POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Evaluating Convex Solvers for Onboard Minimum-Fuel Trajectory Optimisation

Tesi di Laurea Magistrale in
Space Engineering - Ingegneria Spaziale

Author: **Leoluca Grilli**

Student ID: 10800769
Advisor: Dr. Alessandro Morselli
Co-advisors: Alessandra Mannocchi, Andrea Carlo Morelli
Academic Year: 2022-23

Abstract

In recent years, convex optimisation has emerged as a versatile and efficient method to obtain fast and reliable solutions in time-critical applications. This study focused on evaluating the behaviour of different convex solvers for minimum-fuel low-thrust interplanetary problems within the context of onboard applications. Three distinct test cases of increasing complexity were explored to evaluate the effectiveness of convex methods, covering the Earth-to-Mars, SEL$_2$-to-NEA, and Earth-to-Venus transfer problems.

Two solvers, ECOS (interior-point method, IPM) and SCS (first-order method, FOM), were chosen for their prominence in their respective solver family. To ensure fair testing, parameters like the trust region factor and the maximum number of allowable operations were carefully considered.

The Monte Carlo simulation results revealed significant differences between the two methods, with ECOS displaying superior and consistent performance in key criteria such as memory usage and computational burden. SCS, instead, exhibited a greater inconsistency, with values that showed a more prominent variation for the simpler cases. While the accuracy and optimality of solutions were comparable between the two methods, ECOS displayed higher and constant reliability (with a success rate always above 96%) compared to the lower and deteriorating rate of SCS (reaching as low as 82%). Analysing the thrust profiles highlighted a greater consistency for the IPM solver for simpler cases, while for the Earth–Venus trajectory the FOM solver exhibited a more coherent behaviour.

The study also considered the computational aspect, highlighting the advantages of using compiled code over a non-compiled one. CVXPY and CVXPYgen were compared, showing significantly faster results for ECOS.

Keywords: Guidance, Convex, Optimisation, Fuel-optimal, Deep-space

Sommario

Negli ultimi anni, l'ottimizzazione convessa è emersa come un metodo versatile ed efficiente per ottenere soluzioni rapide ed affidabili in applicazioni che richiedono tempi critici. Questo studio si è concentrato sulla valutazione del comportamento di diversi risolutori convessi per problemi di minimo consumo di carburante nel contesto delle applicazioni di bordo per missioni interplanetarie con bassa spinta. Sono stati esplorati tre casi di crescente complessità al fine di valutare l'efficacia dei metodi convessi tra cui i trasferimenti Terra–Marte, SEL$_2$–NEA e Terra–Venere.

Sono stati scelti due risolutori, ECOS (metodo a punto interno, IPM) e SCS (metodo del primo ordine, FOM), per la loro notorietà nelle rispettive famiglie di metodi. Per garantire test equi, sono stati attentamente considerati parametri come il trust region e il numero massimo di operazioni consentite.

I risultati delle simulazioni Monte Carlo hanno rivelato significative differenze tra i due metodi, con ECOS che ha dimostrato una prestazione superiore e costante in criteri chiave come la memoria utilizzata e il carico computazionale. SCS, invece, ha mostrato una maggiore incostanza, con valori che presentavano una variazione più accentuata per i problemi più semplici. Sebbene l'accuratezza e l'ottimalità delle soluzioni fossero confrontabili, ECOS ha esibito una maggiore affidabilità, con un tasso di successo migliore e costante (sempre superiore al 96%) rispetto all'inferiore e deteriorante tasso di SCS (raggiungendo l'82%). Analizzando i profili di spinta, è emersa una maggiore coerenza per il risolutore IPM nei casi più semplici, mentre per la traiettoria Terra–Venere il risolutore FOM ha mostrato un comportamento più coerente.

Lo studio ha anche considerato il tempo computazionale, evidenziando i vantaggi dell'uso di codice compilato rispetto a quello non compilato. CVXPY e CVXPYgen sono stati confrontati, mostrando risultati significativamente più rapidi per ECOS.

Parole chiave: Guida, Convessa, Ottimizzazione, Propellente-ottimale, Spazio profondo

Ringraziamenti

Ringrazio il dottor Morselli per avermi concesso l'opportunità di sviluppare questa tesi, dandomi la possibilità di approfondire un aspetto molto attuale ed importante soprattutto nel prossimo futuro. Ringrazio il professor Topputo per avermi innanzitutto trasmesso attraverso i suoi insegnamenti, la sua passione e conoscenza. In particolare, ringrazio Alessandra ed Andrea per avermi aiutato, soprattutto nelle prime fasi di lavoro dove ho fatto tesoro dei loro consigli per affrontare un argomento nuovo.

Ringrazio poi i miei genitori Marco e Manuela che sono sempre stati al mio fianco. Da quando ho lasciato casa per intraprendere una carriera universitaria a Leeds ho sempre fatto tesoro dei vostri consigli, sia lavorativi ma soprattutto personali. Ringrazio i miei fratelli Leonardo e Leonora, su cui so di poter contare sempre ed i miei nonni e mio zio per la vicinanza che mi hanno sempre dimostrato. Ringrazio i miei nipotini, che con il loro affetto e la loro spensieratezza mi ricordano ogni giorno quali siano le cose veramente importanti nella vita.

Un ringraziamento speciale va a Bianca, una ragazza che ho la fortuna di poter veder crescere e maturare a fianco a me. La prima volta che ti ho visto è stato come essere travolto da una brezza d'aria fresca, e riesci a trasmettermi questa sensazione ogni giorno. Grazie per tutte le domande del giorno, le avventure e gli scherzi che abbiamo condiviso e per tutte le volte che sei stata accanto a me, sapendomi ascoltare e consigliare, ma senza mai giudicare. In questo periodo pieno di cambiamenti sei la mia certezza. Conoscerti è stata la cosa più bella che io abbia fatto.

Grazie ai miei amici del "Cage" Francesco, Kaan, Michele, Susy, Teresa e Vittoria che con serate passate insieme a mangiare, vedere film e fare giochi da tavolo hanno reso Milano casa. Grazie a Marco, con cui ho avuto l'onore di condividere la casa per più di due anni. Non tutti i coinquilini diventano amici, ma in te ho scoperto una persona con cui mi posso aprire e confidare.

Grazie agli amici che ho potuto conoscere attraverso il politecnico, e che sono diventati dei punti fissi nella mia vita nonostante le distanze o il lavoro. Grazie ad Elisa e Francesco per esserci sempre stati nei momenti più belli ed in quelli più buoi. Grazie al Trio Medusa, André e Gonçalo, con cui ho avuto l’onore di lavorare insieme su alcuni progetti e con cui ho condiviso diverse memorie che mi porterò sempre dentro.

Contents

Abstract **i**

Sommario **iii**

Ringraziamenti **v**

Contents **vii**

1 Introduction **1**

1.1 Objectives of the Thesis . . . 3

1.2 Research Question . . . 3

1.3 Limitations . . . 4

1.4 Thesis Outline . . . 4

2 Theoretical Background **7**

2.1 Convex Optimisation Methods . . . 7

2.1.1 Interior-Point Methods . . . 9

2.1.2 First-Order Methods . . . 14

2.1.3 Active-Set Methods . . . 15

2.1.4 Method Comparison . . . 16

2.2 Convex Optimisation in Space Guidance . . . 18

2.2.1 Sequential Convex Programming . . . 20

2.3 Survey of Available Software . . . 21

2.3.1 Tools . . . 21

2.3.2 Solvers . . . 23

2.4 Trade-off Criteria Selection . . . 27

3 Methodology **31**

3.1 Dynamical Model . . . 31

3.1.1 Equations of Motion in Spherical Coordinates . . . 33

3.1.2 Normalisation 33
3.2 Optimal Control Problem 34
3.3 Successive Convexification Method 36
3.3.1 Change of Variables 36
3.3.2 Convexification 39
3.3.3 Discretisation 41
3.4 Sequential Convex Programming Algorithm 43
3.4.1 Artificial Infeasibility 43
3.4.2 Trust Region 44
3.4.3 Convergence Criteria 45
3.4.4 Initial Guess Generation 46
3.4.5 Algorithm 47

4 Numerical Results **51**
4.1 Experimental Setup 52
4.2 Parametric Analysis 55
4.3 Results Validation 59
4.4 Trade-off Criteria Analysis 64
4.4.1 Memory 64
4.4.2 Complexity 65
4.4.3 Computational Burden and Efficiency 66
4.4.4 Accuracy 67
4.4.5 Optimality 69
4.4.6 Reliability 70
4.4.7 Effectiveness 71
4.5 CVXPY Analysis 74

5 Conclusions and Future Work **77**
5.1 Conclusion 77
5.2 Future Work 79

List of Figures **81**

List of Tables **83**

Nomenclature **85**

Bibliography **91**

A Parametric Investigation **99**

A.1 Trust Region Factor 99

A.2 SCS Operations 102

B Verification and Validation **105**

B.1 Accuracy Verification 105

B.2 Control Validation 107

C GitLab Repository **109**

1 | Introduction

Throughout history, the mystery of space has captured the human imagination, fostering a relentless desire to explore the cosmos. Concurrently, the field of optimisation has progressed in tandem with the pursuit of space exploration. Within this domain, the challenges presented by space missions carry the complex task of generating trajectories to achieve predetermined positions and velocities, while optimising variables such as fuel consumption and travel time.

As we stand on the cusp of a new era in space exploration, characterised by the audacious prospect of interplanetary journeys undertaken by small yet promising CubeSats, new challenges and opportunities emerge. The canvas of exploration for small spacecraft broadens, transcending the boundaries of our immediate celestial vicinity comprised by our planet and its moon. In the midst of these dynamic developments, a crucial necessity emerges: the ability to navigate the complexities of interplanetary space with precision and autonomy, ensuring not only the attainment of desired states but also their attainment through optimal solutions. As computational capacities have undergone a high development in recent years, the reality of performing the guidance and control of a spacecraft onboard instead of on ground is becoming more tangent [1, 2]. On one hand, autonomy is essential for the spacecraft to make real-time decisions in the dynamic environment of space, without the need to wait for significant communication delays due to the considerable distance from Earth. On the other, the need for onboard guidance is a result of the limitations of ground stations which may have limited visibility, availability, and capacity.

In this context, two types of methodologies emerge to obtain an optimal solution: direct and indirect methods [3, 4]. The latter is formulated to satisfy the first-order necessary conditions and indirectly solve for the optimal control law through the unknown co-states (Lagrange multipliers), obtaining a continuous-time solution. A benefit of these methods over the direct methods is that they allow for higher flexibility and resolution with respect to changes in the states or co-states [5]. That being said, the information of the co-states does not have a clear physical meaning, and these methods also suffer from poor convergence, long computational times, and high sensitivity to the initial guess [4].

Direct optimisation methods, on the other hand, solve the optimisation problem through the minimisation of a cost function and the disctretisation of the states and controls across a given time-horizon, with a set of constraints imposed at every discretisation point [6]. In the realm of direct methods, the effectiveness of convex optimisation has quickly taken centre stage, becoming one of the preferred techniques for swift and accurate trajectory optimisation problems [7, 8]. These techniques compensate for the computational inefficiencies typically associated with direct methods by reformulating the problem in a convex form. Leveraging the effectiveness of cutting-edge interior-point methods (IPMs), convex optimisation problems can be rapidly solved. Their ability to guarantee convergence in polynomial time makes these methods extremely attractive and suitable for onboard autonomous operations like the minimum-fuel low-thrust trajectory optimisation problem [9, 10]. These types of problems, however, are naturally non-convex due to the strong coupling of the states in the dynamics and the high non-linearities present in the dynamics of the problem [11]. Therefore, different discretisation and convexification techniques have to be applied to the original optimisation problem to reformulate it in a convex form [1]. The linearisation of the dynamics around a reference trajectory forms a subproblem, which has to be solved iteratively through the use of a Sequential Convex Program (SCP) and a convex solver in order to obtain a solution with a predefined level of accuracy. The two main alternatives to the IPMs are represented by the first-order methods (FOMs) and the active-set methods (ASMs).

This thesis embarks on a comprehensive exploration of the capabilities of convex optimisation solvers, scrutinising their performance in the unique landscape of deep space. By means of meticulous analysis and empirical evaluation, this study aims to outline the strengths and limitations of these methods, shedding light on their potential to aid the trajectory optimisation of interplanetary missions, whether conducted by conventional spacecraft or cutting-edge CubeSats.

As humanity's aspirations extend beyond the boundaries of our home planet, the pursuit of autonomous guidance systems assumes an urgency that is both trivial and imperative. The computational efficiency and reliability proven by convex optimisation have made the algorithms particularly popular and attractive in industries where speed and accuracy are required, so much so that important companies like NASA and SpaceX have adopted them in their projects with vast success [8, 12, 13]. Having scratched only the surface of their capabilities the future for these methods remains bright and promising.

