Engineering Optimization: Concepts and Applications

Final Report

Optimization of a trajectory from the Earth to Mars using electric propulsion

Veronica Saz Ulibarrena, 4867491

1 Introduction

A determinant factor in the success of an interplanetary mission is the calculation of the mass of fuel that is needed. In order to increase the payload mass that can be carried, the mass of fuel must be reduced.

Therefore, the goal of this project is optimizing the mass of fuel used for an interplanetary transfer between the Earth and Mars using electric propulsion. The implication of using this kind of propulsion is that the thrust is applied continuously instead of in instantaneous impulses. Therefore, modelling this types of trajectories requires large computation power.

A method to solve this problem is to use Sims-Flanagan formulation, which models the continuous impulse as a series of discrete ones that are equally distributed in time. Therefore, the trajectory from the Earth is propagated and the velocity is modified at the corresponding time.

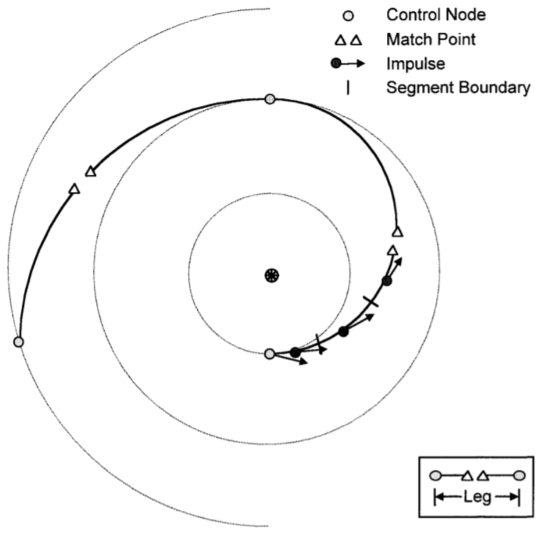


Figure 1: Representation of the trajectory divided into segments. Taken from Sims and Flanagan, *Preliminary design of low-thrust interplanetary missions* [1].

The following problem has been solved by programming the different parts in `Python`. Except for the Monotonic Basin Hopping Algorithm that is already implemented within `scipy.optimize`, the rest of the algorithms have been completely programmed from scratch.

Along the report, much of the information has been taken from the Slides of the course [2], along with articles related to this problem and the best way to optimize it [3], [1].

2 Problem formulation

Description of the model

In order to calculate the trajectory that will be optimized, a function has been created to propagate the position and velocity of the spacecraft from the Earth to every consequent time. To do so, some initial parameters have to be passed to the function. These can be divided into two categories: fixed or decision variables.

Among the fixed, the two first have been chosen according to Dawn mission's parameters [4].

- Dry mass of the spacecraft (m_{dry}): 747 kg.
- Maximum thrust that can be provided (T): 0.09 N.
- Number of impulses that will simulate the constant thrust (N_{imp}): 30. This value has been chosen so that the number of impulses is as large as possible to get an accurate comparison, but low enough to be achievable in a reasonable computation time.

The values that will be decision variables are:

- Time of flight: time it takes the spacecraft to go from the Earth to Mars. This value will affect the position of the planets since what is needed is the position of the Earth at departure and the position of Mars at arrival. Therefore, this is what will be plotted in all the following plots.
- Initial velocity from the Earth: this value is the magnitude of the initial impulse used to escape Earth's orbit and enter the interplanetary trajectory. The goal of electric propulsion is to minimize the initial and final impulses needed, so this value is much lower than for an impulsive transfer. It is assumed to be applied in the direction of the velocity of the Earth for simplification purposes. In contrast to the other impulses, in some cases this can be provided by a chemical source, allowing for larger impulses.
- ΔV: vector with the magnitude of each of the impulses applied. This is set as a value between 0 and 1 that represents the fraction over the maximum impulse that can be achieved:

$$\Delta V_{max} = \frac{T \cdot t_{flight}}{m_{dry} \cdot (N_{imp} - 1)}$$

- Angle of ΔV: angle that indicated the direction in which the impulse is applied with respect to the cartesian reference frame.

Therefore, with those values the trajectory can be completely determined. The first step is calculating the position and velocity of the two planets (at departure time for the Earth and at arrival date, which means departure time plus the time of flight for Mars). With the values at the Earth, the spacecraft state is propagated to the point in which the first impulse is to be applied, and the velocity is increased by the corresponding magnitude. This process is repeated until the maximum number of impulses is achieved.

Finally, the final position and velocity of the spacecraft are compared to those of Mars, since those should match for rendez-vous.

The mass employed can be calculated as the sum of the mass consumed for each of the impulses, including that for the initial impulse from the Earth to assume that it can be performed by electric propulsion.

Assumptions

- Although the orbits of the Earth and Mars do not lie in the same plane, it is assumed that the problem is planar and contained in the ecliptic (or orbital plane of the Earth). Also, it is assumed to be a restricted three body problem, which means that the spacecraft is considered a point mass and no perturbations have been taken into consideration in the calculation of the trajectory.
- The impulses are assumed to be instantaneous, producing a discrete change in velocity, which is applicable since the time it takes to apply an impulse is normally much lower than the total time of the mission.
- The fuel is assumed to be consumed completely by the end of the mission.
- When calculating the maximum ΔV, the mass used is not the current mass at that moment but the dry mass. Also, this value is calculated assuming that the thrust is the instant one plus the time between impulses, to adjust the value used in continuous thrust to the case with limited impulses.

Optimization problem

Once the model is defined, the goal is to optimize the mass of fuel employed. This means that the objective function is precisely that mass. This is equivalent to minimizing the total ΔV applied.

The values that were described as decision variables will be inside a vector (Decision vector), which is the one to modify to achieve the optimal trajectory. These values are bounded according to reasonable values for this kind of mission:

- Time of flight: between 300 and 1200 days.
- Initial impulse from the Earth: between 0 and a 30% of the one needed for a Hohmann trajectory.
- ΔV: from 0 to 1.
- $\alpha_{\Delta V}$: between $-\pi$ and π.

An important consideration is the fact that the calculation of the mass of fuel can be done immediately with the decision vector. The propagation is done to ensure that the state of the spacecraft in the end is similar to the one of Mars. This means that two additional variables have to be taken into consideration as constraints: the error in position and the one in velocity. Acceptable values are in the order of 10 km in position and 0.1 km/s of velocity.

3 Definition of the simplified problem

The main difficulty of this problem is studying the behavior of the function since the number of decision variables is large since there is a value of magnitude and angle for each impulse. Therefore, visualization of the problem becomes extremely hard. In order to solve this problem, the decision vector is reduced to a two-dimensional vector formed by the time of flight and the initial velocity from the Earth.

This is done by fixing the value of the impulses and its direction. Therefore, the magnitude is fixed to 0.2 and the direction is that of the velocity vector at that point.

Now, it is important to notice that the study of the problem becomes simpler, finding an optimum is faster since there are less possible options, but the optimum will normally be worse than in the actual case as the problem is more constrained.

The idea of this simplification is allowing to study the problem and different techniques in a simpler way. Also, allowing for easier visualizations.

3.1 Initial investigation of the simplified problem

In order to get a better understanding of the behavior of the problem, a parameter sweep has been performed over the values of the two decision variables between their bounding values.

First of all, the value of the errors in position and velocity have been plotted for each variable. The variation with respect to time has been done for one value of velocity and vice versa. The following figure shows that the errors is a convex function when seen as a function of the velocity (for the given range of velocities), but it is completely non-linear when looking at the time. There is a cyclic behavior that is given by the positions of the planets, but this function is non-monotonic and non-convex. In any case, the function is also non-monotonic.

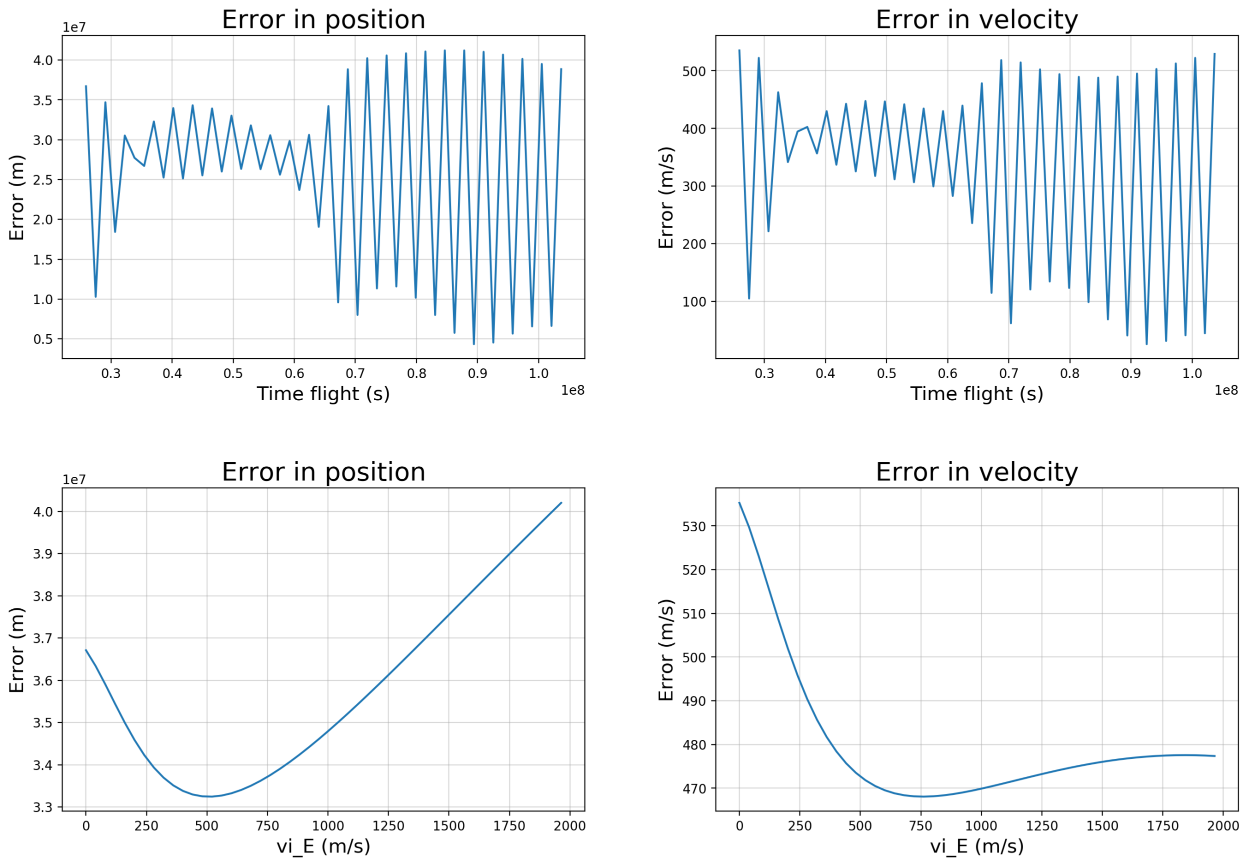


Figure 2: Variation of the errors in position and velocity with respect to the two decision variables.

For a more clear view, the contour plots for those errors are shown. Here, it is seen that there is not a clear pattern that can be distinguished. The conclusion that can be drawn is that the problem is extremely non-linear.

