POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Evaluating Convex Solvers for Onboard Minimum-Fuel Trajectory Optimisation

Laurea Magistrale in Space Engineering - Ingegneria Spaziale

Author: Leoluca Grilli

Advisor: Dr. Alessandro Morselli

Co-advisors: Alessandra Mannocchi, Andrea Carlo Morelli

Academic year: 2022-2023

1. Introduction

As we stand on the cusp of a new era in space exploration, characterised by the audacious prospect of interplanetary journeys undertaken by CubeSats, new challenges and opportunities unfold. With advances in computational (CPU) capacities, the reality of autonomously performing the guidance and control of a spacecraft onboard is becoming more tangent. In the context of onboard trajectory optimisation, two types of methodologies emerge: direct and indirect methods. Indirect methods often exhibit poor convergence, long CPU times, and high sensitivity to the initial guess. At the same time, conventional direct methods suffer from CPU inefficiencies and poor robustness. In this context, the effectiveness of convex optimisation has quickly taken centre stage due to the presence of low computationally demanding solvers which guarantee convergence in polynomial time. Trajectory optimisation problems, however, are naturally non-convex, and need to be redefined through a convex subproblem that then has to be solved iteratively through the use of a Sequential Convex Program (SCP) and a convex solver. Within this framework, understanding how convex solvers behave becomes of the essence to investigate their use for onboard autonomous guidance. This thesis embarks on the evaluation of the performance of different methodologies of convex solvers when applied to fuel-optimal trajectory optimisation.

2. Convex Optimisation

The essence of convex optimisation involves the minimisation of an optimisation function, $\boldsymbol{o}$, subject to equality ($\boldsymbol{b}$) and inequality constraints ($\boldsymbol{g}$) defined in convex form. The optimisation problem is defined as [1]:

$$\begin{aligned} \underset{\boldsymbol{w}}{\text{minimise}} \quad & \boldsymbol{o}(\boldsymbol{w}) \\ \text{s.t.} \quad & \boldsymbol{C}\boldsymbol{w} = \boldsymbol{b} \ , \\ & \boldsymbol{g}(\boldsymbol{w}) \leq \boldsymbol{0} \ , \end{aligned} \tag{1}$$

where $\boldsymbol{w}$ are the variables and $\boldsymbol{C}$ is the equality matrix. A maximisation problem associated with Equation 1, called the dual problem, exists, and both problems are solved to obtain an optimal solution. Optimality of the primal-dual pair is guaranteed granted by the Karush-Kuhn-Tucker (KKT) optimality conditions [1]. A duality gap, represented by the difference between the primal and dual solutions, drives the convergence of the optimisation process.

2.1. Methods

Active-set methods: In the context of convex optimisation, three main methods exist. *Active-set methods* (ASMs) emerged as early as 1963, stemming from an effort to solve convex quadratic problems (QPs) by adapting the widely used simplex method of that time. These types of methods result in being extremely attractive for small QPs, with good results in terms of CPU time and accuracy. ASMs, work by iteratively updating the "working set" (a collection of the active constraints) by the addition or removal of constraints as progress towards the optimal solution is made [2]. Unfortunately, these methods do not provide a priori bounds on the number of iterations needed to find an optimal solution, and only accept feasible inputs.

Interior-point methods: Within the realm of optimisation techniques, *interior-point methods* (IPMs) stand out as the most widely used approaches in convex optimisation, providing efficiency and accuracy. A major advantage of IPMs is their constant CPU load (consistency) when solving various instances of related problems [2]. The original convex problem is modified by removing inequality constraints by introducing a penalising term in the objective function [1]. The resulting problem is then solved by navigating the interior of the feasible region using Newton's method, with this step accounting for the major CPU burden.

First-order methods: The evolution of *first-order methods* (FOMs) went parallel to the rise of data science and machine learning, where fast solutions to large amounts of input data are prioritised over high accuracy. These methods reach the optimal solution by using only first-order information, generally focused on gradients or sub-gradients. The splitting method is the underlying concept of the most widely used FOMs, which is focused on splitting $\boldsymbol{o}$ into two functions. Minimising the two functions is intended to be faster and simpler than minimising the single function $\boldsymbol{o}$ directly [2]. Although FOMs are simple to implement in terms of the amount of source code needed to build a solver, their performance is highly dependent on the characteristics of the problem being solved.