1.1. Objectives of the Thesis

The advances made in the optimisation realm have seen an increased use of the application in which SCP and model predictive control (MPC) processes have been used [4]. In order to allow a solution to be generated there is a necessity to exploit solvers which implement an optimisation method until convergence is reached. Convex optimisation represents the family in which state-of-the-art solvers are available. This is because the solution obtained from these methods is proven to be a global solution while allowing for competitive convergence results at small computational costs [14, 15]. The solvers used within the convex framework can be divided into three main methods: ASMs, FOMs, and IPMs. Although comparative studies of different types of convex methods exist for some types of problems, like the Tennessee Eastman reactor and the Control of Isothermal CSTRs in Series [16–18], these problems remain far from the deep-space application. Moreover, these studies focus on the comparison of only ASMs and IPMs, ignoring the very promising and used FOMs. It is therefore clear that there is a gap to be filled in the literature. This thesis aims to do so by providing a basis to compare and evaluate solvers for deep-space fuel-optimal problems and in doing so unraveling the performance of convex optimisation solvers in this context.

1.2. Research Question

The aim of this thesis is to contribute to the field of autonomous guidance in deep-space through a comprehensive examination of the performance of convex methods. In pursuit of this goal, the main research question that this study seeks to address is as follows:

How does the overall performance of convex solvers compare when applied to fuel-optimal trajectory optimisation?

In order to allow for sufficient data to be gathered, and to mitigate the potential for any bias, it becomes imperative to thoroughly evaluate a range of scenarios of varying difficulty that can effectively address this inquiry. Three minimum-fuel problems of increasing difficulty (i.e., Earth to Mars, Sun-Earth Lagrange point L_2 to near-Earth asteroid 2000 SG344, and Earth to Venus) are solved in spherical coordinates using an SCP and different convex solvers. The findings of this study are expected to serve as an initial reference for selecting an appropriate convex optimisation solver, taking into account the specific requirements of the mission and the desired level of performance of the solver. Furthermore, as this study aims to assess the feasibility of implementing the proposed algorithm and solvers onboard, it is crucial to address the constraints and challenges unique to this application. In fact, the evaluation of the performance of the solvers should include key criteria essential for space applications, such as code size and memory consumption during the optimisation process.

1.3. Limitations

In order to facilitate a comparison of different types of methods used in convex optimisation, the scope of this study is to investigate how the performance criteria of ECOS (an interior-point solver) and SCS (a first-order solver) vary with an increase in problem complexity. These two solvers have been selected after a careful evaluation of the available software and tools on the market which is reported in Section 2.3.

It can be noticed how the solvers investigated do not include ASMs, with this choice being made after selecting the CVXPYgen software as the ideal candidate for this study. CVXPYgen provides a way to generate a compiled custom solver in C code, which can significantly enhance computational speed compared to non-compiled code [19]. However, this software is relatively new and still in the development phase, and hence it does not currently support all the solvers which are instead available in CVXPY. Furthermore, after a careful evaluation of ASMs (seen in Subsection 2.1.3) it was found that the solvers corresponding to this family are less developed and unsuitable for problems with a large number of variables and constraints [7, 20, 21], justifying their absence from this study. No further effort was made to incorporate external ASM solvers into CVXPYgen.

Another limitation of this study is the number of solvers investigated for each method. This limitation has been deemed acceptable because both ECOS and SCS are state-of-the-art solvers for their respective method family, at the time of writing. Having been used with great success over the years, their results would therefore represent the acme of their respective solver families. Moreover, an effort was made to exploit the remaining IPM solvers available in CVXPY and investigate the results obtained for ECOS in terms of performance, where possible (see Section 4.5).

1.4. Thesis Outline

The work of the thesis is organised into three main chapters. In Chapter 2 an extensive overview of convex optimisation applied to guidance in space is given, while also presenting the three predominant methodologies used within this optimisation domain and highlighting the array of available tools and solvers. Moreover, the chapter expands upon the criteria used to properly undergo a comparative analysis among the solvers.

Chapter 3 subsequently dives into greater detail, describing the selected reference frame and the dynamics model adopted. The successive convexification method of the original non-convex problem is presented through a series of change of variables, convexification, and discretisation techniques. The chapter concludes by providing insight in the building of the SCP along with an overall view of the algorithm.

The scenarios analysed are then presented in Chapter 4, along with an in-depth account of the parameters chosen for the SCP and solvers, supported by thorough reasoning and appropriate investigations. The results obtained from the simulations are then reported and extensively discussed. Chapter 5 concludes the study with a summary of the problem analysed and the outcomes, offering prospects for potential future works in this field.

2 | Theoretical Background

Space engineering challenges are inherently tied to optimisation. This fundamental concept serves as a cornerstone in solving a diverse range of problems within the field. Optimisation underpins the very essence of space exploration and engineering, allowing to solve problems that span from the optimisation of fuel consumption required to reach a specific target to the generation of the trajectory necessary to precisely land a rocket or spacecraft [7]. For onboard applications, this has to be done while ensuring that two fundamental attributes remain at a minimum: computational time and memory usage. The former is particularly pertinent in real-time scenarios, where the ability to swiftly adapt a planned trajectory or course in response to unforeseen variations can result in significant savings. Conversely, the second attribute is a direct result of the limitations associated with space-proven boards, which do not typically possess ample memory. Among optimisation methods, convex optimisation includes solvers that grant high reliability, with a large number of converged solutions, while allowing for small computational cost [14, 15].

An introduction to the convex optimisation problem and a description of the main methods used to solve it is provided in Section 2.1. Then, Section 2.2 reports the use of convex optimisation in space guidance applications, with the taxonomy of optimisation problems and a background on SCPs. Section 2.3 dives into the available software, providing a survey of the tools and solvers on the market at the time of writing. Section 2.4 then concludes the literature review by analysing the necessary criteria needed for a correct and fair comparison of the performance of the solvers.

2.1. Convex Optimisation Methods

Before diving into the technical differences between the methods used, it is important to state the convex optimisation problem [22]:

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimise}} \quad & \boldsymbol{f}(\boldsymbol{x})\,, \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \;, \\ & \boldsymbol{g}(\boldsymbol{x}) \leq \mathbf{0} \;, \end{aligned} \tag{2.1}$$

where $\boldsymbol{x}$ represents the variables, $\boldsymbol{b}$ the equality constraints, and the optimisation function $\boldsymbol{f}(\boldsymbol{x})$ and the inequality constraint function $\boldsymbol{g}(\boldsymbol{x})$ are both defined in a convex form [22]. A function is deemed convex if the line segment connecting any two points on the graph of the function lies above the graph between the two points. This problem is known in convex optimisation as the primal problem. The feasible set, $\boldsymbol{Q}$, refers to the set of all possible solutions that satisfy the constraints of the optimisation problem and takes a value of $\mathbf{0}$ when the problem is primal infeasible. If instead a solution exists where $\boldsymbol{x} \in \boldsymbol{Q}$, the problem is feasible, whereas if the values of the inequality constraints lie on the interior (i.e. $\boldsymbol{g}(\boldsymbol{x}) < \mathbf{0}$) the solution is said to be strictly feasible. When an optimal solution is instead obtained, the solution $\boldsymbol{x}^{opt} \in \boldsymbol{Q}$ is called the minimiser as it generates the minimal value of the objective function. The function $\boldsymbol{d}$, called the Lagrange dual function, associated to Equation 2.1 is defined as

$$\boldsymbol{d}(\boldsymbol{y}, \boldsymbol{z}) = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{y}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) + \boldsymbol{z}^T\boldsymbol{g}(\boldsymbol{x}) \;, \tag{2.2}$$

where $\boldsymbol{y}$ and $\boldsymbol{z}$ are the Lagrange multipliers for the equality and inequality constraints respectively. In convex optimisation, a problem associated with Equation 2.1, called the dual problem, exists, which involves a maximisation of the form [22]:

$$\begin{aligned} \underset{\boldsymbol{y},\,\boldsymbol{z}}{\text{maximise}} \quad & \boldsymbol{d}(\boldsymbol{y},\, \boldsymbol{z})\,, \\ \text{s.t.} \quad & \boldsymbol{z} \geq \mathbf{0} \;. \end{aligned} \tag{2.3}$$

A measure which is of extreme importance for convex optimisation methods is the duality gap, which represents the difference between the optimal solutions of the primal and dual problems ($f^{opt} - d^{opt} > 0$) [22]. What drives the convergence of the optimisation process is the tolerance imposed (both absolute and relative) on the duality gap, below which a solution is deemed optimal for the problem. Optimality of the primal-dual pair is granted by the Karush-Kuhn-Tucker (KKT) optimality conditions [21, 22], summarised below.

$$\begin{cases} \nabla \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{A}^T \boldsymbol{y} + \boldsymbol{G}(\boldsymbol{x})^T \boldsymbol{z} = \boldsymbol{0} & \text{(2.4a)} \\ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} & \text{(2.4b)} \\ \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{s} = \boldsymbol{h} & \text{(2.4c)} \\ \boldsymbol{s}_i \boldsymbol{z}_i = 0 \quad \text{for } i = 1, ..., p & \text{(2.4d)} \\ (\boldsymbol{s}, \boldsymbol{z}) \geq \boldsymbol{0}\,, & \text{(2.4e)} \end{cases}$$

where the inequality constraint $\boldsymbol{g}(\boldsymbol{x}) \leq 0$ is transformed into an equality constraint through the positive slack variable $\boldsymbol{s}$, p represents the dimensions of the problem, and $\boldsymbol{G}(\boldsymbol{x})$ represents the Jacobian of $\boldsymbol{g}(\boldsymbol{x})$ evaluated at $\boldsymbol{x}$. The theory behind the KKT conditions is that an optimal solution to the problem exists when the derivatives of the Lagrangian function with respect to $\boldsymbol{x}$ and the Lagrange multipliers, are both zero [23]. Several state-of-the-art optimisation methods are based on the KKT conditions, making them a trivial requirement for optimisation algorithms [24].

2.1.1. Interior-Point Methods

Within the realm of optimisation techniques, interior-point methods stand out as the most widely used approaches for tackling convex optimisation problems, providing efficiency and accuracy [18]. The first application of IPMs dates back to the Karmarkar algorithm, developed by Narendra Karmarkar in 1984, which gained significant interest within the optimisation community itself [25]. This method arrives at the optimal solution by navigating the interior of the feasible region, from which originates its name [22, 26]. IPMs have emerged as a solution to the challenge of non-linear problems which revolutionised the optimisation world, offering an efficient alternative to traditional methods, such as the simplex algorithm, while granting a polynomial run time. In fact, Karmarkar himself claimed that the algorithm was 50-100 times more efficient than the simplex method [27]. In this section, the two most popular methods are presented, with the barrier method which represents the first polynomial time algorithm developed, and the primal-dual method which is the most efficient and majorly used method nowadays.

Barrier method

Considering the convex problem of the form of Equation 2.1, the idea behind this method is to convert the constrained optimisation problem into an unconstrained one in regards to the inequality constraints [22]. This is done with the use of a barrier function, $\boldsymbol{\varphi}(\boldsymbol{x})$, and a barrier parameter, $\zeta>0$, which are added to the objective function and act to remove the inequality constraint. The problem therefore becomes [22]

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimise}} \quad & \boldsymbol{f}(\boldsymbol{x}) + \zeta\boldsymbol{\varphi}(\boldsymbol{x})\,, \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\ \ . \end{aligned} \tag{2.5}$$

The solution of the problem $\boldsymbol{x}^{opt}(\zeta)$ will therefore differ from the solution of the original problem as the objective function has been perturbed by $\boldsymbol{\varphi}(\boldsymbol{x})$.

The logic behind the method is as follows, and can be seen in Algorithm 2.1. Starting from an initial value of ζ_0, the solution to Problem 2.5 is obtained using Newton's method, and the value of the barrier parameter is updated by scaling it by a factor $\kappa > 1$. Newton's method approximates $\boldsymbol{f}$ as a quadratic function using a second-order Taylor series expansion, from which a search direction is found to minimise said function [22]. The new solution to the problem is calculated, and the process continues until the barrier parameter is below a predefined value, ε. Solving Equation 2.5 is called centring as the solutions to the problem form the so-called central path [22].