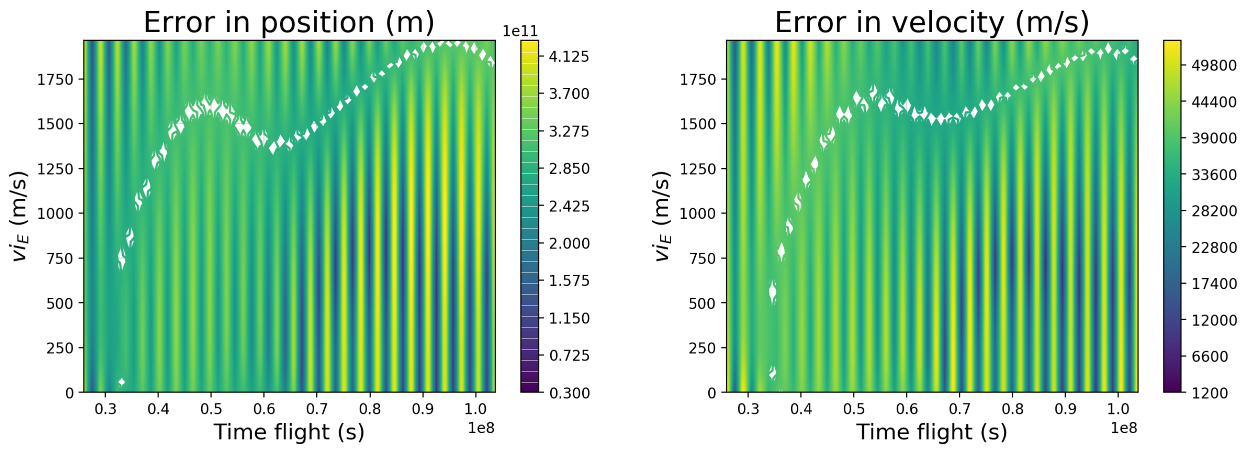


Figure 3: Variation of the errors in position and velocity with respect to the two decision variables for the values obtained in the parameter sweep.

Of course, the mass of fuel varies linearly with both factors, which means that it is monotonic. Because of that, the function can also be considered convex. This makes sense when thinking that the larger the initial impulse (vi_E), the more fuel is required to perform that maneuver. Analogously, since this model simulates constant impulse by multiplying thrust by time of flight, the more time, the more fuel that will be consumed.

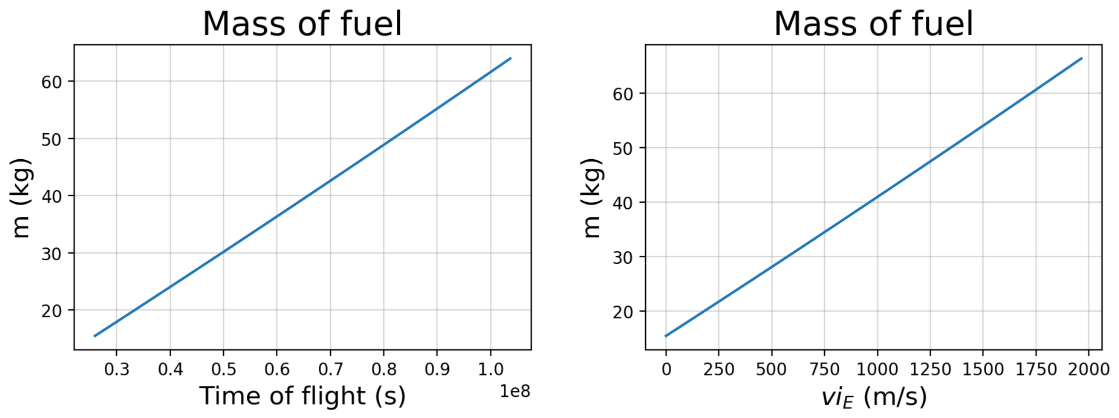


Figure 4: Variation of the mass of fuel with respect to the two decision variables.

With those results, is interesting to think of the bounds of those functions. First of all, the mass of fuel will have a value between 0 (when no impulses are applied) and a value corresponding to the case where the maximum impulse possible is applied at every point and the time of flight is maximum: $T_{max} \cdot t_{flight,max}$. Therefore, it can be stated that this value is bounded. Since the calculation of the errors passes through a complex function that can be thought as a black box, the only information that can be obtained is that their minimum value is 0, as they represent the absolute difference between two position or velocity vectors and an extremely large value that is unknown, but bounded due to the limited magnitude of the impulse that can be applied. This means that the spacecraft cannot get infinity far with the bounded set of impulses.

Although a sensitivity analysis can be interesting, in this case it makes sense to perform it o a point that is closer to the optimum than just to a random decision vector.

In order to see how much the trajectory can vary with the different values of the input parameters, the trajectories for the case in which the decision vector is set to the lowest values allowed and the case in which the decision vector is set to the largest values allowed are plotted in Appendix 1.

3.2 Optimization on the simplified problem

Objective function

The goal is then minimize the mass of fuel while obtaining a final error that is acceptable. This last condition is the hardest to achieve, since normally minimizing one of those implies increasing the value of the other, and minimizing both tends to increase the mass of fuel. The problem is then a constrained optimization with an objective function and two constraints. Because of the difficulty to achieve those constraints, another way to view the problem is as a multiobjective optimization. The approach to be used to avoid this multiobjective optimization consists on including the errors as penalization functions in the objective function. Therefore, it is assumed that instead of achieving those limits, which has been found almost impossible at this level, we want to obtain the best trade-off between those three quantities.

Another problem that arises when adding those functions as penalizations is that that they are all in different scales. For example, the error in position can be in the order of 10^{11}, whereas the error in velocity can initially be of 10^4 and the mass is normally lower than 10^3 kg. Normalization is therefore necessary:

$$f = \frac{m_{fuel}}{m_{dry}} + \frac{E_p - 1 \times 10^4}{1 \times 10^4} + \frac{E_v - 1 \times 10^2}{1 \times 10^2}$$

The normalization is based on he idea that, when getting close to the goals (10 km in position and 0.1 km/s in velocity), those values will be in the same order of magnitude of the mass, or that will become the most relevant factor in the function.

Of course, other normalization or weights have been tried, but this one was proven to be the one that performs better in achieving the specific goal of this problem.

Choice of the optimization method

From the previous sections, it can be noticed that the problem being dealt with is highly non-linear, which implies the possibility of multiple local minimums. This means that the method to be used has to be able to find the global minimum.

Another important consideration is the fact that the calculation of the trajectory is a function that is relatively computationally expensive. Although this cannot be noticed when performing one evaluation, for an optimization algorithm this is an important consideration.

This means that the algorithm has to be adequate for unconstrained problems, to find global minimums and to avoid incurring in a large computation time. Also, although in this simplified case it is not that relevant, the large number of decision variables in the actual problem are also an important factor when choosing the optimization algorithm. The same methods will be used for the simplified and actual cases since there is no point in studying different techniques, and using the simplified problem as an initial study is far more interesting.

First of all, since the objective function is built using a complex function, that can be thought as a black box. Therefore, the optimization method has to be such that is based on the inputs and the value of the function. Therefore, random methods are first considered. A random walk is interesting for comparison, although it can already be noticed that, with the large range of possibilities available, it is really hard that it finds the global minimum. This is the reason why an evolutionary algorithm has been programmed for this problem.

Other methods have been tried, such as the steepest descent direction. However, in order to calculate the magnitude of the step to be taken, it is necessary to perform a minimization of the function. Therefore, instead of this approach, a version of the cyclic coordinate search has also been implemented.

Programming of the optimization methods

Random walk: This algorithm is based on the generation of a number of n random decision vectors taking into account the bounds for those variables. Then, the solution is arranged so that the one that produces a lower value of the objective function is at the top of the matrix. Therefore, the best is returned.

Evolutionary algorithm: The algorithm is based on the creation of an initial population constituted by a certain number of individuals (decision vectors) that are within the allowed bounds [5]. The fitness of each individual is evaluated by using it as the input to the objective function. The matrix of individuals is then arranged so that the best ones are located on the first rows.

Now, if elitism is introduced, a certain percentage of the total population is preserved so that the best solutions pass to the next population generation (children) without any modifications. Then, crossover is executed by performing a cut at a random location of the individual and exchanging one of the parts with a different individual. Then, if mutation is introduced, each bit of the individual is subjected to mutation (random change of value) if a random number is lower than the input value of mutation.

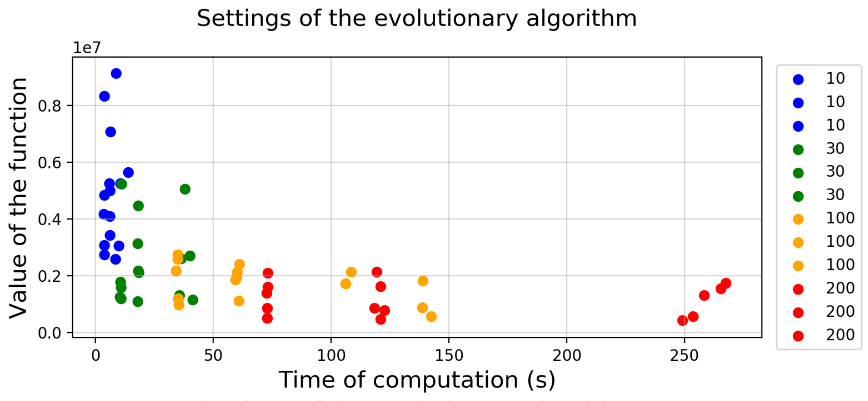
The next population is evaluated and the procedure is repeated. If the minimum is not improved in the next iteration, a counter for the unsuccessful number of consequent iterations is increased. Therefore, a stop criteria is that the number of iterations without success does not reach a certain value. Another criteria is the maximum number of generations, or iterations. The algorithm keeps the best optimum and returns it.

It is interesting to mention that an evolutionary algorithm has been used instead of a genetic algorithm in order to allow for every possible value, which is not achieved with a genetic algorithm as its population is based on binary bits instead of actual numbers.

Since this algorithm is based on a random initialization of the population, the results are also subjected to the initial values. Different runs will most likely yield different optimums. The larger the number of individuals, the more possibilities are available and a better global minimum is more likely to be found.

With this algorithm, it is not possible to ensure that the global optimum has been achieved. However, since the goal is performing a preliminary optimization of the trajectory, obtaining a minimum that is good enough for an initial evaluation is considered enough.

In order to know which settings for the algorithm yield the best results, an evaluation has been done by performing a parameter sweep over the number of individuals and the maximum number of iterations, also performing the optimization 5 times for the same settings to study the variation with the initial conditions. The results of this study can be seen in the following figure:



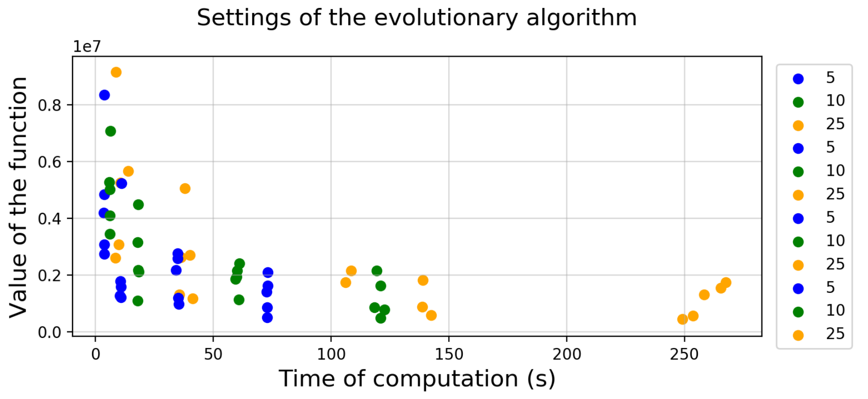


Figure 5: Representation of the results obtained for different values of individuals and maximum number of iterations. Every run has been repeated 5 times to deal with random results. In the first plot, the colors represent the number of individuals whereas in the second one they represent the maximum number of iterations.

It can be seen that the larger the number of individuals, the less disperse the points of the same inputs are. This is due to the fact that more possibilities are available and achieving the global minimum is more likely.