A comparison of the solver methods has been carried out and can be seen in Table 1. A colour code is used where, as for convention, green is used for a positive behaviour, red for a negative one, and orange for a measure in-between. The classification of the problem sizes has been made using the number of variables, constraints, and parameters of respectively 6, 15, and 18 for small problems (S) and 500, 3000, and 2000 for large problems (L). Using this classification, the minimum-fuel trajectory optimisation problem falls in the latter category.

Table 1: Comparison of convex solvers.

	ASMs	**FOMs**	**IPMs**
Implementation	Medium	Easy	Hard
Speed (S)	Good	Medium	Medium
Speed (L)	Worst	Medium	Best
Consistency	Medium	Worst	Best
Accuracy (L)	Low	Medium	High
Memory usage	Medium	Good	Good

From this table, it is clear that key differences exist for the solvers, and the most suitable one depends on the problem characteristics and level of accuracy required. IPM solvers are favoured when the number of variables and constraints is high. For small problems, because IPMs have a fixed cost when solving for the factorisation of the KKT matrix, the required CPU time to reach optimality is higher, with ASMs representing the ideal choice. ASMs can provide results with low accuracy for problems with large sizes, while FOMs can guarantee low to medium accuracy, and for high-accuracy solutions the IPMs are preferred. The data memory required to solve the optimisation problem is worse for ASM solvers as a large number of linear equations is solved compared to IPMs and FOMs, requiring a greater amount of memory to store the matrices.

2.2. Sequential Convex Programming

At the core every SCP lies the reformulation of an original optimisation problem into a convex subproblem. These subproblems are solved sequentially by feeding an initial guess and using the solution of each iteration to update the approximation for the next subproblem. Every SCP algorithm is composed of three major steps which include the generation of the initial guess, the resolution of the subproblem using a convex optimisation solver, and the termination of the algorithm.

2.3. Software selection

Following the review and the comparison of the different methods, it was clear that the poor scalability of ASM solvers for large problems, and their lack of a priori bounds on the complexity certification make them less suitable for the context of onboard trajectory optimisation compared to the FOMs and IPMs. In light of this, CVXPYgen[1] resulted in being the ideal candidate for this study, allowing to compare the state-of-the-art solvers ECOS (IPM) and SCS (FOM). This software allows to generate problem-specific solvers by incorporating the problem defined in a convex formulation in CVXPY[2] with the selected solver.

3. Methodology

The minimum-fuel problem can be represented by a two-point boundary-value problem, with the states at initial and final time forming the boundary conditions (BCs). The objective function, J, for fuel-optimal problems is defined in Mayer form as [3]:

$$J = -m(t_f)\,, \tag{2}$$

where minimising the fuel consumption of a single-stage spacecraft is equivalent to maximising its mass, m, at final time, t_f.

3.1. Convexification

In order to define the optimisation problem in convex form, the state of the spacecraft was described in spherical coordinates as

$$\boldsymbol{x} = [r,\ \theta,\ \phi,\ v_r,\ v_\theta,\ v_\phi,\ z]^T\,, \tag{3}$$

where r represents the distance from the central body to the spacecraft, v the velocity, θ the azimuth angle, and ϕ the elevation angle. z represents the pseudo-mass ($z = \ln(m)$) introduced by a change of variables. This technique was also applied to the control, $\boldsymbol{u}$, to decouple the state and the control in the dynamics, and avoid high-frequency jitters [3]:

$$\boldsymbol{u} = [\tau_r,\ \tau_\theta,\ \tau_\phi,\ \tau]^T\,, \tag{4}$$

where the control variable is $\tau = T/m$, and T is the thrust. The dynamics of the minimum-fuel problem can be represented by a linearisation with respect to a reference trajectory, $\boldsymbol{x}^*$:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}^*) + \boldsymbol{A}(\boldsymbol{x}^*)(\boldsymbol{x} - \boldsymbol{x}^*) + \boldsymbol{B}\boldsymbol{u}\,, \tag{5}$$

where $\boldsymbol{f}$ is the natural dynamics of the spacecraft, $\boldsymbol{A}$ is the derivative of $\boldsymbol{f}$ with respect to $\boldsymbol{x}^*$, and $\boldsymbol{B}$ represents the control matrix. The objective function can be redefined as

$$J = \int_{t_0}^{t_f} \tau(t)\, dt\,. \tag{6}$$

Following the methodology defined in Ref. [3], a convexification was applied to the control constraints and the dynamics to define the problem in convex form. Moreover, a trapezoidal discretisation method was used, forming a set of $N-1$ equality constraints, where N represents the number of nodes:

$$\boldsymbol{x}_i - \boldsymbol{x}_{i-1} = \frac{\Delta t}{2}(\dot{\boldsymbol{x}}_i - \dot{\boldsymbol{x}}_{i-1})\,. \tag{7}$$