Algorithm 2.1 Barrier interior point method [22].

```
1: for $i$ $in$ $i_{\max}$ do
2:     solve Problem 2.5 to find $\boldsymbol{x}^{opt}(\zeta_i)$
3:     $\boldsymbol{x}_i = \boldsymbol{x}^{opt}(\zeta_i)$
4:     if $\zeta_i < \varepsilon$, then
5:         break
6:     else
7:         $\zeta_{i+1} = \zeta_i/\kappa$
8:     end if
9: end for
```

The parameter i indicates the iteration of the optimisation process. Moreover, in order to solve the problem, the initial guess $\boldsymbol{x}_0$ needs to be strictly feasible, respecting the constraint $\boldsymbol{g}(\boldsymbol{x}) < \boldsymbol{0}$. The barrier function can take many forms and has a direct influence on various parameters of the algorithm, such as efficiency and convergence. The most widely used function in barrier methods is the logarithmic function [22]:

$$\boldsymbol{\varphi}(\boldsymbol{x}) = -\sum_{i=1}^{p} \ln(-\boldsymbol{g_i}(\boldsymbol{x})). \tag{2.6}$$

In order to obtain smooth convex optimisation methods, the barrier function needs to be convex and continuously differentiable, with the gradient and Hessian of the logarithmic barrier function defined as [22]:

$$\nabla \varphi(\boldsymbol{x}) = -\sum_{i=1}^{p} \frac{\nabla \boldsymbol{g}_i(\boldsymbol{x})}{\boldsymbol{g}_i(\boldsymbol{x})}, \tag{2.7}$$

$$\nabla^2 \varphi(\boldsymbol{x}) = \sum_{i=1}^{p} \frac{\nabla \boldsymbol{g}_i(\boldsymbol{x}) \nabla \boldsymbol{g}_i(\boldsymbol{x})^T}{\boldsymbol{g}_i(\boldsymbol{x})^2} - \frac{\nabla^2 \boldsymbol{g}_i(\boldsymbol{x})}{\boldsymbol{g}_i(\boldsymbol{x})}. \tag{2.8}$$

The main computational burden of this method is concentrated in the evaluation of the Newton direction required to solve the problem [22]. In order to speed up the process, most methods use direct methods like matrix factorisation to exploit the sparsity of the matrices and make the whole computation more time-efficient.

Primal-dual method

Primal-dual methods stand out as a more efficient alternative when compared to barrier methods. The root of their efficiency is their unique capability to accept initial iterates that might be infeasible, as the equality and inequality constraints must only be satisfied at the final solution. This characteristic is extremely significant as it mitigates the necessity for additional computations, and consequently, time, to transform an initially infeasible guess into a feasible one [22].

The concept behind primal-dual interior point methods is to solve the KKT conditions (Equation 2.4) by applying a modified Newton's method. Relaxed KKT conditions for primal-dual methods are imposed in order to follow the central path, defined by the points $(\boldsymbol{x},\, \boldsymbol{y},\, \boldsymbol{z},\, \boldsymbol{s})$:

$$\begin{cases} \nabla \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{A}^T \boldsymbol{y} + \boldsymbol{G}(\boldsymbol{x})^T \boldsymbol{z} = \boldsymbol{0} & (2.9a) \\ \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{0} & (2.9b) \\ \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{s} = 0 & (2.9c) \\ \boldsymbol{s}_i \boldsymbol{z}_i = \iota \quad \text{for } i = 1, ..., p & (2.9d) \\ (\boldsymbol{s}, \boldsymbol{z}) > \boldsymbol{0}\,. & (2.9e) \end{cases}$$

The main difference between the relaxed and nominal KKT conditions lies in the fact that $\boldsymbol{s}$ and $\boldsymbol{z}$ need to be positive instead of non-negative, and that in Equation 2.9d a scalar path parameter $\iota > 0$ is introduced.

Similar to the barrier method, the problem is solved iteratively, progressively decreasing the value of the parameter (in this case ι, whereas in the barrier method, ζ was reduced) to arrive at the optimal solution as the parameter tends to zero. In this circumstance, however, as previously mentioned the Newton step is applied using a different approach. Algorithm 2.2 shows the key principles behind the primal-dual methods, where h indicates the step length that is executed between one iterate to another [22]:

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + h_i \Delta \boldsymbol{x}_i \,, \tag{2.10}$$

and is determined through a merit function which is chosen to ensure convergence. The progress of the algorithm, instead, is measured with the duality measure χ, which represents the average value of the relaxed condition (Equation 2.9d) [22]:

$$\chi = \frac{\boldsymbol{s}^T \boldsymbol{z}}{p}. \tag{2.11}$$

In order to reduce the value of the duality measure substantially, and hence allow for faster convergence, the search direction needs to be composed of a centring part from the classical Newton direction along with the introduction of a new variable $\boldsymbol{\nu}$ [22]:

$$\begin{bmatrix} \boldsymbol{H}(\boldsymbol{x},\boldsymbol{z}) & \boldsymbol{A}^T & \boldsymbol{g}(x)^T & \boldsymbol{0} \\ \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{G}(\boldsymbol{x}) & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{S} & \boldsymbol{Z} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x} \\ \Delta \boldsymbol{y} \\ \Delta \boldsymbol{z} \\ \Delta \boldsymbol{s} \end{bmatrix} = \begin{bmatrix} \nabla \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{A}^T \boldsymbol{y} + \boldsymbol{G}(\boldsymbol{x})^T \boldsymbol{z} \\ \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \\ \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{s} \\ \boldsymbol{S}\boldsymbol{z} - \boldsymbol{\nu} \end{bmatrix}, \tag{2.12}$$

where solving the linear system gives the search direction $(\Delta \boldsymbol{x}, \Delta \boldsymbol{y}, \Delta \boldsymbol{z}, \Delta \boldsymbol{s})$, and the Hessian is obtained with

$$\boldsymbol{H}(\boldsymbol{x},\boldsymbol{z}) = \nabla^2 \boldsymbol{f}(\boldsymbol{x}) + \sum_{i=1}^{p} \boldsymbol{z}_i \nabla^2 \boldsymbol{g}_i(\boldsymbol{x}). \tag{2.13}$$

$\boldsymbol{S}$ and $\boldsymbol{Z}$, instead, represent the diagonal matrices of $\boldsymbol{s}$ and $\boldsymbol{z}$.

Algorithm 2.2 Primal-dual interior point method [22].

```
1: generate initial guesses $\boldsymbol{x}_0$, $\boldsymbol{y}_0$, $\boldsymbol{z}_0$, $\boldsymbol{s}_0 > 0$
2: set $\sigma \in (0, 1)$
3: for $i$ in $i_{\max}$ do
4:     compute duality measure $\chi_i = \boldsymbol{s}_i^T \boldsymbol{z}_i / p$
5:     solve Equation 2.12 to compute a search direction
6:     select $h_i$ such that $(\boldsymbol{s}_{i+1}, \boldsymbol{z}_{i+1}) > \mathbf{0}$
7:     $(\boldsymbol{x}_{i+1}, \boldsymbol{y}_{i+1}, \boldsymbol{z}_{i+1}, \boldsymbol{s}_{i+1}) = (\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i, \boldsymbol{s}_i) + h_i \Delta[\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i, \boldsymbol{s}_i](\sigma \chi_i)$
8: end for
```

The value of the parameter $\boldsymbol{\nu}$ has a direct effect on the type of search direction executed at the current iteration. In fact, a value of zero would indicate a pure Newton step that solves the nominal KKT conditions. A value of χ, instead, would represent a step primarily focused on centring the solution towards the central path. A compromise between the two extremes is required to build an effective algorithm, which results in the evaluation of the search parameter using:

$$\boldsymbol{\nu} = \sigma \chi \mathbf{1} \,, \tag{2.14}$$

where $\sigma \in (0, 1)$ represents the centring parameter.

As with the barrier method, there exist various variants of the primal-dual method, with the Mehrotra predictor-corrector method representing the most widely used in current commercial and open-source software [22]. The Mehrotra method, as the name states, takes the step using a combination of a predictor and a corrector [28]. The predictor step is taken using an affine-scaling direction, while the corrector accounts for the linearisation error of Equation 2.9d. The combination forms a unique step, which is determined by the value of the centring parameter:

$$\sigma = (\chi_{\text{aff}} / \chi)^3 \,, \tag{2.15}$$

where the subscript "aff" indicates the affine-scaling direction. As it can be deduced, in the cases where poor progress is made towards the solution, the value of σ would increase to guarantee a step which is closer to a pure centring direction. If, instead, the progress made towards the solution is adequate, the value of the centring parameter falls, resulting in a more Newton-like step.

2.1.2. First-Order Methods

The evolution of first-order methods went parallel to the rise of data science and machine learning, where the need for a fast solution is prioritised over high accuracy. In these optimisation problems, the quantity of input data is extremely high, with a large number of decision variables which causes a significant correlation between the computational burden and efficiency of a simulation and the level of accuracy required. Moreover, the randomness of the data supplied makes it illogical to try and optimise the problem below a certain threshold of the accuracy [29].

FOMs reach the optimal solution by using information of only first-order, which is generally focused on sub-gradients or gradients. The fact that they do not use second-order information has a direct effect on the performance of the solvers, which are avoided when a high level of accuracy is required. They are in fact used primarily in large-scale optimisation problems where low accuracy is needed, having the potential to necessitate a low computational complexity. Compared to IPMs and ASMs, these methods are simpler to implement in terms of the amount of source code needed to build a solver [30]. The drawback of these methods, however, is extremely significant as the behaviour of a solver is highly dependent on the problem in question [29]. This means that at times FOMs can be orders of magnitude faster compared to ASMs or IPMs, whereas, in other cases, they can take significantly longer or even result in infeasibilities. The two major methods used in modern FOM solvers are the gradient method and the splitting method.

Splitting Method

The splitting method is the underlying concept of the most widely used FOMs, like the Alternating Direction Method of Multipliers (ADMM). The idea behind this optimisation technique is that the objective function to minimise can be split into two functions, with the objective function of Equation 2.1 being transformed into [30]:

$$\begin{aligned} \underset{\boldsymbol{x},\boldsymbol{v}}{\text{minimise}} \quad & \boldsymbol{m}(\boldsymbol{x}) + \boldsymbol{n}(\boldsymbol{v})\,, \\ \text{s.t.} \quad & \boldsymbol{x} = \boldsymbol{v} \quad , \end{aligned} \tag{2.16}$$

where

$$\begin{aligned} \boldsymbol{m}(\boldsymbol{x}) &= \boldsymbol{f}(\boldsymbol{x}) + \delta(\boldsymbol{A}\boldsymbol{x} \,|\, \boldsymbol{b})\,, \\ \boldsymbol{n}(\boldsymbol{x}) &= \delta(\boldsymbol{x} \,|\, \boldsymbol{\Omega}) \quad . \end{aligned} \tag{2.17}$$

The term δ is an indicator function which takes a value of 0 in the instances in which the variable is in the convex set, and a value of ∞ otherwise.

The concept of the splitting method is that minimising the two functions $\boldsymbol{m}$ and $\boldsymbol{n}$ is faster and simpler than minimising the single function $\boldsymbol{f}$ directly. The minimisation of the newly formed functions can be executed using various methods, with one of the most general methods being the Proximal Method of Multipliers. Introducing the proximal augmented Lagrange function $\boldsymbol{L}$

$$\boldsymbol{L}(\boldsymbol{x},\, \boldsymbol{v}, \boldsymbol{\lambda}) = \boldsymbol{m}(\boldsymbol{x}) + \boldsymbol{n}(\boldsymbol{v}) + \boldsymbol{\lambda}^T(\boldsymbol{x} - \boldsymbol{v}) + \frac{\rho}{2}||\boldsymbol{x} - \boldsymbol{v}||^2 + \frac{1}{2}||\boldsymbol{x} - \boldsymbol{x}^{(i)}||^2_{\boldsymbol{P}_1} + \frac{1}{2}||\boldsymbol{v} - \boldsymbol{v}^{(i)}||^2_{\boldsymbol{P}_2}\,, \tag{2.18}$$

where ρ represents a positive penalty term, $\boldsymbol{\lambda}$ the Lagrange multiplier, and $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ are positive semidefinite matrices. The minimisation can be solved by adopting the following iterative equations:

$$\begin{aligned} (\boldsymbol{x}^{(i+1)}, \boldsymbol{v}^{(i+1)}) &\in \underset{\boldsymbol{x},\boldsymbol{v}}{\operatorname{argmin}} \;\; \boldsymbol{L}(\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{\lambda})\,, \\ \boldsymbol{\lambda}^{(i+1)} &= \boldsymbol{\lambda}^{(i)} + \rho(\boldsymbol{x}^{(i+1)} - \boldsymbol{v}^{(i+1)}) \;\; . \end{aligned} \tag{2.19}$$

Although minimising the proximal augmented Lagrange function over the $\boldsymbol{x}$ and $\boldsymbol{v}$ simultaneously can be as complicated as minimising the original problem, using an alternating method which minimises over the two variables can be much faster and efficient. This concept gives rise to some of the most famous and used methods such as the ADMM, the Douglas-Rachford Splitting Method, and the Chambolle-Pock scheme [30].

