * params[-1]))


# Define function that solves the differential equation and .
def generate_plotting_data(params):
    # Initialize an object to solve the previous differential equation using Runge-Kutta 4(5)
        method.
    ode_solver = ode(f).set_integrator('dopri5', nsteps=10000)
    # Set initial value
    theta_init = params[-3]
    v_init = params[-4]
    ode_solver.set_initial_value([DISTANCE_SUN_EARTH, 0, sin(theta_init) * v_init,
                              cos(theta_init) * v_init + EARTH_ORBITAL_VELOCITY],
                             0)
    ode_solver.set_f_params(params)
    positions = []
    while ode_solver.successful() and ode_solver.t + ODE_SOLVER_STEP < 1:
        ode_solver.integrate(ode_solver.t + ODE_SOLVER_STEP)
        positions.append([ode_solver.y[0] * cos(ode_solver.y[1]), ode_solver.y[0] *
            sin(ode_solver.y[1])])

    return positions


# Define function that solves the differential equation and .
def generate_orbital_plotting_data(params):
    # Initialize an object to solve the previous differential equation using Runge-Kutta 4(5)
        method.
    ode_solver = ode(f).set_integrator('dopri5', nsteps=10000)
    # Set initial value
    theta_init = params[-3]
    v_init = params[-4]
    ode_solver.set_initial_value([R_init, 0, sin(theta_init) * v_init,
                              cos(theta_init) * v_init + sqrt(GM/R_init)],
                             0)
    ode_solver.set_f_params(params)
    positions = []
    while ode_solver.successful() and ode_solver.t + ODE_SOLVER_STEP < 1:
        ode_solver.integrate(ode_solver.t + ODE_SOLVER_STEP)
        positions.append([ode_solver.y[0] * cos(ode_solver.y[1]), ode_solver.y[0] *
            sin(ode_solver.y[1])])

    return positions


# Creating celestial_object class
class CelestialObject(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def move(self, newx, newy):
        self.x = newx
        self.y = newy


def save_params(params, file='learned_parameters.json'):
    angles = [angle for angle in params[:-4]]
    results = {'angles': angles,
               'initial_speed': params[-4],
               'initial_angle': params[-3],
               'sail_size': params[-2],
               'time': params[-1]}
    with open(file, 'w') as outfile:
        json.dump(results, outfile)
```

```
def load_params(file='learned_parameters.json'):
    json_data = open("learned_parameters.json", "r")
    results = json.load(json_data)
    params = [results['initial_speed'], results['initial_angle'], results['sail_size'],
        results['time']]
    params = results['angles'] + params
    return params


def show_animated_plot(params):
    positions = generate_plotting_data(params)

    tf = params[-1]
    n = len(positions)
    step = tf / n

    # Initializing dots
    Earth = CelestialObject(DISTANCE_SUN_EARTH, 0)
    Ship = CelestialObject(DISTANCE_SUN_EARTH, 0)
    Mars = CelestialObject(DISTANCE_SUN_MARS, 0)

    plt.clf()
    fig = plt.figure()
    ax = plt.axes(xlim=(-2 * DISTANCE_SUN_MARS, 2 * DISTANCE_SUN_MARS),
                  ylim=(-2 * DISTANCE_SUN_MARS, 2 * DISTANCE_SUN_MARS))
    d, = ax.plot([Earth.x, Mars.x, Ship.x],
                 [Earth.y, Mars.y, Ship.y], 'ro', markersize=4)
    circle = plt.Circle((0, 0), DISTANCE_SUN_MARS, color='r', fill=False)
    ax.add_artist(circle)
    circle = plt.Circle((0, 0), DISTANCE_SUN_EARTH, color='b', fill=False)
    ax.add_artist(circle)
    circle = plt.Circle((0, 0), 10 * SUN_RADIUS, color='y')
    ax.add_artist(circle)

    # animation function. This is called sequentially
    def animate(i):
        Earth_theta = i * step * EARTH_ANGULAR_VELOCITY
        Earth.move(DISTANCE_SUN_EARTH * cos(Earth_theta), DISTANCE_SUN_EARTH * sin(Earth_theta))
        Mars_theta = MARS_EARTH_INITIAL_ANGLE_DIFF + i * step * MARS_ANGULAR_VELOCITY
        Mars.move(DISTANCE_SUN_MARS * cos(Mars_theta), DISTANCE_SUN_MARS * sin(Mars_theta))
        x, y = positions[i]
        Ship.move(x, y)
        d.set_data([Earth.x, Mars.x, Ship.x],
                   [Earth.y, Mars.y, Ship.y])
        plt.title("Time: {:.2f} days".format(i * step / 24))
        return d,