3.2. SCP Algorithm

To obtain a solution to the original optimisation problem, the convex subproblem needs to be solved sequentially through an SCP, updating the reference trajectory at every iteration (k) with the optimal solution of the previous step. To prevent occurrences of artificial infeasibility, slack variables were introduced to the linearised dynamics and control. To improve the convergence of the algorithm, a trust region was applied using a Cauchy sequence and the solutions of the previous iterations:

$$\left\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\right\|_\infty \leq \gamma \left\|\boldsymbol{x}^{(k-1)} - \boldsymbol{x}^{(k-2)}\right\|_\infty\,, \tag{8}$$

where γ represents the trust region factor. At the first iteration, having only the knowledge of the initial guess $\boldsymbol{x}^{(0)}$, a constant trust region, $\boldsymbol{\delta}$, was applied. The convergence of the algorithm was specified on the maximum constraint violation (ε_c) and the relative change in pseudo-mass at final time (ε_z) between successive SCP iterations. The algorithm was stopped without convergence, instead, if not enough progress was made between SCP solution trajectories (ε_x) or

[1] Available at: <`https://github.com/cvxgrp/cvxpygen`> (last accessed on 12/10/2023)

[2] Available at: <`https://github.com/cvxpy/cvxpy`> (last accessed on 12/10/2023)

if the number of iterations exceeded 50. Algorithm 1 displays the logic behind the SCP along with the convergence values adopted.

Algorithm 1 SCP Algorithm

generate a reference trajectory $\boldsymbol{x}^{(0)}$
for k *in* $k_{\max}$ **do**
  $\boldsymbol{x}^* = \boldsymbol{x}^{(k-1)}$
  solve the optimisation problem to find $\boldsymbol{x}^{(k)}$ and $\boldsymbol{u}^{(k)}$
  if $\varepsilon_c \leq 10^{-6}$ & $\varepsilon_z \leq 10^{-4}$ **then**
    optimal solution found: $\boldsymbol{x}^{opt} = \boldsymbol{x}^{(k)}$
    $\boldsymbol{u}^{opt} = \boldsymbol{u}^{(k)}$
    break
  else if $\varepsilon_x \leq 10^{-7}$ **then**
    progress is too small: break
  end if
end for

The initial guess was generated using a third-order polynomial, with the controls set to zero.

4. Results

All simulations were run on the same Microsoft Surface Laptop 4, powered by an 11th Gen Intel(R) Core(TM) i5-1135G7 and with 16 GB RAM. A Monte Carlo (MC) simulation of 100 samples was run, applying a standard deviation of 10% to the required final position, $\boldsymbol{x}_f$, in order to generate a perturbed initial guess. While the trajectories produced might diverge drastically from the optimal solution, it was deliberately designed to challenge the solvers to their utmost capacity and provide a comprehensive assessment of their performance.

4.1. Experimental Setup

Adopting the SCP scheme presented in Algorithm 1, three test cases of increasing complexity were used to evaluate the performance of the solvers. These problems include an Earth-to-Mars transfer with BCs taken from [3], a CubeSat transfer from Sun–Earth Lagrange point L_2 to near-Earth asteroid (NEA) 2000 SG344 taken from [4], and finally an Earth-to-Venus transfer, also taken from [4]. The simulation parameters for the three problems are given in Table 2, where m_0 is the initial mass, $T_{\max}$ the maximum thrust, and I_{sp} the specific impulse. Due to the limits CVXPYgen presented on the number of parameters that can be employed, the number of nodes used for the three different simulations were of 100, 125, and 150, chosen to reflect the increasing time of flight (ToF) of the transfers.

Table 2: Simulation parameters.

Parameters	Mars	NEA	Venus
ToF (days)	253	700	1000
m_0 (kg)	659.3	22.6	1500
$T_{\max}$ (N)	0.55	2.25e-03	0.33
I_{sp} (s)	3300	3067	3800
N (-)	100	125	150
γ (-)	0.7	0.8	0.99

4.2. Parameter Selection

A parametric analysis was carried out to determine the values of the trust region factor, γ, which granted a fair comparison for the two solvers, analysing accuracy, CPU performance and complexity. The results showed a clear correlation with problem complexity, with tighter constraints imposed for the simpler cases as shown in the table above. The limit on the maximum number of iterations for each solver was also investigated, resulting in a higher limit of 2,500 for SCS compared to 100 for ECOS.