2.1.3. Active-Set Methods

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. Over the years, these solvers have stood the test of time, offering a distinctive advantage by drawing upon well-established methodologies. These types of methods result in being extremely attractive for QPs with a small number of parameters and variables, where the solution is competitive in terms of the computational burden required and the accuracy of the solution obtained. In fact, Ref. [18] performs an investigation on the efficiency of IPMs and ASMs with problem complexity, and finds that for problems where a low number of variables and constraints is small, ASMs perform better, whereas for larger problems the poor scalability of these methods is evident and limiting.

The underlying theory behind ASMs is relatively straightforward. If one poses the QP eliminating the inequality constraints, then the problem is formulated in a reduced space where only the equality constraints are present, which can be treated as solving a trivial linear set of equations. Moreover, it is understandable that in a simulation not all constraints are active at the solution. In the realm of ASMs, the concept of the "working set" plays a pivotal role. This set is comprised of the active constraints imposed to solve the problem and is updated at any given iteration, evolving dynamically throughout the optimisation process by the addition or removal of constraints as progress towards the optimal solution is made [30].

One of the major drawbacks of ASMs is that they do not provide a priori bounds on the number of iterations that are needed to find the optimal solution to a defined problem, which is a vital requirement for real-time systems [31–33]. Although recent studies have been carried out to try and overcome this deficit [34, 35], the certification on the exact bound on the maximum number of iterations and floating point operations is limited to state-of-the-art ASM solvers and to specific problems, and therefore cannot be applied to other types of problems without an extensive investigation.

2.1.4. Method Comparison

In order to compare theoretically the three major methods used in convex optimisation described in this section, one has to first introduce two major concepts: *warm-starting* and *matrix sparsity*. Both of these methods are used by some of the solvers in order to speed up the simulation and grant a faster convergence.

Warm-starting: Warm-starting is a technique used by FOMs and ASMs to speed up the optimisation of a given problem by providing as an initial guess for an iteration the solution of the previous iteration [30]. This serves to provide an initial starting point that is already close to the optimal solution granting a faster convergence and reducing the computations needed to reach optimality. Because of this nature, warm-starting is particularly useful in sequential programming [20], where a sequence of programs is solved and the process can be accelerated by building upon the previous solution. Moreover, this technique proves to be extremely advantageous for ASM solvers, as the solution of a previous problem is by definition comprised of a feasible set [21]. Providing a feasible set to the solver therefore eliminates the need for further computations required to make an infeasible set feasible.

Matrix sparsity: IPMs and FOMs are able to exploit the sparsity of matrices to allow for accelerated gradient computations and fast linear system solutions. The mostly zero entries in the matrices also allow for efficient storage and faster matrix operations such as matrix factorisation and matrix-vector multiplication [30].

Having introduced the different methods along with their benefits and drawbacks, and having talked about the two major techniques used by solvers to allow for more efficient optimisations, a table comparing the solver methods has been constructed and can be seen in Table 2.1 following a similar methodology as in Ref. [21]. A colour code is used where, as for convention, green is used for a positive outcome, 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 as in Ref. [21], respectively 6, 15, and 18 for small problems and 500, 3000, and 2000 for large problems.

Table 2.1: Comparison of convex solution methods.

	ASMs	**FOMs**	**IPMs**
Solver implementation	Medium	Easy	Hard
Speed (small problems)	Good	Medium	Medium
Speed (large problems)	Worst	Medium	Best
Consistency (speed)	Medium	Worst	Best
Accuracy (large problems)	Low	Medium	High
Memory usage	Medium	Good	Good

From this first table, it is clear that key differences exist among the solvers, and the most suitable one depends on the problem characteristics and level of accuracy required. Under a strictly implementable point of view, FOMs result in the most easily executable solvers as the algebra required is more straightforward, compared to the more sophisticated linear algebra used by ASMs and IPMs [4, 30].

The size of the problem, namely the second and third entry of the table, also seems to have an effect on the preferred method of choice when analysing the computational time required to reach convergence, with IPMs favoured when the number of variables and constraints is high [17]. For small problems, due to the fact that IPMs have a fixed cost when solving for the factorisation of the KKT matrix, the required computational time to find an optimal solution is higher, with ASMs representing the ideal choice [18].

Another characteristic that was compared is the ability of a solver to obtain the same solutions with respect to different instances of related problems (consistency), as this reflects the ability of a solver to obtain a solution for different trajectories. IPMs exhibit a favourable trend with an almost constant computational load for different instances, while FOMs result in being greatly affected by the problem at hand and ASMs fall in between these two categories [4, 30].

Moreover, as understandable, the level of accuracy obtained when solving an optimisation problem highly depends on the solver being used. The ASM solvers can provide results with low accuracy for problems with large sizes, while FOMs can guarantee low to medium accuracy [20, 29, 36], and for high-accuracy solutions the IPMs are preferred [17, 18]. The data memory required to solve the optimisation problem is seen to be worse for ASMs. This is because the number of linear equations solved is larger compared to IPMs and FOMs, and hence a greater amount of memory is needed to store the matrices used when solving these linear equations [18]. A second table (Table 2.2) highlights the technical differences between the three convex methods.

Table 2.2: Comparison of characteristics of convex solution methods.

	ASMs	**FOMs**	**IPMs**
Allows warm-starting	Yes	Yes	No
Exploits matrix sparsity	No	Yes	Yes
Infeasible acceptance	No	Yes	Yes

Regarding the techniques used by the solvers of the different families, it can be seen how ASMs and FOMs allow for a warm-starting of the process [21, 30], while FOMs and IPMs exploit the sparsity of the matrices in their calculations [30]. On the other hand, the ASMs are the only which require the initial guess to be feasible [17].

2.2. Convex Optimisation in Space Guidance

Optimisation engineering problems mostly require the minimisation or maximisation of an objective function, subject to some constraints and accuracy requirements. In guidance, these constraints are normally formed by the dynamics of the problem and some parameters that are related to the thrust or to the form of the control used. Over the last 15 years, the use of convex optimisation, where the problem is formulated in a convex manner through linearisation or convexification techniques, has become more popular

[37]. As a result, this has pushed the optimisation industry to create new efficient solvers and improve existing ones.

Applications of convex optimisation in spacecraft are not only interesting, but also represent a forced choice because of the tight times of some mission scenarios. One example lies in the problem involving a Mars precision landing with non-convex constraints on the magnitude of the thrust vector [38]. This investigation is intended to improve the landing precision of future Mars missions by orders of magnitude and enable the opportunity for Mars sample return missions. Ref. [39], instead, focuses on the formulation of the non-convex control constraints into a finite-dimensional convex optimisation problem, addressed by a second-order cone (SOC) problem. The particular scenario of a Mars entry, descent and landing phase represents a more general class of missions where a solution to an optimisation problem is required in a swift time due to the tight flight times of the mission.

Another precise landing scenario which has been investigated by NASA is the idea to develop an algorithm that is capable of autonomously designing the optimal powered descent trajectory onboard a spacecraft, immediately prior to the descent burn [40]. Compared to the aforementioned study, where the gravitational field of Mars can be easily estimated, a smaller and irregular body like that of an asteroid has a much more unpredictable gravitational field which cannot be approximated by a constant gravity. Ref. [41] takes the study to the next level by developing a second convex optimisation method to determine the optimal time of flight that yields the lowest propellant consumption over all flight times investigated.

In the realm of proximity operation, the actions required to reach a target vehicle have been formulated as a nonlinear optimal control problem, resulting in the solving of a SOC problem [42]. The dissertation of Ref. [43] specifically focuses on the use of SOCPs for autonomous trajectory planning in rendezvous and proximity operations.

In spacecraft trajectory planning, the optimisation of an Earth-to-Mars trajectory has been tested and verified by re-writing the problem in a convex formulation using convexification and linearisation techniques [11]. Ref. [9] builds on this investigation, applying a homotopic approach to obtain an increased performance by considering energy-to-fuel smoothing for the Earth-to-Dionysus bench case. A comparison of the different solutions obtained from indirect and convex methods is executed on three different target bodies (Mars, asteroid 67P and Dionysus), with the results showing better convergence for modified equinotical elements coordinates compared to spherical ones [5].

Convex optimisation problems, specifically SOC problems, have low computational complexity and can be efficiently solved within polynomial time. Algorithms like IPMs are capable of finding an optimal solution with a deterministic stopping criteria while maintaining a specified level of accuracy. This feature ensures that the global optimum can be computed to any desired accuracy, all within a determined upper bound on the number of iterations required for convergence. Consequently, numerical second-order cone programs (SOCPs) offer significant promise for performing real-time onboard computations, particularly in scenarios where time is of the essence.

2.2.1. Sequential Convex Programming

Figure 2.1 depicts the taxonomy of the optimisation problem families, with the complexity of the class increasing when moving from the inside to the outside. All of the classes can be seen to be in the non-linear program (NLP) family, with the convex optimisation family being formed at the most general level by the semidefinite program (SDP) class. This class is currently under high development and is extremely promising for robust design algorithms [4]. Next come the SOCPs which are currently the most advanced class that can be used to solve onboard space problems reliably and efficiently [4]. The smaller classes then include quadratically constrained quadratic programs (QCQPs), QPs and the simplest, linear programs (LPs).

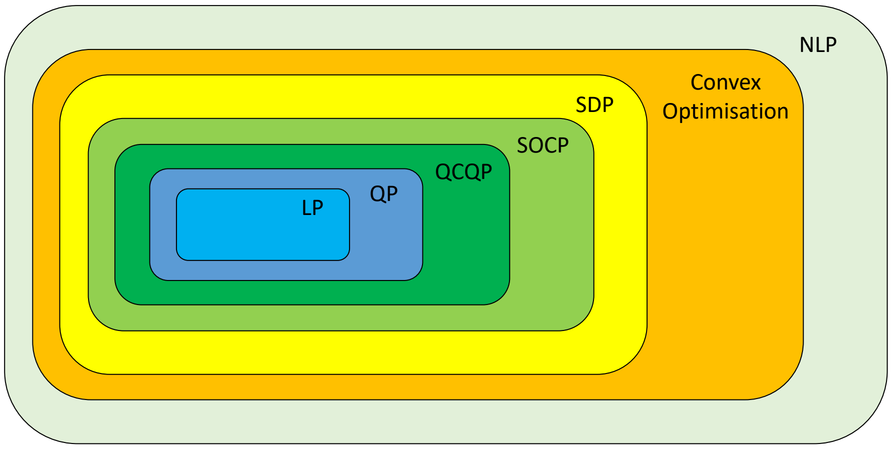


Figure 2.1: Taxonomy of optimisation technique families.

Convex optimisation can be seen to be capable of tackling a large number of problem families. In order to solve these optimisation problems, a popular choice is the use of a sequential convex program (SCP). At the core of this method lies the reformulation of an original optimisation problem in a convex subproblem using different techniques. 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. The initial guess does not need to respect the constraints and can be very far from the optimal solution of the problem [8]. Every SCP algorithm is composed of three major components, which include the starting step, the iteration step and the stopping step:

1. **Starting**: The first step of an SCP involves the generation of an initial guess.
2. **Iteration**: This step involves the most computational time and effort. The optimisation problem is in fact re-written as a convex subproblem, which is solved in a sequential manner using the initial guess generated in the first step. In order to obtain a solution, a convex optimisation solver must be used. The solution of each iteration is used to update the starting approximation of the next subproblem.
3. **Stopping**: At each iteration, a check is made to verify if certain stopping criteria are met. This can take many forms and can be based on the accuracy of the solution, the computational time, and/or the number of iterations.

The implementation of SCPs in space is renowned and it is being used in important applications like the SpaceX Starship landing manoeuvres and Falcon 9 trajectory optimisation [12]. Moreover, NASA and Blue Origin have recently started testing the potential use of SCPs onboard the New Shepard rocket [13, 44].

2.3. Survey of Available Software

This section deals with the state-of-the-art tools and solvers available to formulate a convex optimisation problem and solve it. The tools are programs that allow the formulation of an optimisation problem, while the solvers are focused on the resolution of the defined problem through an optimisation method. This investigation, along with Subsection 2.1.4 was used to determine which tools, software and methods showed potential and were of interest in the analysis.

2.3.1. Tools

FORCESPRO: FORCESPRO represents a competitive commercial tool used to generate optimisation solvers which can be customised based on the user requirements and can be deployed on embedded systems [45]. The ideal case of use is in real-time situations, where the data supplied to the solver varies and the optimisation problem needs to be solved several times. A custom optimisation solver in C-code is generated tailored to the problem constructed by the user, with data that can later be accessed and modified, allowing to solve multiple instances of the same problem[1].