    # call the animator.
    anim = animation.FuncAnimation(fig, animate, frames=n, interval=5)
    plt.show()


def show_orbital_animated_plot(params):
    positions = generate_orbital_plotting_data(params)
    tf = params[-1]
    n = len(positions)
    step = tf / n

    # Initializing dots
    Earth = CelestialObject(DISTANCE_SUN_EARTH, 0)
    Ship = CelestialObject(DISTANCE_SUN_EARTH, 0)
    Mars = CelestialObject(DISTANCE_SUN_MARS, 0)

    plt.clf()
    fig = plt.figure()
    ax = plt.axes(xlim=(-2 * DISTANCE_SUN_MARS, 2 * DISTANCE_SUN_MARS),
                  ylim=(-2 * DISTANCE_SUN_MARS, 2 * DISTANCE_SUN_MARS))
    d, = ax.plot([Earth.x, Mars.x, Ship.x],
                 [Earth.y, Mars.y, Ship.y], 'ro', markersize=4)
    circle = plt.Circle((0, 0), DISTANCE_SUN_MARS, color='r', fill=False)
    ax.add_artist(circle)
    circle = plt.Circle((0, 0), DISTANCE_SUN_EARTH, color='b', fill=False)
    ax.add_artist(circle)
    circle = plt.Circle((0, 0), 10 * SUN_RADIUS, color='y')
    ax.add_artist(circle)

    # animation function. This is called sequentially
    def animate(i):
        Earth_theta = i * step * EARTH_ANGULAR_VELOCITY
        Earth.move(DISTANCE_SUN_EARTH * cos(Earth_theta), DISTANCE_SUN_EARTH * sin(Earth_theta))
        Mars_theta = MARS_EARTH_INITIAL_ANGLE_DIFF + i * step * MARS_ANGULAR_VELOCITY
        Mars.move(DISTANCE_SUN_MARS * cos(Mars_theta), DISTANCE_SUN_MARS * sin(Mars_theta))
        x, y = positions[i]
        Ship.move(x, y)
        d.set_data([Earth.x, Mars.x, Ship.x],
                   [Earth.y, Mars.y, Ship.y])
        plt.title("Time: {:.2f} days".format(i * step / 24))
        return d,

    # call the animator.
    anim = animation.FuncAnimation(fig, animate, frames=n, interval=5)
    plt.show()


def save_animated_plot(params, file_name='animated_plot.html'):
    Writer = animation.writers['ffmpeg']
    writer = Writer(fps=15, metadata=dict(artist='Team 744'), bitrate=1800)

    positions = generate_plotting_data(params)
    tf = params[-1]
    n = len(positions)
    step = tf / n

    # Initializing dots
    Earth = CelestialObject(DISTANCE_SUN_EARTH, 0)
    Ship = CelestialObject(DISTANCE_SUN_EARTH, 0)
    Mars = CelestialObject(DISTANCE_SUN_MARS, 0)

    fig = plt.figure()
    ax = plt.axes(xlim=(-2 * DISTANCE_SUN_MARS, 2 * DISTANCE_SUN_MARS),
                 ylim=(-2 * DISTANCE_SUN_MARS, 2 * DISTANCE_SUN_MARS))
    d, = ax.plot([Earth.x, Mars.x, Ship.x],
                [Earth.y, Mars.y, Ship.y], 'ro', markersize=4)
    circle = plt.Circle((0, 0), DISTANCE_SUN_MARS, color='r', fill=False)
    ax.add_artist(circle)
    circle = plt.Circle((0, 0), DISTANCE_SUN_EARTH, color='b', fill=False)
    ax.add_artist(circle)
    circle = plt.Circle((0, 0), 10 * SUN_RADIUS, color='y')
    ax.add_artist(circle)

    # animation function. This is called sequentially
    def animate(i):
        Earth_theta = i * step * EARTH_ANGULAR_VELOCITY
        Earth.move(DISTANCE_SUN_EARTH * cos(Earth_theta), DISTANCE_SUN_EARTH * sin(Earth_theta))
        Mars_theta = i * step * MARS_ANGULAR_VELOCITY
        Mars.move(DISTANCE_SUN_MARS * cos(Mars_theta), DISTANCE_SUN_MARS * sin(Mars_theta))
        x, y = positions[i]
        Ship.move(x, y)
        d.set_data([Earth.x, Mars.x, Ship.x],
                  [Earth.y, Mars.y, Ship.y])
        plt.title("Time: {:.2f} days".format(i * step / 24))
        return d,