4.3. Trade-off Criteria Analysis

4.3.1 Complexity

The complexity of the solvers was measured through the number of SCP and solver iterations required to reach optimality. ECOS shows a constant behaviour between problems, with SCS exhibiting a more significant increase in complexity when passing from the Mars case to the NEA case. Moreover, the number of solver iterations is much more significant for SCS, with values reaching 37,500 for the Venus test case compared to 118 for ECOS.

4.3.2 CPU Performance

The CPU toll of the solver was measured through the total CPU time required to obtain convergence and the average time required to solve an SCP iteration, namely the burden and efficiency. Both methods show a linear increase in the CPU parameters, although the values of

the FOM remain one order of magnitude higher. ECOS was remarkably fast, with solutions spanning from 0.344 s for the Mars case to 0.711 s for the Venus trajectory. In contrast, SCS results in a change from 2.438 s to 18.938 s for the same cases, with a more pronounced increase.

4.3.3 Memory

The compiled codes obtained through CVXPYgen show an increase in size with problem complexity, and hence nodal number, with results from ECOS being 18% larger, indicative of the more complex operations performed by the IPM solvers. The memory consumption of the solvers during the SCP algorithm, on the other hand, reveals a better behaviour for ECOS, with an almost constant memory usage which increases linearly from 164.979 MB for the Mars case to 170.527 MB for the Venus case. Results are also very consistent between different runs, with a maximum standard deviation of 37 kB. The FOM, instead, has a larger sparseness of results and a greater increase between cases as shown by the error bars in Figure 1, reaching 291.414 MB for the Venus trajectory.

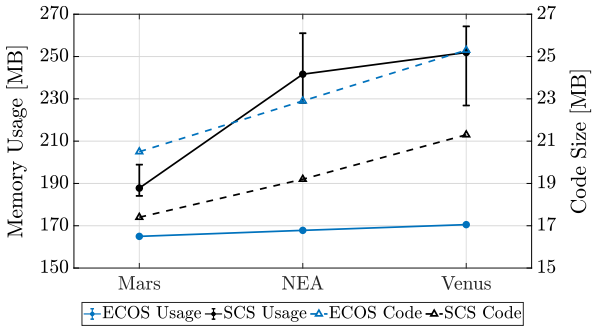


Figure 1: Memory of compiled code.

4.3.4 Accuracy

The accuracy of the solution, ϵ, was found by propagating the optimised control vector, $\boldsymbol{u}^{opt}$, using the `scipy`[3] integrator `odeint`. As the values of $\boldsymbol{u}^{opt}$ are known only at the nodes, the control in between the nodes was obtained by interpolating the control history. Due to the discretisation used, there is a difference in solution between the desired final state, $\boldsymbol{x}_f$, and the propagated one, $\tilde{\boldsymbol{x}}$, at final time:

$$\epsilon = |\tilde{\boldsymbol{x}}(t_f) - \boldsymbol{x}_f| \,. \tag{9}$$

The values obtained for the two solvers are comparable, which is expected as the accuracy of the optimal solution is dependent on the level of tolerance set in the solver. The tolerance is imposed on the relative and absolute values of the duality gap (see section 2), which was selected to be of 10^{-8} for this study. Both solvers show a reduction in the accuracy with problem complexity, which is attributed to the increase in ToF. As trajectories become more prolonged, small errors in the initial states can lead to larger errors in the final state, when propagated, and the interpolation error becomes more significant.

4.3.5 Optimality

As with the accuracy, the values of the optimality (J) of the solvers increase with the ToF, with very similar values for the IPM and FOM solvers. Recalling the problem definition, the objective of the minimum-fuel problem is reflected in maximising the final mass. It is therefore understandable that the final masses obtained from the solvers are comparable, with a maximum difference of 17 g between the median MC results of the two solvers.

4.3.6 Reliability

The reliability of the solvers is defined as the percentage of converged optimal solutions among all MC samples, and is shown in Figure 2 with varying convergence tolerances on ε_c. The IPM solver is superior, with consistently high success rates for all cases. Tighter convergence criteria highlight the discrepancies between the solvers, showing a change in behaviour for the FOM.