Since this simplification is chosen as an initial approach to the actual problem, the time of computation is more relevant than the actual result. As a result of this study, the number of individuals chosen is equal to 100 and the number of iterations to 10. The other settings incur in either too large computation time or large values of the function.

A visualization of the orbit achieved is shown for the Evolutionary algorithm. Since the goal is approaching Mars (the red point) with the last triangle (each triangle is an impulse applied), it can be seen that the results are much better than for the initial orbit shown in Figure 9.

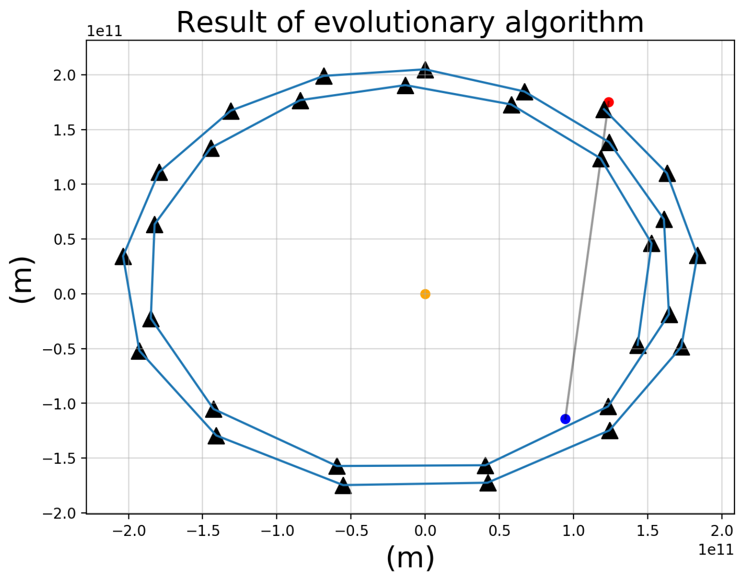


Figure 6: Trajectory flown from the Earth to Mars with the results of the Evolutionary Algorithm as inputs.

The orbits achieved for the other algorithms are shown in Appendix 1.

Coordinate search: An algorithm based on the cyclic coordinate search has been programmed. The idea is, from an initial decision vector, vary each of the values by adding or subtracting some step so that the minimum is improved. The function is evaluated for the nominal value, the one for $x + h$ and the one for $x - h$. The program keeps the vector with the best minimum. After all the values have been modified, if the vector has varied because an improvement has been achieved, the counter with unsuccessful consecutive iterations is set to zero. The step is then modified by a certain percentage, which has been set to 70% of the previous value (chosen by experimentation and trial and error). Therefore, each step becomes smaller with each iteration.

This algorithm is an intensive search of the space around a point that is highly determined by the values of step chosen. Those have to be different for the different variables of the decision vector as their scales of variation are extremely different.

This algorithm can only achieve the finding of a local minimum and has the risk of getting stuck if the choice of step is not the adequate for the specific point.

Based on the study of the sensitivities that will be later explained, the step was chosen to be:

- $h_t = 5 \times 10^3$ s
- $h_{vi} = 100$ m/s
- $h_{DeltaV} = 0.05$
- $h_t = \pi/2$ rad

Monotonic Basin Hopping: This algorithm has been used for comparison since it is already implemented in `Python` in contrast to the previous ones. This algorithm performs a local search around a point until it reaches a local minimum. Then, a jump is performed to another initial point to be able to find global minimums. This algorithm can include the introduction of the bounds of the variables if the choice of the local optimizer is adequate. That is the reason why the `COBYLA`

(Constrained Optimization By Linear Approximation) algorithm has been chosen in contrast to the other options (among which Nelder-Melder method is found).

An interesting finding regarding this method is that it tends to return solutions that are outside the bounds set for the functions of the decision variables. The reason for that lies on the fact that it can sometimes be more efficient to jump from points outside the feasible region to obtain the minimum. However, a limitation of this algorithm consists on the fact that, instead of saving the best value that is within the valid region and allow for jumping from points outside it, only saves a single point for both. It is therefore important to make sure that the given optimum is based on valid values.

Evolutionary Algorithm + Coordinate Search: Another interesting approach is the one corresponding to the Evolutionary algorithm plus the coordinate search. The idea behind is to search for a global minimum using the EA and, when it is found, a local search is performed around it to keep improving the result. This is done by using the optimum value of the EA as an initial value for the coordinate search. The improvement is noticeable and the results show that this approach is extremely interesting.

In order to visualize the improvement of the optimum, the following plot shows the best solution found for each iteration of the genetic algorithm (with circles), and the best one found for each iteration of the consequent coordinate search (with crosses). Encircled are the points that are the initial value of each of the algorithms. The color represents the value of the mass of fuel consumed, so that the three constituents of the objective function are visualized.

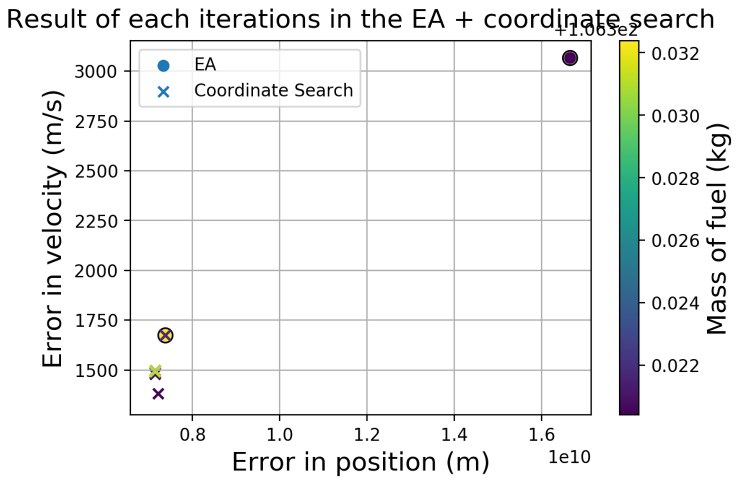


Figure 7: Representation of the three values that form the objective function for each iteration of the EA and the consequent coordinate search. Encircled are the initial values for each of those algorithms.

Comparison of algorithms and final approach

In order to find what the achieved result is with the previous methods, a comparison has been done trying to adjust the settings to similar situations. The results are seen in the following figure:

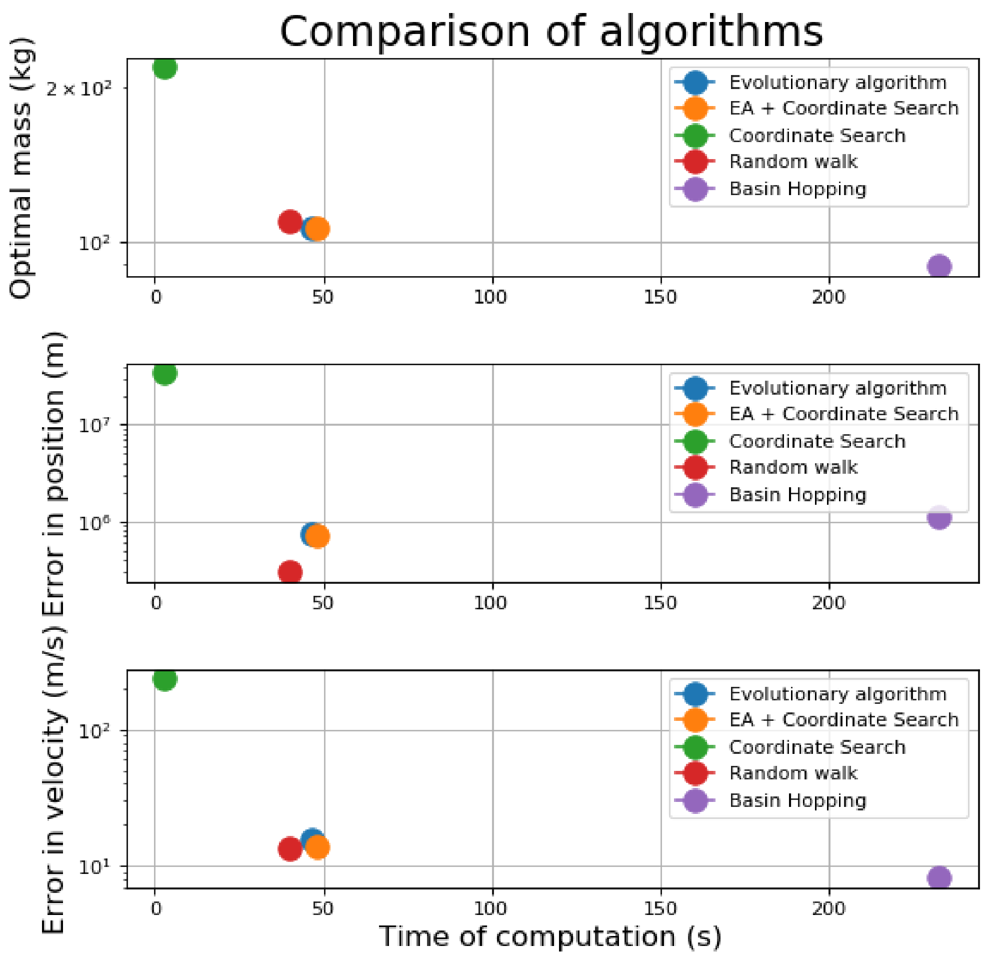


Figure 8: Comparison of the results for the five algorithms chosen against the computation time.

Each point represents the results of each algorithm. Although some of them depend on ”luck” as they use random procedures, others like the coordinate search are highly dependant on the initial point given, which has been set to an initial value within the bounds.

The plot shows that the evolutionary algorithm (EA) and the Basin Hopping are the ones with the largest computation times, which is due to the large number of function evaluations to be performed. The random walk can achieve better values than any other function, but that is not normally the case and it is not a robust method for this kind of problems. The case in which the coordinate case is added to the EA, shows little to no improvement with respect to the EA. However, this is not always the case. In other runs that have been executed, a different minimum is achieved by the EA, and in some cases the coordinate search can largely reduce the value of the function. This means that, depending on how close the result from the EA is from the local minimum, the coordinate search is more or less relevant.

In this case, the Basin Hopping has a much larger computation time than the others. Depending on the run, this can largely vary.

The coordinate search has been included in the plot although the results are much worse to the others. The reason for that is that it is only achieving a local optimum in the region of the initial value. This is easily seen in the trajectories generated by the initial point and the coordinate search:

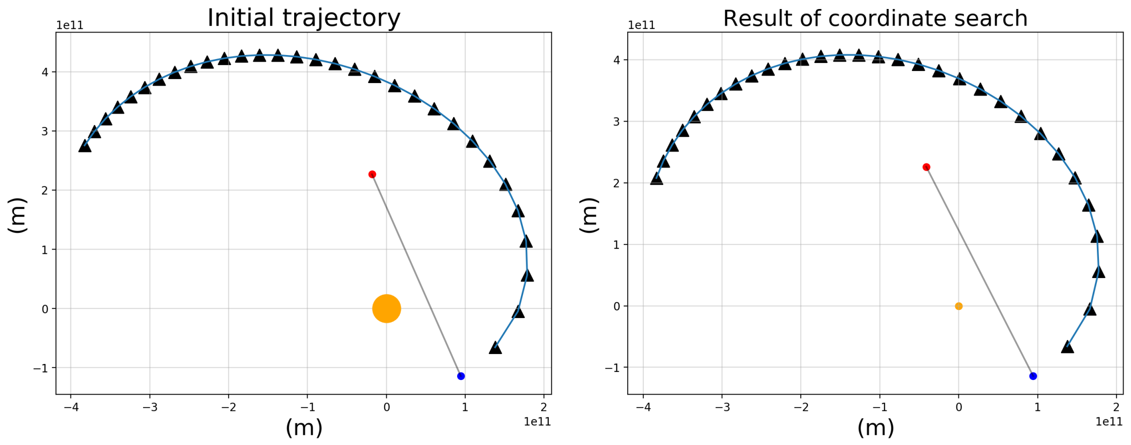


Figure 9: Comparison of the trajectory for the initial decision vector and the trajectory obtained after performing the coordinate search with the initial decision vector as starting point.