    # call the animator.
    anim = animation.FuncAnimation(fig, animate, frames=n, interval=5)
    anim.save(filename=file_name, writer=writer)


def plot_trajectory(params):
    positions = generate_plotting_data(params)
    x = [pos[0] for pos in positions]
    y = [pos[1] for pos in positions]
    plt.clf()
    plt.plot(x, y, 'r', color='black')
    N = 100
    x = [DISTANCE_SUN_EARTH * cos(2 * pi * i / N) for i in range(N + 1)]
    y = [DISTANCE_SUN_EARTH * sin(2 * pi * i / N) for i in range(N + 1)]
    plt.plot(x, y, 'r', color='blue')
    x = [DISTANCE_SUN_MARS * cos(2 * pi * i / N) for i in range(N + 1)]
    y = [DISTANCE_SUN_MARS * sin(2 * pi * i / N) for i in range(N + 1)]
    plt.plot(x, y, 'r', color='red')
    plt.title('{} angle variations'.format(len(params) - 4))
    show_results(params)
    plt.show()


def save_trajectory(params, file_name='animated_plot.mp4'):
    positions = generate_plotting_data(params)
    x = [pos[0] for pos in positions]
    y = [pos[1] for pos in positions]
    plt.clf()
    plt.plot(x, y, 'r', color='black')
    N = 100
    x = [DISTANCE_SUN_EARTH * cos(2 * pi * i / N) for i in range(N + 1)]
    y = [DISTANCE_SUN_EARTH * sin(2 * pi * i / N) for i in range(N + 1)]
    plt.plot(x, y, 'r', color='blue')
    x = [DISTANCE_SUN_MARS * cos(2 * pi * i / N) for i in range(N + 1)]
    y = [DISTANCE_SUN_MARS * sin(2 * pi * i / N) for i in range(N + 1)]
    plt.plot(x, y, 'r', color='red')
    plt.title('{} angle variations'.format(len(params) - 4))
    plt.savefig(file_name)


def save_orbital_trajectory(params, file_name='animated_plot.mp4'):
    positions = generate_orbital_plotting_data(params)
    x = [pos[0] for pos in positions]
    y = [pos[1] for pos in positions]
    plt.clf()
    plt.plot(x, y, 'r', color='black')
    N = 100
    x = [R_init * cos(2 * pi * i / N) for i in range(N + 1)]
    y = [R_init * sin(2 * pi * i / N) for i in range(N + 1)]
    plt.plot(x, y, 'r', color='blue')
    x = [R_fin * cos(2 * pi * i / N) for i in range(N + 1)]
    y = [R_fin * sin(2 * pi * i / N) for i in range(N + 1)]
    plt.plot(x, y, 'r', color='red')
    plt.title('{} angle variations'.format(len(params) - 4))
    plt.savefig(file_name)


def save_areatime(params, file='learned_parameters.json'):
    angles = [angle for angle in params[:-4]]
    results = {'angles': angles,
               'initial_speed': params[-4],
               'initial_angle': params[-3],
               'sail_size': params[-2],
               'time': params[-1]}
    with open(file, 'w') as outfile:
        json.dump(results, outfile)


def plot_angles(params):
    angles = params[:-4]
    n = len(angles)
    t = params[-1] / n
    x = [(i + 1 / 2) * t / 24 for i in range(n)]
    plt.clf()
    plt.xlabel('time (days)')
    plt.ylabel('Angle (radians)')
    plt.plot(x, angles, 'r')
    plt.show()


def plot_multiple_angles(params_list):
    angles_list = [params[:-4] for params in params_list]
    x_list = [[(i + 1 / 2) * params_list[j][-1] / (len(angles_list[j])*24) for i in
        range(len(angles_list[j]))] for j in
            range(len(angles_list))]
    plt.clf()
    plt.xlabel('time (days)')
    plt.ylabel('Angle (radians)')
    lines = []
    for i in range(len(angles_list)):
        line, = plt.plot(x_list[i], angles_list[i], label=str(len(x_list[i])))
        lines.append(line)

    plt.legend(handles=lines)
    plt.show()