ACADO: The ACADO Toolkit is a versatile commercial software environment comprising a collection of algorithms designed for automatic control and dynamic optimisation tasks [46]. This toolkit offers a comprehensive framework for employing an extensive range of algorithms, encompassing direct optimal control, model predictive control, state and parameter estimation, and robust optimisation [47]. ACADO Toolkit is crafted as self-contained C++ code and is complemented by an intuitive MATLAB interface. Its object-oriented architecture provides an efficient means of integrating external optimisation packages and allows users to expand its capabilities with custom-built optimisation routines. The choice of the solver in use spans from ASMs with qpOASES to IPMs with FORCES and HPMPC.

μAO-MOC: μAO-MOC is a freely available code generation tool for embedded linear model predictive control. The software is tailored to real-time embedded applications, with a particular effort in making the optimisation algorithm suitable and effective for micro-controller applications providing a low memory footprint and having deterministic execution time while adopting only additions and multiplications [48]. The software, written in Python, uses a QPs optimisation algorithm based on an augmented Lagrangian method combined with Nesterov's fast gradient method, but other QP solvers are supported.

FiOrdOs: FiOrdOs is a MATLAB toolbox that allows to specify problem parameters through MATLAB objects, providing a certification of the iteration count. It can be freely installed and used for automated C-code generation for first-order methods.

CVXPY: CVXPY is a powerful and user-friendly open-source optimisation framework in Python designed for solving convex optimisation problems [49]. This tool provides a high-level interface that allows users to specify optimisation problems in a simple and intuitive manner, rather than in the restrictive and technical form associated with optimisation solvers. Its ease of use and versatility in tackling different types of problems, along with the wide range of state-of-the-art solvers available to choose from (such as ECOS, SCS and MOSEK, presented in the following section), has allowed CVXPY to gain popularity in both academia and industry.

CVXPYgen: CVXPYgen is a tool that builds on CVXPY by transforming a family of convex optimisation problems, initially modelled using CVXPY, into a tailored solver implemented in C [19]. The resulting solver is uniquely designed for the specific prob-

[1]Available at: <`https://embotech.com/FORCES-Pro`> (last access on 06/04/2023)

lem family and is adaptable to accept various parameter values. Notably, this solver is well-suited for use in embedded systems, making it a valuable choice for optimising performance in such environments. The problems formulated need to satisfy two strict requirements: they need to be compliant with Disciplined Convex Programming (DCP) and Disciplined Parametrized Programming (DPP). For the problem to respect the DCP rules, only a certain amount of base functions with known curvature can be used [50]. DPP, instead, allows to define parameters that can be accessed and modified very quickly and efficiently. The reason behind this is because by defining the problem once and accessing the parameters at a later time to assign their values is much quicker than repeatedly solving a new problem [51]. The DPP formulation comes with its challenges, as for a system to be compliant it needs to satisfy the following rules [49]:

- all parameters, like variables, need to be classified as affine,
- the product of two expressions is affine when at least one of the expressions is constant, or when one is parameter-affine and the other is parameter-free.

A limit of CVXPYgen, however, is that contrarily to CVXPY the number of solvers supported, although extremely powerful, is much smaller. At the time of writing, the choice was limited to ECOS, SCS, and OSQP (another FOM solver, see Subsection 2.3.2).

2.3.2. Solvers

There exists a multitude of solvers for addressing optimisation problems, comprising a spectrum that ranges from commercially licensed to open-source alternatives, and from long-established algorithms to emerging, in-development solutions. To keep this section concise, only the most relevant solvers are reported, categorising them based on the optimisation method employed.

IPMs

ECOS: An open-source solver implemented in C. It adopts a standard primal-dual Mehrotra predictor-corrector method with Nesterov-Todd scaling and self-dual embedding, with search directions found via a symmetric indefinite KKT system. It results in being very competitive for medium-sized problems, which go up to tens of thousands of variables, however, the ideal size of problems to optimise is small, showing a faster behaviour compared to most existing SOCP solvers [52].

MOSEK: An optimisation suite which provides three different solvers based on the problem type: a simplex solver (for linear problems), an interior-point conic solver (for linear, nonlinear continuous, conic, and quadratic problems), and a mixed-integer solver (for problems with integer variables or disjunctive constraints)[2]. It is extremely suited for large-scale problems and allows to solve a multitude of problems which take different forms, including linear, conic, convex quadratic and quadratically constrained formulations. It also provides support for a variety of programming languages, which extend from C and Java to Julia and Rust.

NAG: Another optimisation suite which provides a vast collection of robust and tested solvers for discrete and continuous optimisation[3]. It allows the modification of existing models to vastly increase the efficiency, computational burden, and accuracy of the solution. The NAG library covers a range of different types of problems, including machine learning and numerical integration.

GUROBI: The Gurobi optimiser provides the choice between two types of algorithms: a barrier method algorithm and a simplex method algorithm[4]. The barrier method results in being the most suitable choice for large complex problems, granting a fast solution but showing a greater numerical sensitivity. The simplex method instead grants greater numerical stability at the expense of computational time. GUROBI also allows to use a concurrent optimiser, which runs various algorithms simultaneously and returns the optimal solution from the first converged algorithm.

CPLEX: Among the different solvers available, an IPM Barrier algorithm is implemented using a primal-dual predictor-corrector scheme which provides a valid alternative to the simplex method. This solver is generally preferred when tackling large problems or in cases in which the problems may exhibit issues related to numerical instability[5].

CVXOPT: A free software package for convex optimisation, which interfaces with Python and exploits the usage of dense and sparse matrices to find an optimal solution. Additionally, it provides interfaces to some external solvers such as MOSEK [53].

COPT: A high-performance mathematical programming solver that supports a large variety of problem formulations and is intended to support even more in the future [54]. It has interfaces which allow the solver to be run from many well-known programming languages and is intended to efficiently solve large-scale problems.

[2]Available at: <`https://www.mosek.com/documentation/`> (last access on 20/04/2023)
[3]Available at: <`https://support.nag.com/numeric/nl/`> (last access on 20/04/2023)
[4]Available at: <`https://www.gurobi.com/documentation/`> (last access on 20/04/2023)
[5]Available at: <`https://www.ibm.com/docs/en`> (last access on 20/04/2023)

qpSWIFT: An open-source optimisation solver developed in C that implements a primal-dual IPM with a Mehrotra predictor-corrector step and Nesterov-Todd scaling [55]. It was developed specifically for embedded and robotic QP applications.

CLARABEL: Unlike other IPM solvers, the Clarabel open-source solver handles a quadratic objective function without requiring any epigraphical reformulation, resulting in significant computational savings for problems with these objective functions [56]. It allows the solution of different types of problems, including LPs, QPs, SOCPs, and SDPs.

FOMs

SCS: A Douglas-Rachford splitting method solver applied to a homogeneous embedding of the quadratic cone program [57]. It is designed specifically for solving large-scale problems, representing the first ADMM-based solver available, and being used widely in the optimisation community.

OSQP: Developed by the University of Oxford, this solver uses an ADMM algorithm [58]. It also grants the user the possibility to generate compiled and tailored C code to obtain a fast and reliable solver for the given family of QPs. Using this compiled code, the user can access and change the problem data, but not its dimensions, between problem instances. A drawback of this solver is that it does not support second-order constraints.

ProxQP: Has its first targeted application is Robotics [59]. It has been extensively tested, showing the best reliability and performance on the toughest problems found in the literature. It is able to deal with dense, sparse or matrix-free problems.

COSMO: An accelerated ADMM-based solver for convex conic optimisation problems written in Julia which supports a variety of problem types, including LPs, QPs, SOCPs and SDPs [60]. It is well-suited for large-scale problems.

pipG: A novel primal-dual ADMM first-order method for conic optimisation [61]. pipG improves various parameters of IPMs, including the convergence rate of the duality gap and the convergence rate of constraint violations. However, it is still unclear whether it can outperform state-of-the-art second-order solvers like MOSEK.

ASMs

qpOASES: A freely available implementation of the recently proposed online active set strategy [62], which was inspired by important observations from the field of parametric QPs [63]. It has several theoretical features that make it well-suited for MPC applications.

daqp: An active-set solver developed for condensed real-time MPC problems, resulting in a competitive performance for small to medium-scale QPs and LPs [64]. For larger problems, the solver does not perform as well as other well-known and established solvers that exploit matrix sparsity.

HiGHS: A high-performance serial and parallel software that is freely available under the MIT license [65]. It is extremely suited for solving large-scale sparse LPs, mixed-integer programs and QP models.

quadprog: A convex quadratic solver that adopts the Goldfarb/Idnani numerically stable dual algorithm [66].

QPDAS: A solver which uses a dual active-set method. The performance of the solver results can be very efficient when the number of inequalities of the problem is small [67]. A drawback of the solver is that, unlike other ASMs, it is not possible to warm-start the problem and hence allow for a faster convergence of the solution.

Software Selection

Following the study and the comparison of the different methods, it was clear that ASMs represent a family with solvers that are less developed and less suitable for the context of onboard trajectory optimisation with respect to the FOMs and IPMs. While ASMs have demonstrated their robustness and efficiency over decades of practical application, they are not without their challenges. As mentioned in Subsection 2.1.3, their scalability becomes a concern for larger and more complex problems, rendering them less fitting for certain high-dimensional applications like interplanetary trajectory optimisation. Furthermore, their major drawback lies in the absence of a priori bounds on the complexity certification of the solvers which represents a significant limitation of the method for real-time applications. These factors have contributed to their declining usage in contemporary optimisation practices, and have therefore not been selected to be investigated in this study. In the wake of this decision, CVXPYgen resulted in being the ideal candidate for this investigation as it allowed to compare the state-of-the-art solvers for the IPMs and the FOMs. This software allowed to generate problem-specific solvers by incorporating the problem defined in a convex formulation in CVXPY with the open-source ECOS and SCS solvers.

2.4. Trade-off Criteria Selection

This section delves into the aspect of selecting the trade-off criteria needed for a fair and correct comparison of the different solvers. Effective decision-making in real-time and complex scenarios often hinges on the consideration of multiple factors. These selected trade-off criteria form the cornerstone of this study, guiding the comparative analysis of the solvers in Chapter 4 and helping distinguish the different methods in terms of performance and solution obtained. The selected parameters include:

1. Optimality;
2. Accuracy;
3. Reliability;
4. Efficiency;
5. Computational Burden;
6. Complexity;
7. Memory;
8. Effectiveness.

Optimality

In the context of decision-making, the term "optimality" is an extremely important criterion. In essence, optimality represents the ultimate objective of an optimisation process and is the first parameter that is analysed in any solution obtained. At its core is the value of the objective function needed to minimise (or maximise, depending on the nature of the problem). This criterion serves as a pivotal reference point, allowing to assess and compare the values obtained, given a desired level of accuracy, using different solvers. It is measured by the value of the objective function J, which is a reflection of the mass of the spacecraft at final time t_f. The lower the value of J, the higher the value of the final mass, and the more the solution produced is optimal.

Accuracy

The optimality alone is not sufficient to rank the solvers from worst to best. In fact, as various discretisation techniques can be used to transform problems from a continuous time to a discrete time, and hence be executable on computers, accuracy (ϵ) is indeed a fundamental aspect that can significantly impact the success of an endeavour. Once the optimal solution is obtained, its propagation using an interpolation method results in differences between the desired final solution and the obtained one. In the context of this study, accuracy refers to the precision obtained concerning critical parameters such as radial position (r), velocity (v), and spherical angles (θ, ϕ) with respect to the desired final values. Achieving a high level of accuracy in these variables is of paramount importance, as they often pertain to the success of a mission. A low accuracy can in fact lead to extensive additional fuel consumption to reach the desired state and, in some extreme cases, also mission failure.