It is obvious that both trajectories look extremely alike, although the coordinate search improves slightly the result, specially compared to the other orbits achieved.

A table with the best values obtained after one run of each of the previous algorithms is shown:

Table 1: Comparison of results for the different optimization methods.

Method	Computing Time (s)	Mass of fuel (kg)	Error in position (m)	Error in velocity (m/s)
EA	46.477486	106	7386816504.4	1673.3
EA+ coord	48.416861	106	7153658299.9	1491.6
Coordinate Search	2.893534	217	342583916494.6	23532.1
Random Walk	39.883440	109	3002054206.6	1447.3
Basin Hopping	232.507398	89	10975149917.6	926.2

As stated before, for the Monotonic Basin Hopping it is necessary to check if the values obtained are within the established bounds. A quick examination shows that many of the decision variables are outside the bounds, which makes the solution unfeasible.

Therefore, the EA + Coordinate Search has been chosen to perform the optimization.

3.3 Investigation of the obtained optimum

Starting from the solution of the Evolutionary algorithm plus the Coordinate Search around the solution point, different studies can be performed to understand the behavior of the function around that point. The study of the sensitivities is important to get an idea of how the space behaves around the minimum. Since the decision vector in this case is just formed by two variables, it is easy to represent the derivatives with respect to those two variables:

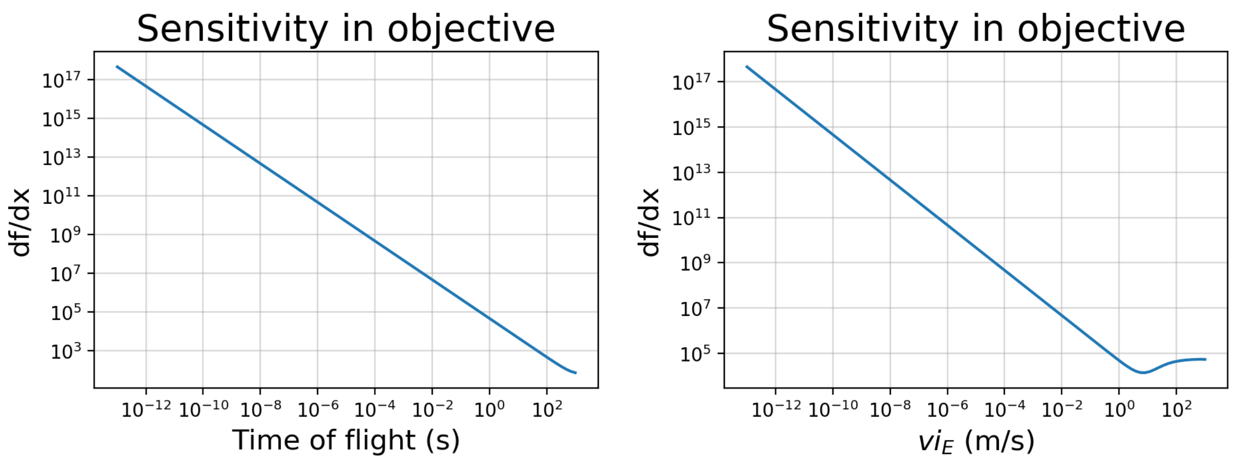


Figure 10: Study of the sensitivity of the function to changes in the decision variables.

In order o calculate the sensitivities, the finite difference approach has been chosen for its simplicity. Therefore, the derivatives are calculated using the central derivative according to the equation:

$$\frac{\partial f}{\partial x_i} = \frac{f(x+h) - 2 \cdot f(x) + f(x-h)}{dh}$$

h is a vector of steps that is used to see at which values the function varies more rapidly.

The sensitivity seems to vary linearly reaching a minimum for large steps. The initial decrease in the sensitivity can be due to round-off errors. However, those are normally presented in the shape of oscillations.

Although no formal study has been done to prove it, it seems logical to assume that the derivatives would be different for a different point due to the extremely large non-linearity of the problem.

A study of the Karush-Kuhn-Tucker (KKT) conditions would be interesting if the error in position and velocity had been treated as constraints instead of penalization functions. However, since no point has been found that achieves what was initially assumed to be reasonable results for those errors, using them as constraints would imply not finding a point in the feasible region.

However, it is indeed interesting to know which constrains are being active for the optimum. In this case, the bounds of each decision variable are assumed to be the only constrains. First of all, it is necessary to study whether the minimum is in the feasible region. Therefore, each value is compared to its limits individually. It is found that the minimum is indeed in the feasible region. In order to know if the constraints are active, the value of the absolute difference of the variable minus the bound has to be lower than a 0.5%, value that has been chosen arbitrarily. In this case, none of the constraints are active.

3.4 Results

After performing a comparison of algorithms, and a comparison of settings for the Evolutionary Algorithm, the decision made is to use a combination of a global algorithm (EA) and a local optimizer (Coordinate Search). Although this procedure is computationally expensive (as it can be seen in Figure 8) due to the large number of function evaluations and the complexity of the function to evaluate, the algorithm achieves a value that improves the initial one significantly.

Now, in order to perform a full optimization with the settings chosen for the EA and the step sizes of the coordinate search based on the sensitivity of the optimum, the procedure is repeated three times to allow to choose the best one without depending too much on "luck":

Table 2: Comparison of results for the different optimization methods.

Method	Time of computation (s)	Mass of fuel (kg)	Error in position (m)	Error in velocity (m/s)
EA:	40.370790	104	16970736715.8	3014.7
EA + coord:	2.665267	104	15731936196.5	2173.0
EA:	45.769189	104	16970736715.8	3014.7
EA + coord:	2.588097	89	21910085058.4	940.6
EA:	40.996075	104	16970736715.8	3014.7
EA + coord:	4.503107	102	8263459846.4	996.9

It can be observed for this values that the results for the position error specially are not even close to the goal stated in the beginning. The reason for this is that, by fixing the magnitude and direction of the impulses, the problem is more constrained. Therefore, the following step is eliminating that restriction to study if better results can be achieved. Although the point achieved cannot be assumed to be the global optimum, it serves as a reference to locate a feasible trajectory. Also, it is a good exercise to understand the behavior of the problem and the influence of the different input values.

4 Definition of the actual problem

Now that the analysis of the simplified case has been performed, the restriction over the impulses applied is eliminated and the magnitude and angle of each of those are included in the decision vector to be part of the optimization.

It was previously mentioned that, by setting the impulse properties as constants, the problem is more constrained. A way to easily see that is by comparing the plots for the extreme cases, which means those in which the decision vector is set to the minimum and maximum values allowed by the bounds. The difference is specially noticed in the trajectory for the maximum impulses, since in the previous case the magnitude was set to a relatively low value:

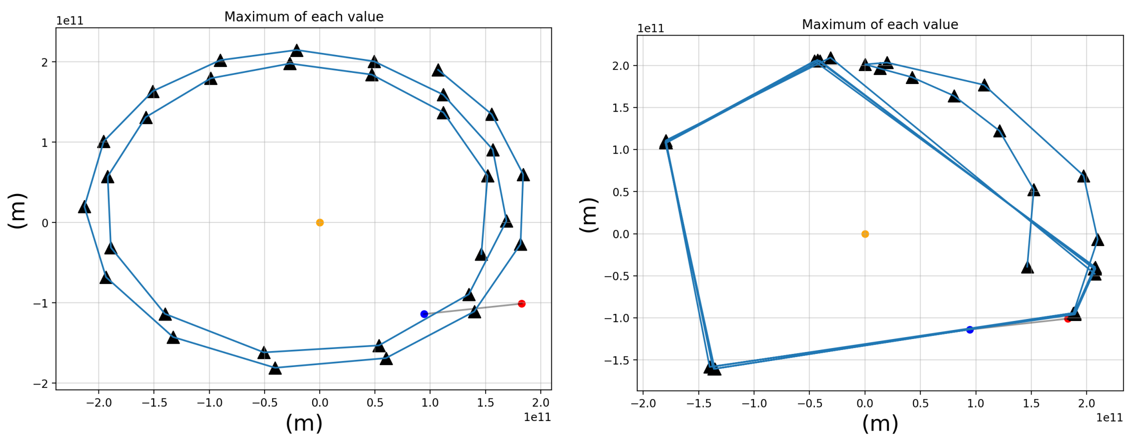


Figure 11: Trajectory flown from the Earth to Mars with the results minimum and maximum allowed values for the decision vector as inputs.

It can be seen that the second figure is completely chaotic and the orbit does not make sense. However, this is due to the angles of the impulses, and the conclusion that the trajectory is less constrained can be drawn.

4.1 Initial investigation of the actual problem

Now, as in the previous case, it is interesting to investigate the behavior of the problem and the variation with respect to different values. In this case, the method used consists on a parameter sweep of the time of flight and initial velocity, for different cases of the magnitude of ΔV. Therefore, the variable of the angle is fixed and the magnitude for all the impulses is the same. This implies that, in order to make this study, many simplifications are being made. This also means that the problem is much more complex than what is seen. However, the goal is studying the variation with different magnitudes of the impulse.

First of all, the dependency of the errors with time and initial velocity are shown. The oscillatory behavior that was seen in the simple case is also seen in these cases. It can be observed that the larger the impulse applied, the larger the amplitude of the oscillations although the period seems to be similar, most likely due to the dependency on the positions of the planets.

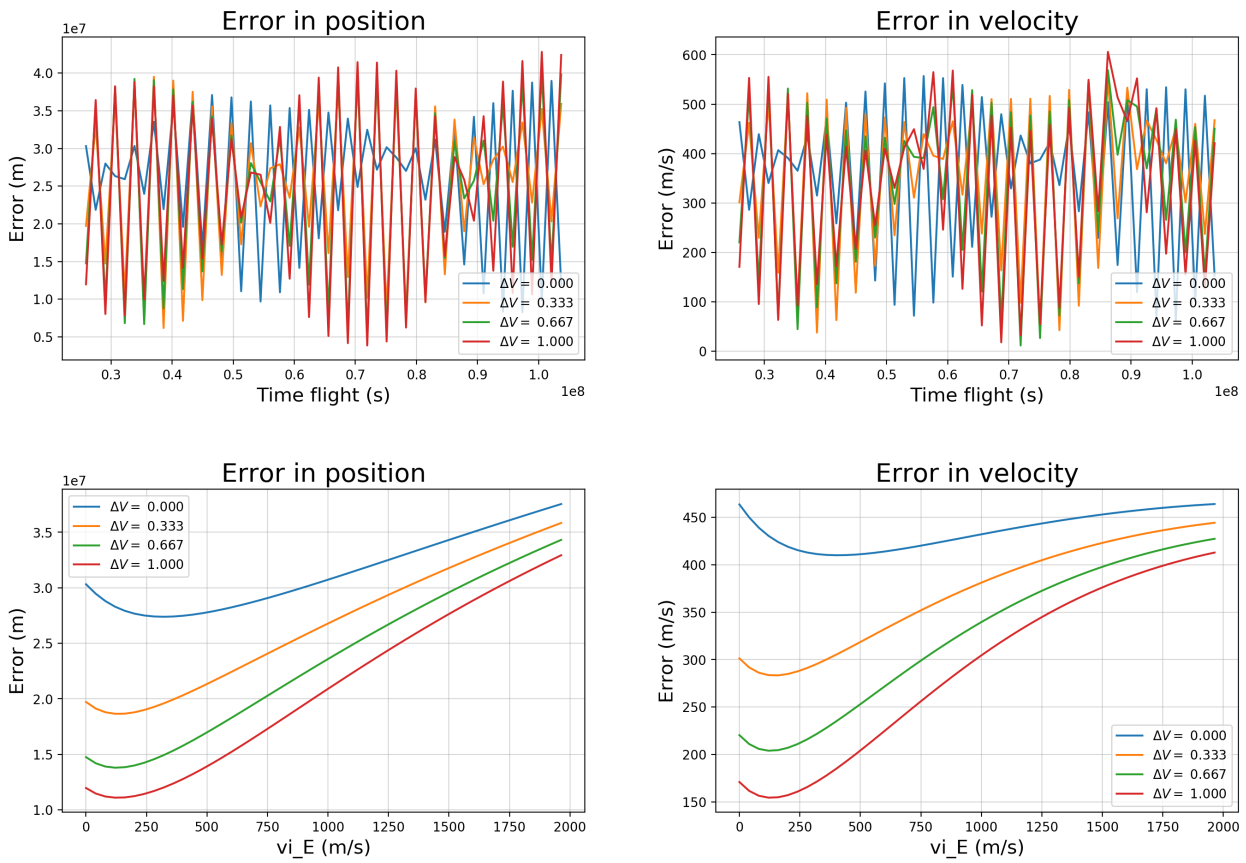


Figure 12: Variation of the errors in position and velocity with respect to the two first decision variables for different magnitudes of the impulse.