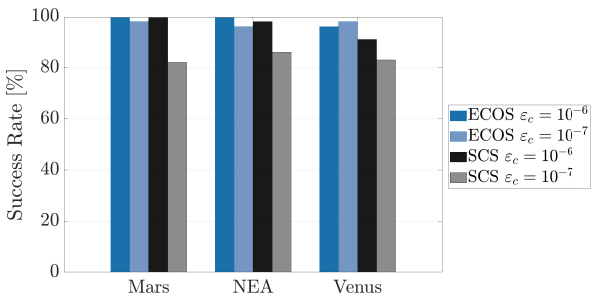
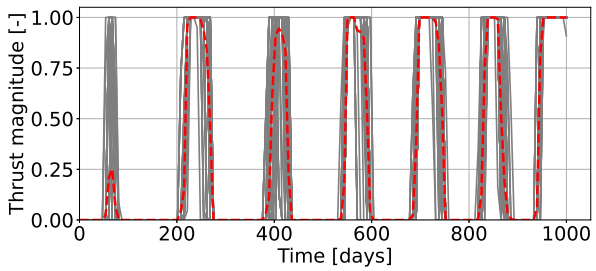


Figure 2: Reliability of simulations.

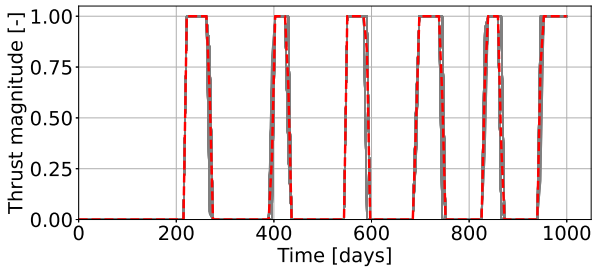
[3] Available at: <`https://scipy.org/`> (last accessed on 13/11/2023)

4.3.7 Effectiveness

When judging the consistency of a method, it is essential to observe how the parameters vary between different runs and from one case to another. The thrust profiles of the MC samples show an interesting behaviour. For simpler problems, ECOS is more consistent, reaching the same optimal solution over repeated simulations while SCS shows a greater variance in profiles. The Earth–Venus trajectory, instead, results in extremely congruent bang-bang profiles for the FOM solver, whereas the IPM solver obtains two conflicting solutions. This can be seen in Figure 3, where the thrust profiles of the converged MC samples are shown in grey and the mean of the samples taken at every node in red.



(a) ECOS soultions.



(b) SCS solutions.

Figure 3: Earth–Venus thrust profiles.

SCS therefore behaves better for the most complex scenario in terms of trajectory solution compared to ECOS, and this is also reflected in its lower variation of the final mass of 11.448 g compared to 191.998 g for ECOS.

4.4. CVXPY Analysis

The CVXPY software was used to investigate the solutions obtained with ECOS, using the other available IPM solvers for the Earth–Mars trajectory. All the simulations carried out with the IPM solvers converged to an optimal solution, highlighting the consistency of these methods in terms of accuracy, complexity and memory usage. The simulations using ECOS and SCS showed the benefit of using a compiled code instead of a non-compiled one, with a total CPU improvement from 45.836 s to 0.344 s for ECOS and 55.898 s to 2.438 s for SCS when passing from CVXPY to CVXPYgen.

5. Conclusions

In conclusion, the trajectories investigated show a similar result in terms of accuracy and final mass. The behaviour of the IPM solver remains superior in terms of the computational toll, success rate, and memory usage when compared to the FOM solver. Moreover, it has been found that for simpler problems ECOS results in being the preferred solver of choice, with consistent bang-bang thrust profiles. For more complex trajectories like the Earth–Venus case, instead, although SCS exhibits a lower reliability of 82% and a longer CPU time of 11.444 s, the standard deviation of the accuracy is smaller, with much more consistent thrust profiles making it an extremely attractive solution.

References

[1] Domahidi A. *Methods and tools for embedded optimization and control.* Doctor of sciences dissertation, ETH ZURICH, 2013. doi:10.3929/ETHZ-A-010010483.

[2] Ferreau H. J. et al. Embedded optimization methods for industrial automatic control. *IFAC-PapersOnLine*, 50(1):13194–13209, 2017. doi:10.1016/j.ifacol.2017.08.1946.

[3] Wang Z. and Grant M. J. Minimum-fuel low-thrust transfers for spacecraft: A convex approach. *IEEE Transactions on Aerospace and Electronic Systems*, 54(5):2274–2290, 03 2018. doi:10.1109/TAES.2018.2812558.

[4] Hofmann C., Morelli A. C., and Topputo F. Performance assessment of convex low-thrust trajectory optimization methods. *Journal Of Spacecraft And Rockets*, 60(1):299–314, 01 2023. doi:doi:10.2514/1.A35461.