Reliability

As in real-time decision-making scenarios, the optimisation algorithm is run several times, and the consistency of the solver to reach an optimal and feasible solution stands as a critical criterion in this field. This fundamental characteristic defines the dependability of the output of a solver, making it an indispensable consideration when selecting a solver. To evaluate this parameter as a percentage as in Equation 2.20, a series of simulations has to be run, and the number of non-converged solutions (failures) recorded:

$$R\,(\%) = \frac{\text{simulations} - \text{failures}}{\text{simulations}} \times 100\,. \tag{2.20}$$

Efficiency

Efficiency is quantified by the computational time required to solve a given problem per iteration (t_k). It serves as a pivotal metric for evaluating the computational performance and resource utilisation of various processes as this parameter describes how swiftly and economically a solver progresses towards the optimal solution. The value is calculated by taking a mean of the computational time required to solve each SCP iteration, until convergence at the final iteration k_f:

$$t_k = \frac{1}{k_f} \sum_{i=1}^{k_f} t_{k_i}\,. \tag{2.21}$$

Computational Burden

Another parameter that is crucial when evaluating any algorithm, especially for real-time problems where a fast response is needed, is the total allocated computational (CPU) time t_{CPU}. Compared to the efficiency, which represents the progress made towards a solution, this parameter is required to determine the overall computational time required to obtain convergence (Equation 2.22). It embodies the demand placed on computing resources, measuring the extent to which the solvers tax the available computational power. In decision-making scenarios, a swift response can provide significant overall savings for the mission and is hence a parameter to be considered when selecting a solver.

$$t_{\text{CPU}} = \sum_{i=1}^{k_f} t_{k_i} \, . \tag{2.22}$$

Complexity

Complexity is determined by the number of iterations required to attain an optimal converged solution. It delineates the intricacy of the path of a solution towards optimality and is of significance as it sheds light on the computational time and resource demands inherent to a given problem. This is not only represented by the number of SCP iterations, but also by the number of internal solver iterations, hereafter defined as "operations" to distinguish with the former.

Memory

Memory is characterised by both the memory consumption required during the solver execution and the overall size of the compiled code. This dual aspect of memory encapsulates the resource demands of a computational process and exerts a direct and crucial impact on real-life scenarios in which memory size and consumption are constrained. To study the memory usage during the execution of the SCP, the Resident Set Size (RSS) was measured, which represents the actual physical memory consumption.

Effectiveness

Effectiveness is characterised by the behaviour of an approach when faced with various problems. The study of effectiveness involves understanding how well a particular method adapts to different types of problems or different instances of the same problem. Since each problem formulation may differ from previously solved ones, all the aforementioned criteria may be influenced by the new problem formulation and therefore require evaluation. The adaptability of a solver lies at the core of many real-life implementations, as consistent performance is essential when encountering diverse problems.

3 | Methodology

The following chapter dives into the methodology adopted in the study, along with the techniques used to represent the strictly non-convex minimum fuel optimisation problem in a convex manner. Moreover, the logic behind the sequential convex program is explained, providing the convergence criteria which has been implemented and the main parameters applied. The chapter starts with Section 3.1 which describes the dynamical model in spherical coordinates adopted by the study along with the normalisation criteria used. Section 3.2 then defines the general optimal control problem of a fuel-optimal problem, with Section 3.3 reporting the techniques used to reformulate the problem in convex form, using methods such as linearisation, convexification and change of variables. The parameters used for the SCP algorithm are then outlined in Section 3.4 highlighting their importance in achieving convergence. The chapter then ends by reporting the whole problem concisely, along with the logic of the defined algorithm.

3.1. Dynamical Model

In this study, the spherical coordinate system is used to denote the state of the spacecraft. The convention adopted is shown in Figure 3.1a where r represents the distance from the central body to the spacecraft, θ the azimuth angle from the **x** axis, in the **x-y** plane, and ϕ the elevation angle from the **x-y** plane. The spherical coordinate system is powerful and versatile as it allows to extend the ability to pinpoint locations in three-dimensional space, especially in scenarios where cylindrical or spherical symmetries prevail.

In spherical coordinates, the state $\boldsymbol{x} \in \mathbb{R}^7$ of a spacecraft can be described as

$$\boldsymbol{x} = [\boldsymbol{r},\, \boldsymbol{v},\, m]^T = [r,\, \theta,\, \phi,\, v_r,\, v_\theta,\, v_\phi,\, m]^T\,, \tag{3.1}$$

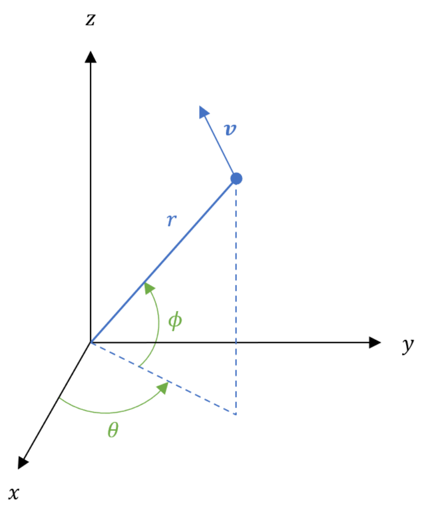
where m indicates the mass of the spacecraft. The control vector $\boldsymbol{u} \in \mathbb{R}^3$, instead, is

$$\boldsymbol{u} = [T,\, \alpha_r,\, \alpha_{\phi\theta}]^T\,, \tag{3.2}$$

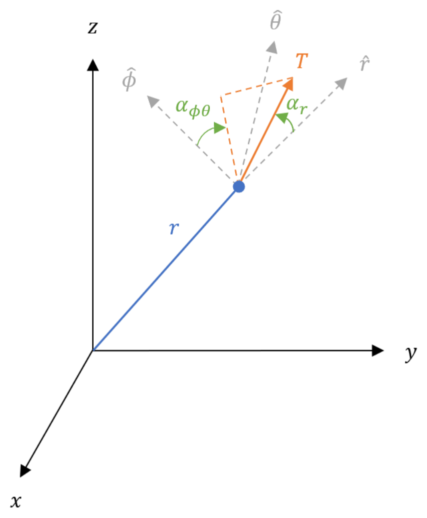
where T is the thrust magnitude. The angles α_r and $\alpha_{\phi\theta}$ represent how the thrust vector projects onto the reference frame adopted [11] and are illustrated in Figure 3.1b.

$$\alpha_r = \arccos\left(T/T_r\right), \tag{3.3a}$$

$$\alpha_{\phi\theta} = \arcsin\left(\frac{T_\theta}{T\sin\alpha_r}\right). \tag{3.3b}$$



(a) Coordinates.



(b) Control.

Figure 3.1: Spherical reference frame adopted.

The dynamics of a spacecraft can hence be represented in a compact state space formulation [11] that takes the form of

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}), \tag{3.4}$$

where the vector $\boldsymbol{f} \in \mathbb{R}^7$ represents the dynamics of the spacecraft dependent on both the state and the control.

3.1.1. Equations of Motion in Spherical Coordinates

The equations of motion represent a fundamental framework for describing the dynamics of an object in three-dimensional space. For a spherical system, they are as follows [11]:

$$\begin{cases} \dot{r} = v_r & \text{(3.5a)} \\ \dot{\theta} = \dfrac{v_\theta}{r\cos\phi} & \text{(3.5b)} \\ \dot{\phi} = \dfrac{v_\phi}{r} & \text{(3.5c)} \\ \dot{v}_r = \dfrac{v_\theta^2}{r} + \dfrac{v_\phi^2}{r} - \dfrac{\mu}{r^2} + \dfrac{T\cos\alpha_r}{m} & \text{(3.5d)} \\ \dot{v}_\theta = -\dfrac{v_\theta v_r}{r} + \dfrac{v_\phi v_\theta \tan\phi}{r} + \dfrac{T\sin\alpha_r \sin\alpha_{\phi\theta}}{m} & \text{(3.5e)} \\ \dot{v}_\phi = -\dfrac{v_\phi v_r}{r} - \dfrac{v_\theta^2 \tan\phi}{r} + \dfrac{T\sin\alpha_r \cos\alpha_{\phi\theta}}{m} & \text{(3.5f)} \\ \dot{m} = -\dfrac{T}{v_e} & \text{(3.5g)} \end{cases}$$

where v_e is the exhaust velocity. As notable in Equation 3.5d, Equation 3.5e, and Equation 3.5f the control is highly coupled with the state.

3.1.2. Normalisation

In order to allow the variables used in this study to have a similar magnitude, and hence allow for faster computations, a normalisation method is used. The values used for the normalisation of the position, velocity, time, thrust, and mass are reported in Table 3.1, along with the physical constants used in every simulation. As the angles in spherical coordinates are already in radians, there is no need to normalise them [11]. The value of the distance unit used is of exactly one astronomical unit (AU), from which the velocity unit is derived through $VU = \sqrt{\mu/DU}$, representing the velocity of the Earth if its orbit was circular. The time unit, instead, is obtained as $TU = DU/VU$. The implementation of this approach allowed the scaled gravitational constant in all simulations to take the value of one, simplifying the equations of motion.

Table 3.1: Physical constants.

Parameters	**Value**
Gravitational constant, μ (m^3/s^2)	1.32712440e+20
Gravitational acceleration, g_0 (m/s^2)	9.80665e+00
Distance unit, DU (m)	1.49597870e+11
Velocity unit, VU (m/s)	2.97846919e+04
Time unit, TU (s)	5.02264286e+06
Thrust unit, FU (N)	$T_{\max}$
Mass unit, MU (kg)	m_0

As can be seen from the table above, all the parameters reported are constant through simulations except for FU and MU, as they depend on the specifics of the spacecraft and the propulsion unit in use. Following the normalisation of the parameters, the equations of motion described in Equation 3.5 become

$$
\begin{cases}
\dot{r} = v_r & \text{(3.6a)} \\
\dot{\theta} = \dfrac{v_\theta}{r\cos\phi} & \text{(3.6b)} \\
\dot{\phi} = \dfrac{v_\phi}{r} & \text{(3.6c)} \\
\dot{v}_r = \dfrac{v_\theta^2}{r} + \dfrac{v_\phi^2}{r} - \dfrac{1}{r^2} + \dfrac{cT\cos\alpha_r}{m} & \text{(3.6d)} \\
\dot{v}_\theta = -\dfrac{v_\theta v_r}{r} + \dfrac{v_\phi v_\theta \tan\phi}{r} + \dfrac{cT\sin\alpha_r\sin\alpha_{\phi\theta}}{m} & \text{(3.6e)} \\
\dot{v}_\phi = -\dfrac{v_\phi v_r}{r} - \dfrac{v_\theta^2\tan\phi}{r} + \dfrac{cT\sin\alpha_r\cos\alpha_{\phi\theta}}{m} & \text{(3.6f)} \\
\dot{m} = -\dfrac{cT}{v_e}, & \text{(3.6g)}
\end{cases}
$$

where c takes the value of $T_{\max}DU/(m_0 VU^2)$, and the first three equations reported are unchanged compared to the previous formulation.

3.2. Optimal Control Problem

Taking the dynamics of the spacecraft as defined in Equation 3.4, and the state and control as Equation 3.1 and Equation 3.2 respectively, the minimum-fuel optimisation problem can be represented by a two-point boundary-value problem. The initial and final times (t_0 and t_f) of the problem are fixed along with the initial state $\boldsymbol{x}(t_0)$ and final state $\boldsymbol{x}(t_f)$. The final mass $m(t_f)$, however, is left free in order to perform a minimisation on it and obtain a fuel-optimal solution. The representation of the aforementioned boundary conditions (BCs) can be summarised as

$$\boldsymbol{x}(t_0) = [r(t_0),\ \theta(t_0),\ \phi(t_0),\ v_r(t_0),\ v_\theta(t_0),\ v_\phi(t_0),\ m(t_0)]^T = \boldsymbol{x}_0\,, \tag{3.7a}$$

$$\boldsymbol{x}(t_f) = [r(t_f),\ \theta(t_f),\ \phi(t_f),\ v_r(t_f),\ v_\theta(t_f),\ v_\phi(t_f)]^T = \boldsymbol{x}_f\,. \tag{3.7b}$$

Additional limits can be imposed on the problem by setting lower and upper bounds on the state and the controls to ensure that the solution is feasible [5, 11]:

$$\begin{bmatrix} 0.1 \\ 0 \\ -\pi \\ -10 \\ -10 \\ -10 \\ 0.1 \end{bmatrix} \leq \begin{bmatrix} r \\ \theta \\ \phi \\ v_r \\ v_\theta \\ v_\phi \\ m \end{bmatrix} \leq \begin{bmatrix} 10 \\ 10\pi \\ \pi \\ 10 \\ 10 \\ 10 \\ 1 \end{bmatrix}\,, \tag{3.8}$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \leq \begin{bmatrix} T \\ \alpha_r \\ \alpha_{\phi\theta} \end{bmatrix} \leq \begin{bmatrix} 1 \\ \pi \\ 2\pi \end{bmatrix}\,. \tag{3.9}$$

Although, theoretically, the lower bound of r should be zero, a value of 0.1 is chosen to prevent singularities from occurring. All other values are chosen following [11] and common sense. The thrust is bounded within the limit $0 \leq T \leq 1$ as the values are normalised by the maximum thrust [5]. The objective function J, instead, for fuel-optimal problems is defined in Mayer form as:

$$J = -m(t_f)\,, \tag{3.10}$$

where minimising the fuel consumption of a single-stage spacecraft is equivalent to maximising its final mass. The minimum-fuel problem can hence be posed as an optimal control problem with the form:

$$
\begin{aligned}
\underset{\boldsymbol{u}}{\text{minimise}} \quad & J = -m(t_f)\,, \\
\text{s.t.} \quad & \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})\,, \\
& \boldsymbol{x}(t_0) = \boldsymbol{x}_0\,, \\
& \boldsymbol{x}(t_f) = \boldsymbol{x}_f\,, \\
& \begin{bmatrix} 0.1 \\ 0 \\ -\pi \\ -10 \\ -10 \\ -10 \\ 0.1 \end{bmatrix} \leq \begin{bmatrix} r \\ \theta \\ \phi \\ v_r \\ v_\theta \\ v_\phi \\ m \end{bmatrix} \leq \begin{bmatrix} 10 \\ 10\pi \\ \pi \\ 10 \\ 10 \\ 10 \\ 1 \end{bmatrix}\,, \\
& \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \leq \begin{bmatrix} T \\ \alpha_r \\ \alpha_{\phi\theta} \end{bmatrix} \leq \begin{bmatrix} 1 \\ \pi \\ 2\pi \end{bmatrix}\,.
\end{aligned}
\tag{3.11}
$$

3.3. Successive Convexification Method

As previously stated in Subsection 3.1.1, the problem is highly non-linear, with the coupling of the control and states which can cause problems of convergence [10]. This section deals with the techniques and methods used to correctly formulate the dynamics in a linear manner and the problem in a convex form.