For the initial velocity, it is seen that the larger the impulse, the lower the error. This is definitely dependent on the case, and the position of the minimum cannot be generalized, although it can be seen that it increases with velocity. It is therefore interesting to assume that the initial velocity will tend to lower values. However, the study of the variation of mass has not been done and the objective function depends on the three values.

Therefore, the conclusions obtained are that, the function is non-monotonic, as for both cases it presents at least a descendent and ascendant part. Also, the errors with respect to the initial velocity can be considered convex. However, although visualization is not possible, the shape of this function varies when different magnitudes and directions are applied for the impulses. Therefore, it is most likely that this function is also not convex. The conclusion is then that the studied function is non-linear, non-monotonic and non-convex.

4.2 Optimization of the actual problem

Optimization approach

In this case, the formulation of the optimization problem is similar to the simplified case. The objective function remains the same but the decision vector will be formed by a much larger number of parameters since the magnitude and angle of the velocities are left as free variables. In the introduction of the problem, the bounds the problem is subjected to were explained, and those remain the same.

Now, for the choice of the optimization method, the same comparison between algorithms has been done in order to compare whether the same approach is adequate. The difference now will lie on the fact that the time of computation is much larger due to the increased number of variables.

Now, methods like Nelder-Mead or the steepest descent direction become inefficient and hard to implement because of the large number of variables involved. In any case, the first algorithm does not allow for the introduction of constrains, so it would not be possible to include the bounds of the variables.

Comparison of algorithms and final approach

Starting with the Evolutionary Algorithm, a parameter sweep has been implemented with the number of individuals and the maximum number of iterations in order to choose the best settings for the algorithm (Figure 13).

In this case, the values in the central area have been considered since the result is more important than the difference in computation time. Therefore, 100 iterations and 25 individuals have been chosen.

It is interesting to perform a quick comparison of this results with the ones for the simplified case (Figure 5). It can be seen that, in general, the values of the function are larger, except for large numbers of individuals and iterations. However, the times of computation are extremely similar, contrary to what was intuitive when increasing the number of decision variables. The points are not as concentrated in vertical lines but more dispersed, accounting for the larger set of possibilities.

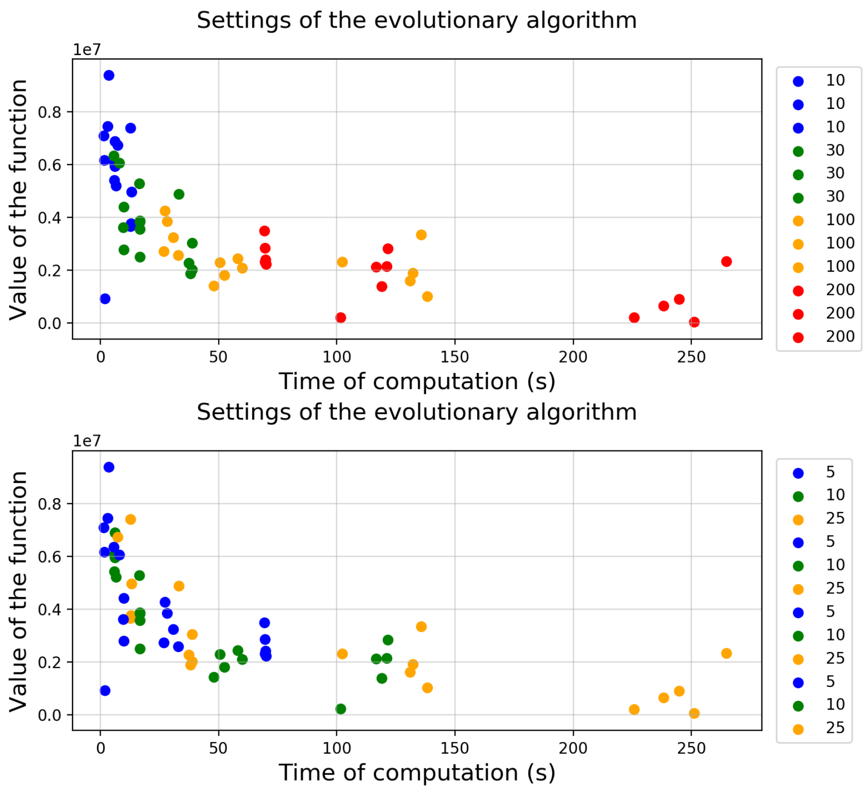


Figure 13: Representation of the results obtained for different values of individuals and maximum number of iterations. Every run has been repeated 5 times to deal with random results. In the first plot, the colors represent the number of individuals whereas in the second one they represent the maximum number of iterations.

Since the visualization of the trajectories does not provide further information, those plots have been included in the Appendix. Nevertheless, it is interesting to see the evolution of the minimum achieved by the EA with the increase of the iteration. Also, for the following figure, the evolution of the coordinate search is shown:

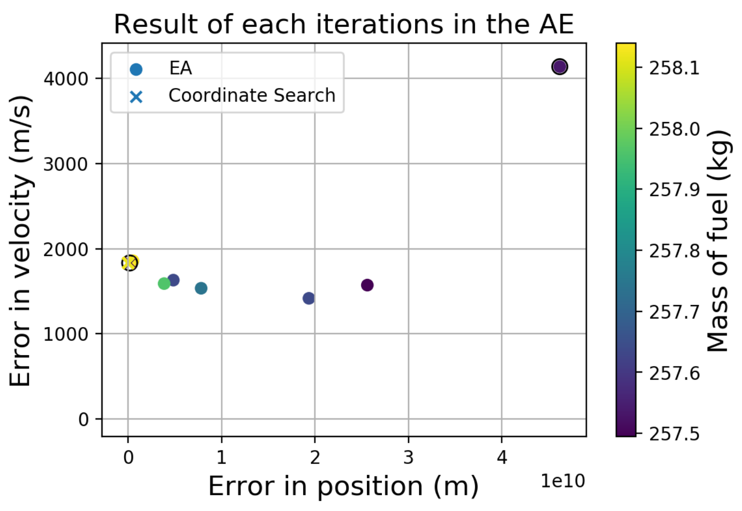


Figure 14: Representation of the three values that form the objective function for each iteration of the EA and the consequent coordinate search.

As observed, the coordinate search is in this case not providing a large improvement over the optimum. It is interesting to see, even with the low number of points, that a certain distribution can be observed, maybe showing what would be a Pareto front.

Although the previous plot does not give much useful information because of the low number of points, for a different run of the evolutionary algorithm a more interesting plot is obtained. For this case, a Pareto front can be clearly observed both for the EA and the following Coordinate Search. This implies that, in order to improve any of the constituents of the objective function, any of the others has to get worse.

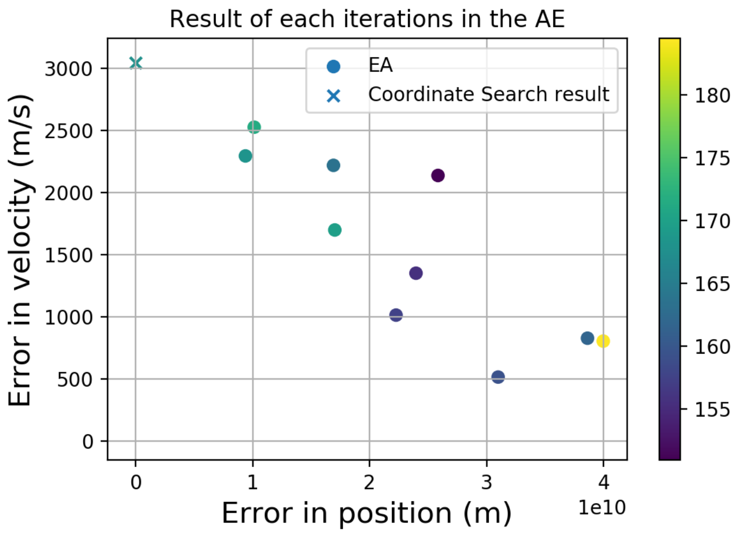


Figure 15: Representation of the three values that form the objective function for each iteration of the EA and the consequent coordinate search.

Also, it is interesting to see how the algorithm evolves with the iterations. Below, there is a plot that represents the values of the errors in position and velocity and the mass for the best individual of each iteration performed. Since the initial point is the one on the upper right corner, it can be observed how all the values are reduced over the iterations except for the mass, which increases. This is due to the fact that the errors are orders of magnitude larger, so reducing those contributes more to reducing the objective function. Here is where the design of the optimization function is involved, since different weights would possibly imply a different evolution of the algorithm:

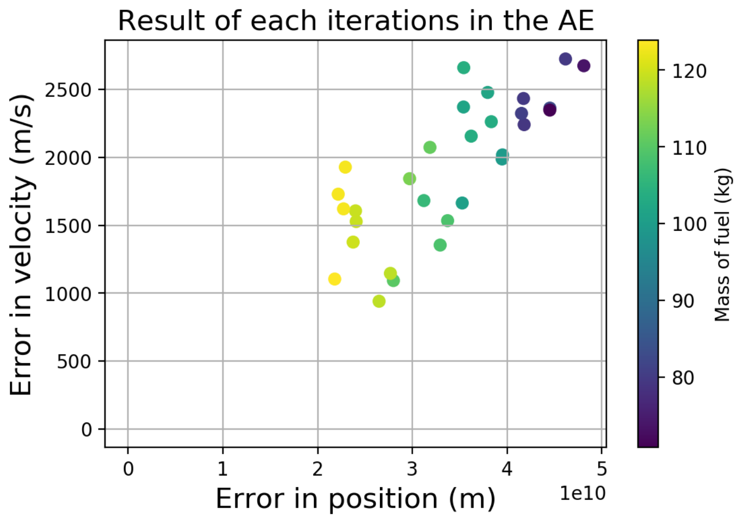


Figure 16: Representation of the three values that form the objective function for each iteration of the EA when the number of iterations has been set to a large value.

Now, the results for the different algorithms can be compared. In this comparison, the Monotonic Basin Hopping results have been included. However, when studying those results, it can be observed that the decision vector exceeds the bounds imposed in many of the variables. After trying different runs since there is a random factor involved, none of the results obtained were inside the feasible region.

In the following figure, it can be observed that the Coordinate Search performs poorly as expected. The random walk is better than the EA in many cases, but incurs in a position error that is orders of magnitude larger. The improvement of the coordinate search when added after the EA can be observed, and also seen in the table with the results. Also, the computation time is much larger, but this time is worthy as otherwise a reasonable result is not reached.