def find_optimal_value_list(n_list, verbose=False, area_relevance=AREA_RELEVANCE,
     time_relevance=TIME_RELEVANCE):
    param_list = [find_optimal_values(n, verbose, area_relevance=area_relevance,
         time_relevance=time_relevance) for n in
                  n_list]
    return param_list


def optimizing_area_time(n=20):
    x = []
    y = []
    d = []
    s = []
    for i in range(n):
        params = find_optimal_values(15, True, area_relevance=i / n, time_relevance=(1 - i / n))
        angles = params[:-4]
        init_speed, init_angle, area, time = params[-4:]
        r_fin, theta_fin, radial_speed, tangential_speed = solve_ode(params)
        tang_speed_diff = tangential_speed - DISTANCE_SUN_MARS * MARS_ANGULAR_VELOCITY
        s.append(sqrt(tang_speed_diff ** 2 + radial_speed ** 2) / H_TO_S)
        theta_mars = MARS_EARTH_INITIAL_ANGLE_DIFF + MARS_ANGULAR_VELOCITY * params[-1]
        x_diff = (r_fin * cos(theta_fin) - DISTANCE_SUN_MARS * cos(theta_mars))
        y_diff = (r_fin * sin(theta_fin) - DISTANCE_SUN_MARS * sin(theta_mars))
        d.append(sqrt(x_diff ** 2 + y_diff ** 2))
        x.append(area)
        y.append(time / 24)

    print("n \t area (km^2)\t time (days) \t Dv (km/s)\t dist (km) \n")
    for i in range(n):
        print(i, "\t", x[i], "\t", y[i], "\t", s[i], "\t", d[i], "\n")


# apply the same algorithm bu to transfer from orbit to orbit and not from planet to planet
def inter_orbital_transport(n=N, verbose=True, initial_orbit_radius=DISTANCE_SUN_EARTH,
                         final_orbit_radius=DISTANCE_SUN_MARS):
    global R_init, R_fin
    R_init = initial_orbit_radius
    R_fin = final_orbit_radius
    # Define constraints for the angles values (angles must be in the range [-pi/2, pi/2])
    bounds = [(-pi / 2, pi / 2) for i in range(n)]
    # Define bounds for initial speed respect to Earth (from 0 to escape velocity).
    bounds.append((0, EARTH_ESCAPE_VELOCITY))
    # Define bounds for the initil angle formed with Earth.
    bounds.append((-pi, pi))
    # Define constraints for the Area of the sail (area must be between 0 and maximum
        transportable weight.
    bounds.append((0, 0.28))
    # Define time constraints between 10 hours and 15 years
    bounds.append((10, 24 * 15 * 365))

    initial = [11 * pi / 180 for i in range(n)] # angles
    initial.append(EARTH_ESCAPE_VELOCITY / 100) # Initial speed relative to Earth
    initial.append(0) # initial angle with Earth orbit
    initial.append(0.14) # Area sail.
    initial.append(500 * 24) # Time.

    if verbose:
        results = fmin_slsqp(orbit_objective, initial, bounds=bounds, iprint=2, iter=1000)
    else:
        results = fmin_slsqp(orbit_objective, initial, bounds=bounds, iter=1000)

    show_orbital_results(results)
    return results


def test_model():
    n_list = [10, 15, 20, 25, 30]
    # n_list = [1, 2, 3]
    param_list = find_optimal_value_list(n_list, False)
    # param_list = [load_params('learned_parameters_n={}.json'.format(n)) for n in n_list]
    for params in param_list:
        n = len(params) - 4
        save_params(params, 'learned_parameters_n={}.json'.format(n))
        # save_animated_plot(params, 'animated_plot_n={}.html'.format(n))
        save_trajectory(params, 'trajectory_plot_n={}.png'.format(n))

    n = input('insert number of different angles in the simulation you want to see\n')
    params = load_params('learned_parameters_n={}.json'.format(n))
    show_animated_plot(params)
    plot_multiple_angles(param_list)


def test_inter_orbital():
    params = inter_orbital_transport(initial_orbit_radius=DISTANCE_SUN_MARS,
        final_orbit_radius=DISTANCE_SUN_EARTH)
    save_params(params, 'learned_orbital_parameters_earth_mars.json')
    save_orbital_trajectory(params, 'learned_orbital_parameters_mars_earth.png')
    plot_angles(params)
    show_orbital_animated_plot(params)
    print("")


if __name__ == '__main__':
    # test_model()
    test_inter_orbital()
```