3.3.1. Change of Variables

The coupling of the state and the control in the dynamics can cause high-frequency jitters to emerge when a successive linearisation is applied [10]. A way around this problem is to apply a change of variables to define the original problem. Introducing the new control variable τ and the pseudo-mass variable z:

$$
\tau = T/m\,, \tag{3.12}
$$

$$
z = \ln m\,, \tag{3.13}
$$

allows to rewrite Equation 3.6g as

$$\dot{z} = -c\tau/v_e. \tag{3.14}$$

In order to ensure that the control is decoupled from the states, there is a need to introduce another three variables that represent the projection of τ onto the reference frame:

$$\tau_r = \tau \cos \alpha_r \,, \tag{3.15}$$

$$\tau_\theta = \tau \sin \alpha_r \sin \alpha_{\phi\theta} \,, \tag{3.16}$$

$$\tau_\phi = \tau \sin \alpha_r \cos \alpha_{\phi\theta} \,, \tag{3.17}$$

which follow the constraint

$$\tau_r^2 + \tau_\theta^2 + \tau_\phi^2 = \tau^2 \,. \tag{3.18}$$

Following the changes in the problem formulation, the limit of the control is transformed from Equation 3.9 to $0 \leq T \leq 1/m$, which generates the subsequent BCs [5, 11]:

$$0 \leq \tau \leq \exp(-z). \tag{3.19}$$

$$\begin{bmatrix} -10 \\ -10 \\ -10 \end{bmatrix} \leq \begin{bmatrix} \tau_r \\ \tau_\theta \\ \tau_\phi \end{bmatrix} \leq \begin{bmatrix} 10 \\ 10 \\ 10 \end{bmatrix} . \tag{3.20}$$

The BC of the pseudo mass, instead, can be written as:

$$\ln(0.1) \leq z \leq 0 \,, \tag{3.21}$$

with the bounds on the position and velocity unchanged from Equation 3.8. Using the definition presented in Equation 3.13, and applying the newly defined variables, the dynamics of the problem becomes

$$\begin{cases} \dot{r} = v_r & \text{(3.22a)} \\ \dot{\theta} = \dfrac{v_\theta}{r\cos\phi} & \text{(3.22b)} \\ \dot{\phi} = \dfrac{v_\phi}{r} & \text{(3.22c)} \\ \dot{v}_r = \dfrac{v_\theta^2}{r} + \dfrac{v_\phi^2}{r} - \dfrac{1}{r^2} + c\tau_r & \text{(3.22d)} \\ \dot{v}_\theta = -\dfrac{v_\theta v_r}{r} + \dfrac{v_\phi v_\theta \tan\phi}{r} + c\tau_\theta & \text{(3.22e)} \\ \dot{v}_\phi = -\dfrac{v_\phi v_r}{r} - \dfrac{v_\theta^2 \tan\phi}{r} + c\tau_\phi & \text{(3.22f)} \\ \dot{z} = -\dfrac{c\tau}{v_e}\,, & \text{(3.22g)} \end{cases}$$

which can be written as a state space formulation

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{B}\boldsymbol{u}\,, \tag{3.23}$$

with the matrix $\boldsymbol{B} \in \mathbb{R}^{7\times4}$ representing the control matrix, and the newly defined state vector ($\boldsymbol{x} \in \mathbb{R}^7$) and control vector ($\boldsymbol{u} \in \mathbb{R}^4$) are

$$\boldsymbol{x} = [r,\ \theta,\ \phi,\ v_r,\ v_\theta,\ v_\phi,\ z]^T\,, \tag{3.24}$$

$$\boldsymbol{u} = [\tau_r,\ \tau_\theta,\ \tau_\phi,\ \tau]^T. \tag{3.25}$$

The initial BC has to be updated in order to account for the pseudo-mass using $\boldsymbol{x}_0 = [r(t_0),\ \theta(t_0),\ \phi(t_0),\ v_r(t_0),\ v_\theta(t_0),\ v_\phi(t_0),\ z(t_0)]^T$.

The parameters $\boldsymbol{f}$ and $\boldsymbol{B}$ hence take the form of

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} v_r \\ v_\theta/(r\cos\phi) \\ v_\phi/r \\ v_\theta^2/r + v_\phi^2/r - 1/r^2 \\ -v_\theta v_r/r + v_\phi v_\theta \tan(\phi)/r \\ -v_\phi v_r/r - v_\theta^2 \tan(\phi)/r \\ 0 \end{bmatrix}\,, \tag{3.26}$$

$$\boldsymbol{B}(\boldsymbol{x}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ c & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & -c/v_e \end{bmatrix} . \tag{3.27}$$

The control, $\boldsymbol{u}$, is now decoupled from the state in the dynamics, removing the possibility of the aforementioned jitters and allowing for better convergence. It can also be noted that, once the problem parameters are defined, the $\boldsymbol{B}$ matrix remains unchanged throughout the simulation as both the exhaust velocity, v_e, and c are constants of the spacecraft. This means that the dependency of the parameter on $\boldsymbol{x}$ can be dropped.

The objective function of the optimal control problem presented in Equation 3.10 can also be redefined as

$$J = \int_{t_0}^{t_f} \tau(t)\, dt \,, \tag{3.28}$$

considering that the lower the total control cost, the lower the fuel consumption.

A drawback of the change of variables used is that the control constraints (Equation 3.18 and Equation 3.19) will need to be convexified, as they are now in a non-linear and non-convex form. This can be resolved and is addressed in the following subsection.

3.3.2. Convexification

Control

The problem formulation after the change of variables remains non-convex due to the constraints of Equation 3.18 and Equation 3.19. By applying a first-order Taylor series expansion to the upper bound of the latter, the constraint can be represented in a convex form, through the following condition [11]:

$$0 \leq \tau \leq \exp(-z^*)[1 - (z - z^*)] \,, \tag{3.29}$$

where z^* represents the reference time history of the pseudo-mass. The second constraint (Equation 3.18), instead, is non-convex as it represents the surface of a cone in three-dimensional space, where the interior of the cone is not included in the domain. In order to ensure that the interior is taken into account, and hence allow for a convex constraint to be used, a relaxation technique is adopted to expand the feasible set to

$$\tau_r^2 + \tau_\theta^2 + \tau_\phi^2 \leq \tau^2 \,. \tag{3.30}$$

Although the constraint in 3D space can be complicated to visualise, a representation in 2D space [11] can help explain the convexification process visually by setting one of the left-hand side parameters, in this case, τ_r, equal to zero as illustrated in Figure 3.2.

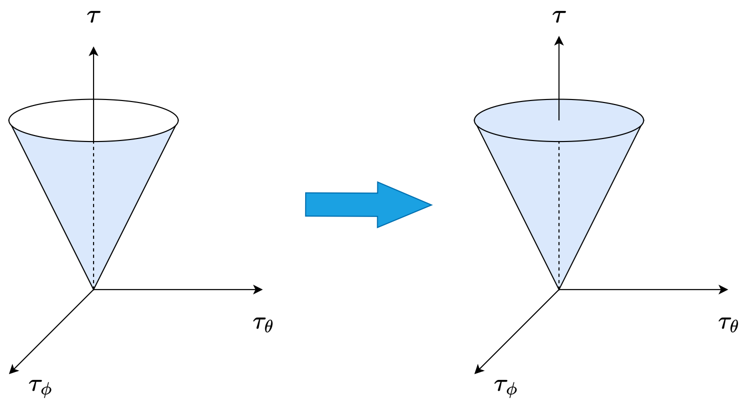


Figure 3.2: Control constraint in 2D space.

While imposing this constraint, a "bug" was found in the atomic function used to generate second-order constraints (`soc`), with errors arising when dealing with more than 100 constraints. A way around this problem has been found by rewriting these constraints in explicit form. These are all issues that the CVXPYgen community is currently working on, and therefore it is expected that in the future these limits will be addressed.

Dynamics

Having the constraints all in a linear or SOC form, the remaining part that needs to be linearised is the dynamics. The change of variables introduced in Subsection 3.3.1 allowed the control to be fully decoupled from the state, and hence the only term which remains nonlinear in the dynamics is $\boldsymbol{f}$. This term can be linearly approximated by introducing a reference trajectory $\boldsymbol{x}^*$:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}^*) + \boldsymbol{A}(\boldsymbol{x}^*)(\boldsymbol{x} - \boldsymbol{x}^*) + \boldsymbol{B}\boldsymbol{u} \,, \tag{3.31}$$

where $\boldsymbol{A} \in \mathbb{R}^{7\times 7}$ is the derivative of the dynamics with respect to the reference trajectory:

$$\boldsymbol{A}(\boldsymbol{x}^*) = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ -\dfrac{v_\theta}{r^2\cos\phi} & 0 & \dfrac{v_\theta \sin\phi}{r\cos^2\phi} & 0 & \dfrac{1}{r\cos\phi} & 0 & 0 \\ -\dfrac{v_\phi}{r^2} & 0 & 0 & 0 & 0 & \dfrac{1}{r} & 0 \\ \dfrac{2}{r^3} - \dfrac{v_\theta^2 + v_\phi^2}{r^2} & 0 & 0 & 0 & \dfrac{2v_\theta}{r} & \dfrac{2v_\phi}{r} & 0 \\ \dfrac{v_r v_\theta - v_\theta v_\phi \tan\phi}{r^2} & 0 & \dfrac{v_\theta v_\phi}{r\cos^2\phi} & -\dfrac{v_\theta}{r} & \dfrac{v_\phi \tan\phi - v_r}{r} & \dfrac{v_\theta \tan\phi}{r} & 0 \\ \dfrac{v_r v_\phi + v_\theta^2 \tan\phi}{r^2} & 0 & -\dfrac{v_\theta^2}{r\cos^2\phi} & -\dfrac{v_\phi}{r} & -\dfrac{2v_\theta \tan\phi}{r} & -\dfrac{v_r}{r} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} . \tag{3.32}$$

3.3.3. Discretisation

In order to solve the problem on a computer, the convex subproblem needs to be transcribed from continuous time to discrete time through the use of an integration method [4]. Ref. [1] focuses on analysing different discretisation techniques, covering the Adaptive Legendre-Gauss-Radau Pseudospectral Method, the Arbitrary-Order Legendre-Gauss-Lobatto (LGL) Method, and the First-Order-Hold (FOH) Method for three different test cases. The study finds the FOH method as being the most suitable for onboard low-thrust minimum-fuel trajectory optimisation, with LGL and pseudospectral methods presenting oscillations in the control which violate the bounds of the thrust. As the focus of this thesis is not on the discretisation methods, but on the performance of convex solvers, the trapezoidal integration scheme is used as in Ref. [11] due to its straightforwardness and simplicity in implementation [68].