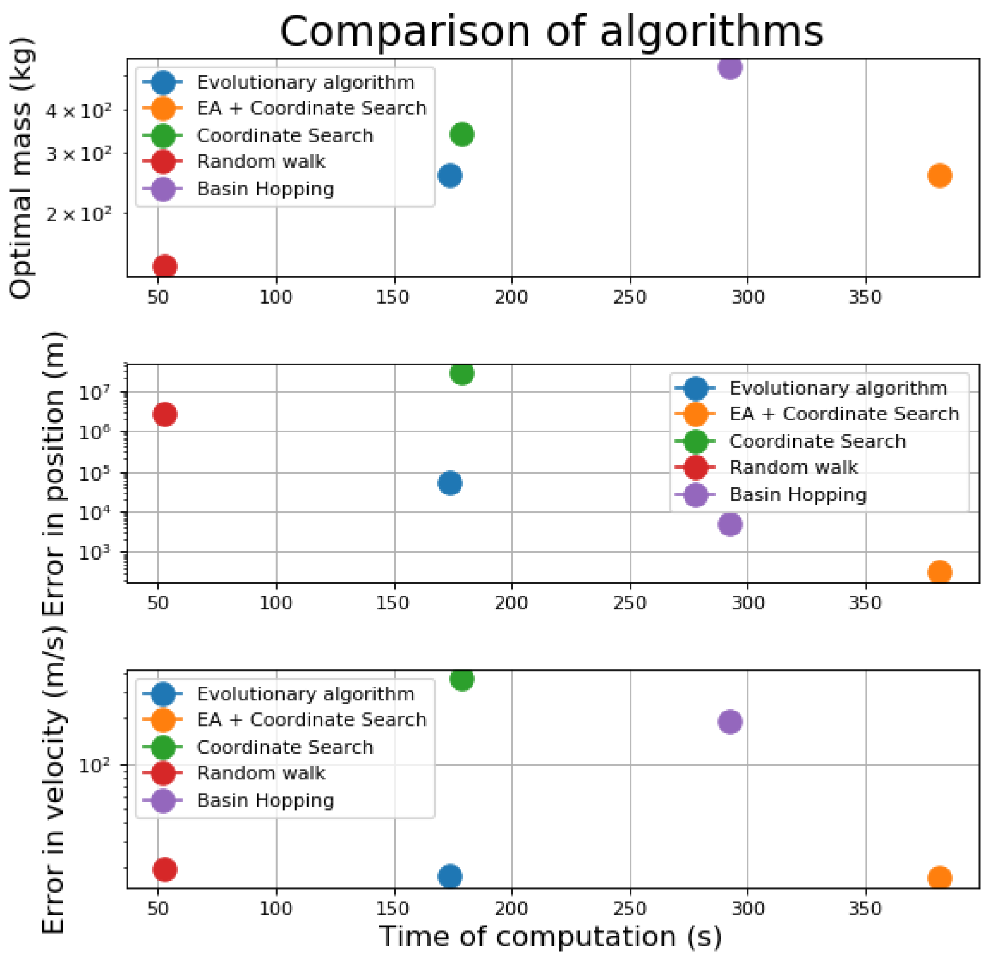


Figure 17: Comparison of the results for the five algorithms chosen against the computation time.

Method	Computation time(s)	Mass of fuel (kg)	Error in position (m)	Error in velocity (m/s)
EA	173.920502	256	509987378.2	1857.6
EA+coord	381.405212	258	3251164.6	1828.5
Coordinate Search	178.763217	339	271449554746.7	36856.1
Random Walk	52.695481	140	26917792751.6	2052.5
Basin Hopping	292.281129	528	51859124.2	19473.8

As a conclusion from this comparison, the EA + coordinate search approach has been chosen to solve the problem.

4.3 Investigation of the obtained optimum

First of all, it is interesting to know how sensitive the obtained minimum is to a perturbation. Therefore, the sensitivities have been calculated using the central derivative:

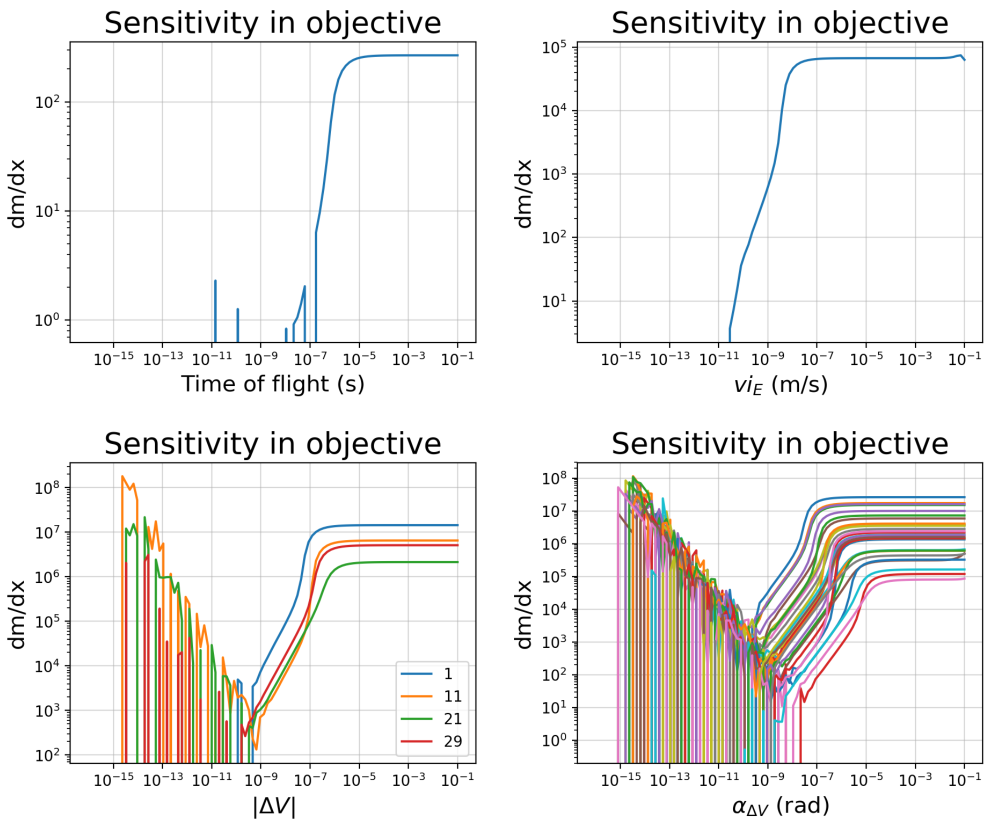


Figure 18: Study of the sensitivity of the function to changes in the decision variables. In the lower left plot, only some impulses are plotted to allow for a better visualization.

The conclusion that can be drawn from this plot is that, first of all, the oscillations that occur before the minimum is reached can be due to round-off errors (errors due to the limited precision of the computer). From that point, the derivative increases linearly with the step size, which seems more reasonable that the oscillatory behavior. It can be observed that, for large step sizes, the variation of the function becomes really large, specially in the case of the initial velocity.

In the plot for the sensitivities, the ones corresponding to the magnitude of the impulse have been plotted just for some impulses, which order can be seen in the legend. The purpose is to see more clearly the common pattern that all of them experiment, although there is an offset between them. For the angles, all the impulses have been plotted, and it is interesting to observe that also all of them follow a similar pattern with an offset in both the x and y axis. This offset is due to the fact that depending on the position of the impulse, it will be more or less determinant in the calculation of the final error, indicating that not all the impulses are as relevant.

The study of the sensitivities is extremely important when choosing the step size for the Coordinate Search algorithms implemented since that step size is a determinant factor in improving the solution. A simplification included is that all the impulses vary with the same step, although it is seen that the variation of the function will be different depending on which impulse is varied.

Again, it is interesting to know which constraints are being active for the optimum. The assumptions are the same as for the simple case. First of all, it is found that the minimum is indeed in the feasible region. In this case, two constraints are active, corresponding to the impulses applied in 20th and 21st place. Those impulses have a value close to 1, which means that increasing the bound for them may improve the minimum obtained.

4.4 Results

With the previous studies performed, a final result can be achieved. Using the best parameters found for the evolutionary algorithm and the information of the sensitivities for the Coordinate Search step size, the optimization process is repeated three times in order to find an optimum and take into consideration the "luck" factor.

Table 3: Comparison of results for the different optimization methods.

Method	Time of computation (s)	Mass of fuel (kg)	Error in position (m)	Error in velocity (m/s)
EA:	122.189015	215	6608123949.8	2659.7
EA + coord:	181.081194	217	1727.6	2947.4
EA:	133.595429	210	3999588575.0	1686.5
EA + coord:	175.980228	216	747.0	1863.2
EA:	130.019040	212	15715857015.4	703.8
EA + coord:	175.370652	213	2617.6	1808.5

From the table it can be observed that the results are really dependent on the initialization of the algorithm. However, a more important conclusion that can be drawn is that, in order to decrease the error in position for example (done with the Coordinate Search), the error in velocity increases significantly.

The values of the decision vector are shown in a table in Appendix 2 as they do not provide any further information.

An interesting experiment is increasing the number of individuals to 250 and see if the results improve significantly for this case:

Table 4: Results obtained when the number of individuals is set to 250.

Method	Time of computation (s)	Mass of fuel (kg)	Error in position (m)	Error in velocity (m/s)
EA:	281.771990	229	9350286287.4	1892.4
EA + coord:	146.494628	236	7020.1	2416.7

As observed in the comparison of settings for the EA, some values increase and others decrease, but there is not a large difference in the results with respect to the chosen one, although the computation time is much larger.

5 Conclusions and recommendations

The challenge with this problem is the choice of the settings and algorithms in order to find the global minimum and achieve a feasible trajectory.

Due to the characteristic non-linearity of the problem, the choice of algorithms had to take that into consideration. The approach of global search (EA) plus a local refinement (Coordinate Search) is observed to perform really well for this problem although the minimum obtained cannot be ensured to be the global one.

For the simplified case, the results are far from what was initially required. The conclusion for that is that the fixation of the impulses does not allow for enough variation. In other words, the problem is overconstrained.

For the actual case, the results achieved with those two methods are surprisingly good for the limited computation time that was available, even if the implementation does not include parameters such as immigration (for the EA) and the change in step size for the Coordinate Search is basically chosen based on experimentation.

However, after performing different comparisons and studies, it is interesting to notice that there seems to be a threshold to the values of the errors in position and velocity that can be achieved for both the actual and the simplified problems. Two explanations can be possible: the problem is overconstrained, which is seen in the fact that some constraints are active. If this is the case, increasing the allowed maximum thrust would improve the optimum. The second possible explanation is that the objective function is limiting the success of the optimization. Although different weights and normalizations have been tested, the one used is the one that performed better. This however does not imply that there are not better options, and the study of the possibilities would be a recommendation for further studies.

Also, the choice of the settings of the algorithms is really important for the success of the optimization. Studying those settings in more detail would definitely improve, if not the final minimum, the efficiency of the methods.

In short, this problem has been a really interesting case to study different tools. However, the complexity of the calculation of the trajectory has been a limiting factor in the execution of optimization tests. Nevertheless, the final results are extremely promising and, although they do not achieve the required values of the error in position and velocity, the improvement shown with respect to the choice of a reasonable set of values is outstanding.

It is therefore beyond the scope of this project to perform detailed analysis of the different settings to achieve an actual feasible trajectory. In any case, this results can always be used as preliminary trajectories for more in depth projects.

References

[1] J. Sims and S. Flanagan, "Preliminary design of low-thrust interplanetary missions," 1999.

[2] T. D. Matthijs Langelaar, "Engineering optimization: Concepts and applications, lectures," 2019.

[3] C. H. Yam, D. D. Lorenzo, and D. Izzo, "Low-thrust trajectory design as a constrained global optimization problem," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 225, no. 11, pp. 1243–1251, 2011.

[4] Jet Propulsion Laboratory, "Dawn ion propulsion." `https://dawn.jpl.nasa.gov/mission/ion_prop.html`. [Online; accessed 2018-04-01].

[5] B. Craenen, A. Eiben, and E. Marchiori, "How to handle constraints with evolutionary algorithms," *Practical Handbook Of Genetic Algorithms: Applications*, pp. 341–361, 2001.

6 Appendix 1: trajectory plots

6.1 Trajectory results for the simplified problem

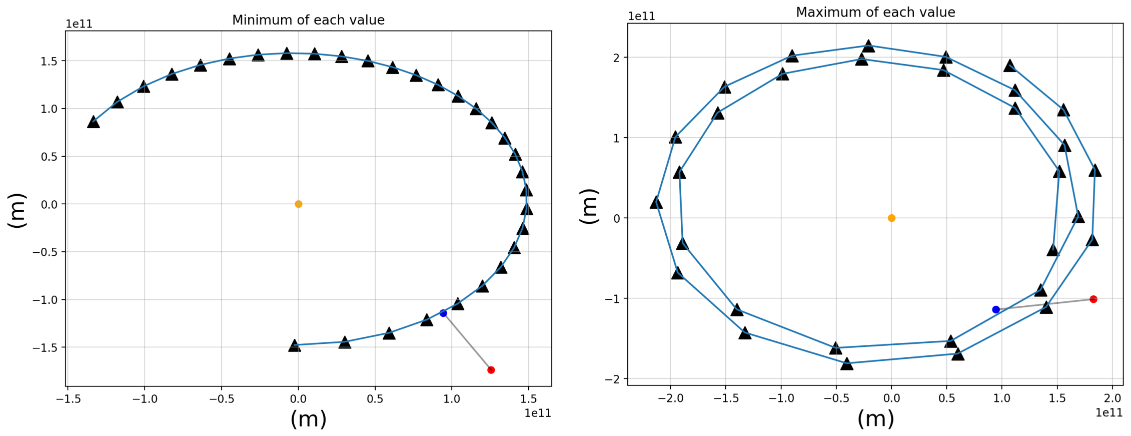
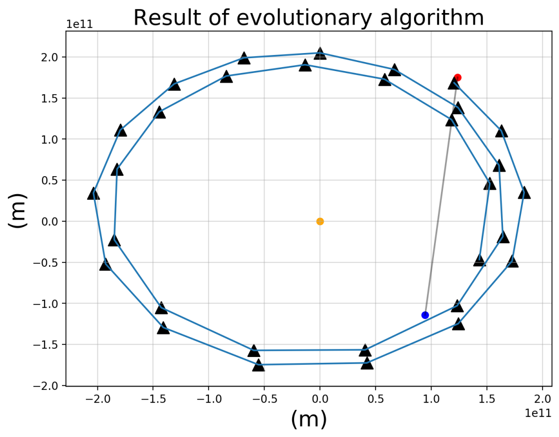
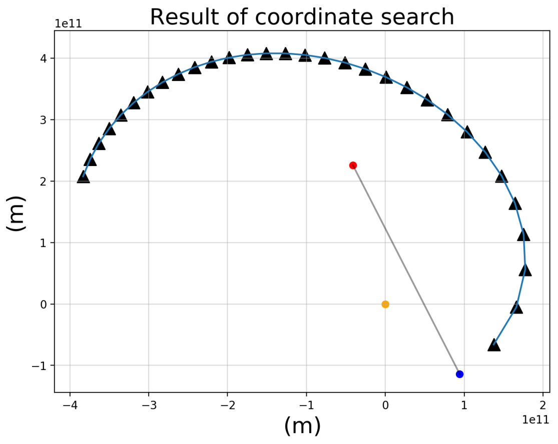
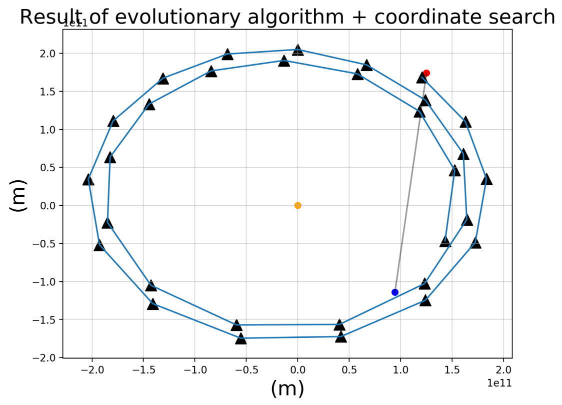
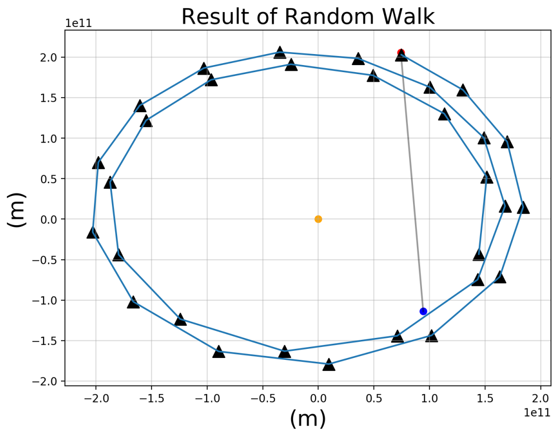


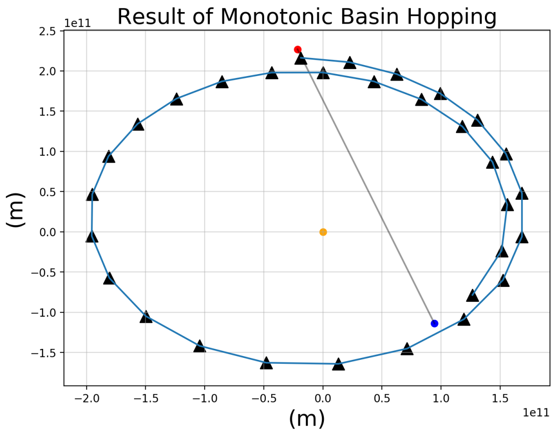
Figure 19: Trajectory flown from the Earth to Mars with the results minimum and maximum allowed values for the decision vector as inputs.



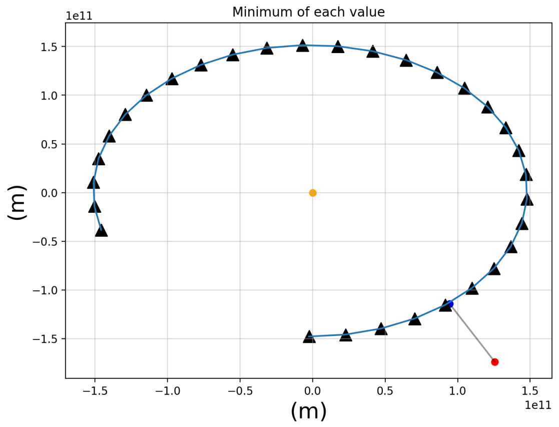








6.2 Trajectory results for the actual problem



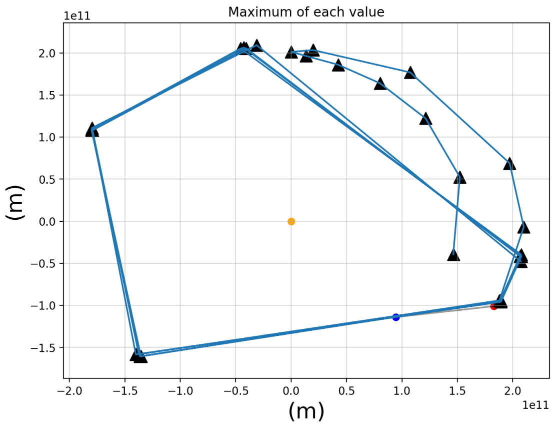
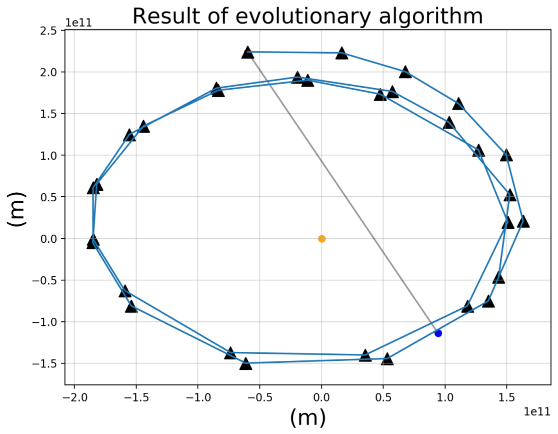
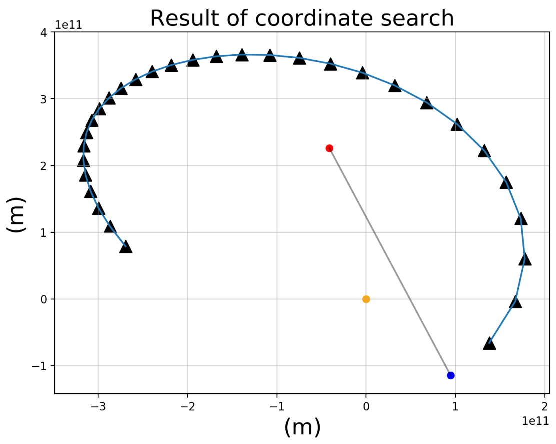
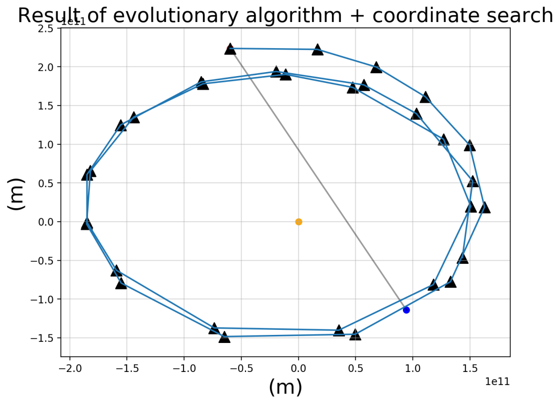
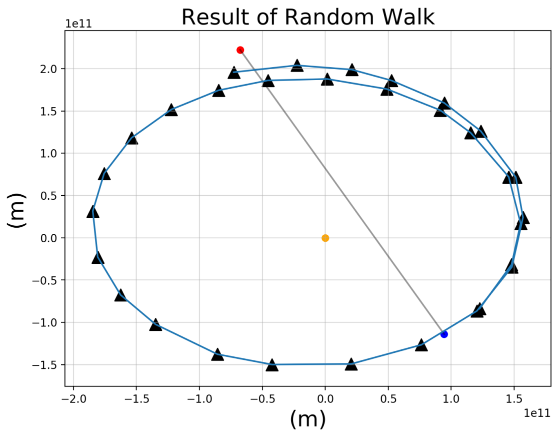


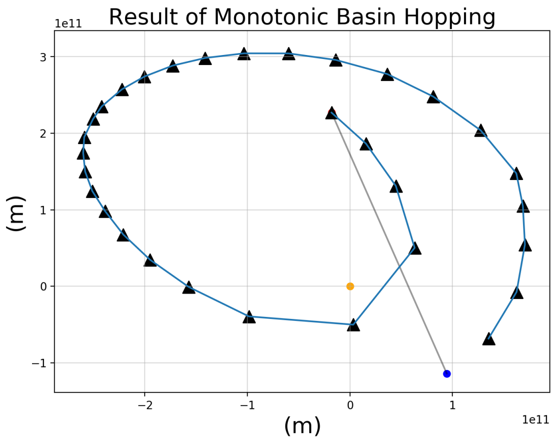
Figure 20: Trajectory flown from the Earth to Mars with the results minimum and maximum allowed values for the decision vector as inputs.






Result of evolutionary algorithm + coordinate search
(m)
(m)
1e11
1e11


Result of Random Walk
(m)
(m)
1e11
1e11


Result of Monotonic Basin Hopping
(m)
(m)
1e11
1e11

7 Appendix 2: Decision vectors to obtain the best solution

For the simplified case:

t_f = 98728252 (s)
vi_E = 1531 (m/s)

For the actual problem:

t_f = 90520975 (s)	vi_E = 1268 (m/s)
Delta V = 0.087	alpha = 0.303 (rad)
Delta V = 0.547	alpha = 0.586 (rad)
Delta V = 0.571	alpha = -1.734 (rad)
Delta V = 0.430	alpha = 1.273 (rad)
Delta V = 0.640	alpha = -0.429 (rad)
Delta V = 0.776	alpha = -1.158 (rad)
Delta V = 0.884	alpha = -1.009 (rad)
Delta V = 0.729	alpha = -1.222 (rad)
Delta V = 0.163	alpha = 2.546 (rad)
Delta V = 0.208	alpha = -2.570 (rad)
Delta V = 0.550	alpha = 0.283 (rad)
Delta V = 0.610	alpha = 0.417 (rad)
Delta V = 0.358	alpha = 1.758 (rad)
Delta V = 0.663	alpha = 2.416 (rad)
Delta V = 0.388	alpha = 1.160 (rad)
Delta V = 0.204	alpha = 2.998 (rad)
Delta V = 0.412	alpha = -2.143 (rad)
Delta V = 0.986	alpha = 0.792 (rad)
Delta V = 0.525	alpha = 0.180 (rad)
Delta V = 0.843	alpha = 0.778 (rad)
Delta V = 0.919	alpha = -2.014 (rad)
Delta V = 0.315	alpha = -0.805 (rad)
Delta V = 0.517	alpha = -0.064 (rad)
Delta V = 0.956	alpha = 0.048 (rad)
Delta V = 0.579	alpha = -0.980 (rad)
Delta V = 0.819	alpha = 1.440 (rad)
Delta V = 0.876	alpha = 1.754 (rad)
Delta V = 0.493	alpha = 1.714 (rad)
Delta V = 0.314	alpha = -1.435 (rad)

8 Main function for the model

The complete code that is needed is included in the folder with both the scripts and the notebooks. However, the most important pieces have been copied here.

For the optimization, the return value has been adapted.

```python
def main(DecV,Nimp,*args, **kwargs):
    plot = kwargs.get('plot', 'no')
    #Retrieve new decision variables
    timeflight, vi_E = DecV[0:2]
    deltav_magn = DecV[2:2+Nimp]
    deltav_angle = DecV[2+Nimp:]

    deltav_max = 90e-3* timeflight/ ((Nimp-1)*m_dry)
    m_0 = m_dry / np.exp(-(sum(deltav_magn*deltav_max) + vi_E) / (Isp*g0) )

    #Calculate position and velocity at the time of desired position
    rv_E, rv_M = calculateAngle(timeflight)

    #First impulse is from the Earth
    rv_E[3:] += np.array([rv_E[3], rv_E[4], rv_E[5]]) * vi_E / np.linalg.norm(rv_E[3:])
     ↪  #Perpendicular to the rotation of the Earth about the Sun

    #Planar assumption
    rv_E[2] = 0
    rv_E[-1] = 0
    rv_M[2] = 0
    rv_M[-1] = 0

    if plot == 'yes':
        plt.plot([rv_M[0],rv_E[0]],[rv_M[1],rv_E[1]],alpha = 0.4,color = 'black')
        plt.scatter(rv_E[0],rv_E[1],color = Ear.color)
        plt.scatter(rv_M[0],rv_M[1],color = Mar.color)
        plt.scatter(0,0,color = Sun.color)

    #Propagate
    rv_0 = rv_E
    TransferOrbit = AL_2B.BodyOrbit(rv_0, 'Cartesian',Sun) #Create transfer orbit object

#     if plot == 'yes':
#         plt.scatter(rv_0[0],rv_0[1],s=120,color="dodgerblue",marker="o")

    t_Imp = timeflight / (Nimp+1)
    scattermatrix = np.zeros([Nimp+1,2])
    for i in range(Nimp+1):
        rv = TransferOrbit.Propagation(t_Imp,'Cartesian', rv0 = rv_0)

        if i < Nimp:
            #Add impulse
            rv[3:] += deltav_max * deltav_magn[i] *
             ↪  np.array([np.cos(deltav_angle[i]),np.sin(deltav_angle[i]),0])
            TransferOrbit = AL_2B.BodyOrbit(rv, 'Cartesian',Sun) #Create transfer orbit object

        if plot == 'yes':
            plt.scatter(rv[0],rv[1],s=120,color = 'black',marker="^")
            scattermatrix[i,:] = rv[0:2]
    plt.plot(scattermatrix[:,0],scattermatrix[:,1])

    #Calculate error
    Error_pos = np.linalg.norm(rv[0:3]-rv_M[0:3])
    Error_vel = np.linalg.norm(rv[3:]-rv_M[3:])

    #Objective function
    m_fuel = m_0 - m_dry

    return m_fuel, Error_pos, Error_vel
```

9 Optimization algorithms

9.1 Evolutionary algorithm

```
def EvolAlgorithm(f, x, bounds, *args, **kwargs):
    """
    EvolAlgorithm: evolutionary algorithm
    INPUTS:
        f: function to be analyzed
        x: decision variables
        bounds: bounds of x to initialize the random function
        x_add: additional parameters for the function. As a vector
        ind: number of individuals.
        cuts: number of cuts to the variable
        tol: tolerance for convergence
        max_iter: maximum number of iterations (generations)
        max_iter_success
        elitism: percentage of population elitism
    """
    x_add = kwargs.get('x_add', None)
    ind = kwargs.get('ind', 100)
    cuts = kwargs.get('cuts', 1)
    tol = kwargs.get('tol', 1e-4)
    max_iter = kwargs.get('max_iter', 1e3)
    max_iter_success = kwargs.get('max_iter_success', 1e2)
    elitism = kwargs.get('elitism',0.1)
    mut = kwargs.get('mutation',0.01)

    ###############################################
    ###### GENERATION OF INITIAL POPULATION #######
    ###############################################
    pop_0 = np.zeros([ind, len(x)+1])
    for i in range(len(x)):
        pop_0[:,i+1] = np.random.rand(ind) * (bounds[i][1]-bounds[i][0]) + bounds[i][0]

    ###############################################
    ###### FITNESS EVALUATION               #######
    ###############################################
    for i in range(ind):
        pop_0[i,0] = f(pop_0[i,1:],x_add)

    Sol = pop_0[pop_0[:,0].argsort()]
    minVal = min(Sol[:,0])
    x_minVal = Sol[0,:]

    ###############################################
    ###### NEXT GENERATION                  #######
    ###############################################
    noImprove = 0
    counter = 0
    lastMin = minVal

    Best = np.zeros([100,len(x)+1])
    while noImprove <= max_iter_success and counter <= max_iter :
        ################################################
        #Generate descendents

        #Elitism
        ind_elit = int(round(elitism*ind))

        children = np.zeros(np.shape(pop_0))
        children[:,1:] = Sol[:,1:]

        #Separate into the number of parents
        pop = np.zeros(np.shape(pop_0))
        pop[:ind_elit,:] = children[:ind_elit,:] #Keep best ones
        np.random.shuffle(children[ind_elit:,:]) #shuffle the others


        for j in range ( (len(children)-ind_elit) //2 ):
            if len(x) == 2:
                cut = 1
            else:
                cut = np.random.randint(1,len(x)-1)
            pop[ind_elit +2*j,1:] = np.concatenate((children[2*j,1:cut+1],children[2*j+1,cut+1:]),axis = 0)
            pop[ind_elit+ 2*j + 1,1:] = np.concatenate((children[2*j+1,1:cut+1],children[2*j,cut+1:]),axis = 0)
        if (len(children)-ind_elit) %2 != 0:
            pop[-1,:] = children[-ind_elit,:]


        #Mutation
        for i in range(ind):
            for j in range(len(x)):
                if np.random.rand(1) < mut: #probability of mut
                    pop[i,j+1] =  np.random.rand(1) * (bounds[j][1]-bounds[j][0]) + bounds[j][0]

        ################################################
        # Fitness
        for i in range(ind):
            pop[i,0] = f(pop[i,1:],x_add)
        Sol = pop[pop[:,0].argsort()]
        minVal = min(Sol[:,0])

        ################################################
        #Check convergence
        counter += 1 #Count generations

        if  minVal >= lastMin:
            noImprove += 1
#         elif abs(lastMin-minVal)/lastMin > tol:
#             noImprove += 1
        else:
            lastMin = minVal
            x_minVal = Sol[0,:]
            noImprove = 0
            Best[counter,:] = Sol[0,:]
```

```

    print("minimum:", lastMin)
    print("Iterations:", counter)
    print("Iterations with no improvement:", noImprove)

    return x_minVal, Best
```

9.2 Cyclic Coordinate Search

```
def coordinateSearch(f,x,bounds,h,*args, **kwargs):
    """
    h: vector of length x, with the magnitude of the step
    """
    x_add = kwargs.get('x_add', None)
    tol = kwargs.get('tol', 1e-5)
    max_iter = kwargs.get('max_iter', 100)
    max_iter_success = kwargs.get('max_iter_success', 20)
    percentage = kwargs.get('percentage', 0.8)

    if np.all(x) == None:
        #Initial point
        x0 = np.zeros(len(x))
        for i in range(len(x)):
            x0[i] = np.random.rand(1) * (bounds[i][1]-bounds[i][0]) + bounds[i][0]

    x0 = x
    step = h
    improvement = 2*tol
    Prev_Min = 0
    counter = 0
    counter_success = 0

    while  improvement > tol and counter< max_iter and counter_success< max_iter_success:
        for i in range(len(x)): #Go through all variables
                x_plus = np.copy(x0)
                x_plus[i] += step[i]
                x_minus = np.copy(x0)
                x_minus[i] -= step[i]

                #Adjust to bounds
                if x_plus[i] >= bounds[i][1]:
                    x_plus[i] = x0[i]
                if x_minus[i] <= bounds[i][0]:
                    x_minus[i] = x0[i]

                #Evaluate
                fx = f(x0,x_add)
                fx_plus = f(x_plus,x_add)
                fx_minus = f(x_minus,x_add)

                Min = min([fx,fx_plus,fx_minus])
                if Min == fx_plus:
```

```python
                    x0 = x_plus
                elif Min == fx_minus:
                    x0 = x_minus

        improvement = abs(Prev_Min - Min)/Min
        if improvement == 0:
            improvement = 2*tol
            counter_success += 1
        else:
            counter_success = 0

        counter += 1
        step *= percentage

        Prev_Min = Min


    print('Iterations', counter)
    print('Iterations no improvement', counter_success)
    print('Min',Min)
    return x0, Min
```

9.3 Random Walk

```python
def randomJump(f,x,bounds,*args, **kwargs):
    x_add = kwargs.get('x_add', None)
    npoints = kwargs.get('npoints', 100)

    x0 = np.zeros([npoints,len(x)])
    for i in range(len(x)):
        x0[:,i] = np.random.rand(npoints) * (bounds[i][1]-bounds[i][0]) + bounds[i][0]

    fx = np.zeros([npoints,len(x)+1])
    for i in range(npoints):
        fx[i,0] = f(x0[i,:],x_add)
        fx[i,1:] = x0[i,:]

    Sol = fx[fx[:,0].argsort()]

    return Sol[0,:]
```