The problem is therefore discretised evenly through time, forming a set of $N-1$ equality constraints, where N represents the number of nodes. The state differential equation can be written using the trapezoidal scheme as

$$\boldsymbol{x}_i - \boldsymbol{x}_{i-1} = \frac{\Delta t}{2}(\dot{\boldsymbol{x}}_i - \dot{\boldsymbol{x}}_{i-1})\,, \tag{3.33}$$

where i represents the node, taking values of $i = 2, ..., N$, and Δt indicates the time step $(t_i - t_{i-1})$ which is constant throughout the integration scheme. The trapezoidal representation, combined with the dynamics of Equation 3.31 forms the following constraint which is enforced at each node:

$$\begin{aligned}\boldsymbol{x}_i - \boldsymbol{x}_{i-1} = \ & \frac{\Delta t}{2}[\boldsymbol{f}(\boldsymbol{x}^*{}_i) + \boldsymbol{A}(\boldsymbol{x}^*{}_i)(\boldsymbol{x}_i - \boldsymbol{x}^*{}_i) + \boldsymbol{B}\boldsymbol{u}_i - ... \\ & \boldsymbol{f}(\boldsymbol{x}^*{}_{i-1}) - \boldsymbol{A}(\boldsymbol{x}^*{}_{i-1})(\boldsymbol{x}_{i-1} - \boldsymbol{x}^*{}_{i-1}) - \boldsymbol{B}\boldsymbol{u}_{i-1}].\end{aligned} \tag{3.34}$$

The cost function (Equation 3.28) also needs to be converted using the same trapezoidal scheme, for consistency, resulting in

$$J = \frac{\Delta t}{2}\sum_{i=2}^{N}(\tau_i + \tau_{i-1}), \tag{3.35}$$

which can be written in a more compact form as

$$J = \frac{\Delta t}{2}(\tau_{2:N} + \tau_{1:N-1}). \tag{3.36}$$

The original non-convex problem can hence be rewritten as a convex subproblem:

$$\begin{aligned}\underset{\boldsymbol{u}}{\text{minimise}} \quad & J = \frac{\Delta t}{2}(\tau_{2:N} + \tau_{1:N-1})\,, \\ \text{s.t.} \quad & \boldsymbol{x}_i - \boldsymbol{x}_{i-1} = \frac{\Delta t}{2}(\dot{\boldsymbol{x}}_i - \dot{\boldsymbol{x}}_{i-1}) \,\text{for}\;\; i = 2, ..., N \;\;, \\ & \boldsymbol{x}(t_0) = \boldsymbol{x}_0\,, \\ & \boldsymbol{x}(t_f) = \boldsymbol{x}_f\,, \\ & 0 \le \tau_i \le \exp(-z_i^*)[1 - (z_i - z_i^*)] \;\;\text{for}\;\; i = 1, ..., N\,, \\ & \tau_{ri}^2 + \tau_{\theta i}^2 + \tau_{\phi i}^2 \le \tau_i^2 \;\;\text{for}\, i = 1, ..., N \;\;, \\ & \begin{bmatrix} 0.1 \\ 0 \\ -\pi \\ -10 \\ -10 \\ -10 \\ \ln(0.1) \end{bmatrix} \le \begin{bmatrix} r \\ \theta \\ \phi \\ v_r \\ v_\theta \\ v_\phi \\ z \end{bmatrix} \le \begin{bmatrix} 10 \\ 10\pi \\ \pi \\ 10 \\ 10 \\ 10 \\ 0 \end{bmatrix}\;, \\ & \begin{bmatrix} -10 \\ -10 \\ -10 \end{bmatrix} \le \begin{bmatrix} \tau_r \\ \tau_\theta \\ \tau_\phi \end{bmatrix} \le \begin{bmatrix} 10 \\ 10 \\ 10 \end{bmatrix}\;.\end{aligned} \tag{3.37}$$

Due to the linearisation of the dynamics the solution of Equation 3.37, along with the lower and upper limits, does not necessarily grant an optimal solution to the minimum-fuel optimisation problem [5]. In order to overcome this issue, the convex subproblem is solved sequentially through an SCP, updating the reference trajectory at every iteration with the optimal solution of the previous step until a predefined tolerance is reached.

3.4. Sequential Convex Programming Algorithm

The equations and constraints defined in the previous section are not sufficient alone to generate a state-of-the-art SCP. In fact, in order to allow for better convergence of the algorithm some methods can be used, like the introduction of slack variables or a trust region. Moreover, a convergence criterion needs to be imposed to allow the algorithm to terminate when a certain tolerance in the solution is achieved. The following subsections will cover these aspects, presenting the methods adopted in the algorithm.

3.4.1. Artificial Infeasibility

The problem of artificial infeasibility arises when the solution of the convex subproblem obtained from the SCP process defined in Subsection 2.2.1 results infeasible at times in which the original problem is theoretically solvable [8, 37, 69]. This is caused by the linearisation of the dynamics, and can be solved through the introduction of slack variables [5]. In the case of the dynamics, $\boldsymbol{h} \in \mathbb{R}^7$ transforms the equation into

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}^*) + \boldsymbol{A}(\boldsymbol{x}^*)(\boldsymbol{x} - \boldsymbol{x}^*) + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{h}. \tag{3.38}$$

As a linearisation is executed also on the control, the same procedure needs to be applied to Equation 3.29 by introducing the slack variable $\eta \in \mathbb{R}$ at each node [9]:

$$0 \leq \tau \leq \exp(-z^*)[1 - (z - z^*)] + \eta. \tag{3.39}$$

The slack variables must also be taken into account in the cost function as

$$J = \frac{\Delta t}{2}(\tau_{2:N} + \tau_{1:N-1}) + C \sum_{j=1}^{7} \sum_{i=1}^{N} |\boldsymbol{h}_{j,i}| + D \sum_{i=1}^{N} \max(0, \eta_i), \tag{3.40}$$

where $C > 0$ and $D > 0$ are weight factors of the slack variables, both chosen to be 100, which act as a penalising term in the cost function [1]. It is therefore understandable that, at the end of the optimisation process, the values of the slack variables $\boldsymbol{h}$ and η must be near zero in order to satisfy the constraints and yield an optimal result.

3.4.2. Trust Region

For SCP algorithms a trust region has been demonstrated to greatly improve convergence by keeping the solution in the neighbourhood of the reference trajectory [1, 5, 9, 11]. A study has been carried out by Ref. [1] to analyse the effect that different trust regions have on the convergence and optimal solution obtained for fuel-optimal trajectories. The findings, which cover a variety of hard and soft trust regions, concluded that all methods yield similar results but the soft trust region required more iterations to solve the SOC constraint, leading to a greater computational time. For this study, a constant trust region, $\boldsymbol{\delta}$, has been applied for the first SCP iteration.

$$\left\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\right\|_\infty \leq \boldsymbol{\delta}, \tag{3.41}$$

where the superscript k represents the iteration of the SCP algorithm. This constraint is applied when $k = 1$ and the only information available for the reference trajectories is the initial guess $\boldsymbol{x}^{(0)}$, with $\boldsymbol{\delta}$ taking the value of

$$\boldsymbol{\delta} = [1,\, 5\pi,\, 10\pi,\, 0.4,\, 0.4,\, 0.4,\, 10]^T\,. \tag{3.42}$$

As the initial generated reference trajectory $\boldsymbol{x}^{(0)}$ holds no information on the pseudo-mass z, the limit imposed on this variable was selected to be of a high magnitude in order to mitigate a restriction on the convergence of this parameter. The remaining values of $\boldsymbol{\delta}$ can be seen to be relatively high if it is considered that the variables are normalised. This is done to account for the possibility that the initial reference trajectory supplied to the solver might be far from the optimal one, and therefore the output from the first SCP iteration might diverge drastically from $\boldsymbol{x}^{(0)}$.

When the algorithm enters the second iteration (i.e. for $k > 1$), the information stored includes two reference trajectories, allowing for a Cauchy sequence to be applied [11] to guarantee a better convergence.

$$\left\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\right\|_\infty \leq \gamma \left\|\boldsymbol{x}^{(k-1)} - \boldsymbol{x}^{(k-2)}\right\|_\infty, \tag{3.43}$$

where the trust region factor $\gamma \in (0, 1)$ is selected in accordance with the desired level of convergence, as indicated by Ref. [5]. The choice of this factor bears a direct impact on the optimisation process. While a lower value of γ can potentially allow for faster convergence, this accelerated progress might come at the cost of introducing infeasibilities into the optimisation. Conversely, a higher value of γ can increase the likelihood of successful convergence at the expense of a longer optimisation process, potentially leading to a computational burden that outweighs the gains.

3.4.3. Convergence Criteria

As stated in Subsection 3.3.3 an SCP algorithm continues to sequentially solve the optimisation problem until a certain, predefined, tolerance is achieved. The choice of the method used for convergence is trivial to ensure that the solution obtained is in fact optimal, and reflects the desired level of accuracy.

Like the trust region, the selection of the convergence technique can take various forms. In this paper, the process reported in Ref. [1] is adopted which focuses on the maximum constraint violation (ε_c) and the relative change of final pseudo-mass (ε_z) between the latest and previous SCP iteration. These values enforce the following stopping criteria:

$$\varepsilon_c \leq 10^{-6}\,, \tag{3.44}$$

$$\varepsilon_z = \left| z^{(k)}(t_f) - z^{(k-1)}(t_f) \right| \leq 10^{-4}\,, \tag{3.45}$$

where ε_c is found by recording the maximum constraint violation of the problem defined through CVXPY (`prob.constraint.violation`). An optimal solution is therefore found if both constraints are respected. The values of the pseudo-mass convergence and constraint violation were selected in accordance with Ref. [1].

In order to prevent the algorithm from running endlessly when an optimal solution is obtained without satisfying the convergence criteria, an additional check is implemented to monitor the progress of the solution across successive SCP iterations, as adopted in Ref. [1]. The execution of the algorithm is therefore stopped if the relative difference in solution of the state, ε_x, falls below a predefined threshold:

$$\varepsilon_x = \left\| \boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)} \right\|_\infty \leq 10^{-7}\,, \tag{3.46}$$

or if the number of SCP iterations, k, exceeds a predefined limit ($k_{\max}$). In this study, the maximum number of iterations is set to 50 following initial analyses which showed convergence at a value far below that chosen for $k_{\max}$.

3.4.4. Initial Guess Generation

As there is a lack of information on the reference trajectory for the initial iteration of the SCP algorithm, a guess has to be generated. Describing the position of the spacecraft using a third-order polynomial, where

$$\boldsymbol{r} = \boldsymbol{a}t^3 + \boldsymbol{b}t^2 + \boldsymbol{c}t + \boldsymbol{d}\,, \tag{3.47}$$

and

$$\boldsymbol{v} = 3\boldsymbol{a}t^2 + 2\boldsymbol{b}t + \boldsymbol{c}\,, \tag{3.48}$$

one can find the constants of the equation ($\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, $\boldsymbol{d}$) using the initial and final boundary conditions. In fact, through a correct substitution, the constants take the values of

$$\begin{cases} \boldsymbol{a} = \dfrac{2(\boldsymbol{r}_0 - \boldsymbol{r}_f)}{t_f^3} + \dfrac{\boldsymbol{v}_f + \boldsymbol{v}_0}{t_f^2}\,, & (3.49a) \\ \boldsymbol{b} = \dfrac{\boldsymbol{v}_f - \boldsymbol{v}_0}{2t_f} - \dfrac{3\boldsymbol{a}t_f}{2}\,, & (3.49b) \\ \boldsymbol{c} = \boldsymbol{v}_0\,, & (3.49c) \\ \boldsymbol{d} = \boldsymbol{r}_0\,. & (3.49d) \end{cases}$$

Equation 3.47 and Equation 3.48 are therefore used, along with the problem bounds, to generate a reference trajectory for the first SCP iteration, where the time is discretised in $N-1$ uniform time steps ranging from t_0 to t_f. No reference was generated for the mass of the spacecraft, leaving the parameter unchanged.

3.4.5. Algorithm

Combining all the criteria, techniques and bounds discussed in this section, the convex subproblem takes the following form:

$$
\begin{aligned}
\underset{\boldsymbol{u},\boldsymbol{h},\eta}{\text{minimise}} \quad & J = \frac{\Delta t}{2}(\tau_{2:N} + \tau_{1:N-1}) + C\sum_{j=1}^{7}\sum_{i=1}^{N}|\boldsymbol{h}_{j,i}| + D\sum_{i=1}^{N}\max(0,\eta_i)\,, \\
\text{s.t.} \quad & \boldsymbol{x}_i - \boldsymbol{x}_{i-1} = \frac{\Delta t}{2}(\dot{\boldsymbol{x}}_i - \dot{\boldsymbol{x}}_{i-1}) \;\; \text{for}\, i = 2, ..., N\,, \\
& \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}^*) + \boldsymbol{A}(\boldsymbol{x}^*)(\boldsymbol{x} - \boldsymbol{x}^*) + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{h}\,, \\
& \boldsymbol{x}(t_0) = \boldsymbol{x}_0\,, \\
& \boldsymbol{x}(t_f) = \boldsymbol{x}_f\,, \\
& 0 \le \tau_i \le \exp(-z_i^*)[1 - (z_i - z_i^*)] + \eta_i \;\; \text{for}\, i = 1, ..., N\,, \\
& \tau_{ri}^2 + \tau_{\theta i}^2 + \tau_{\phi i}^2 \le \tau_i^2 \;\; \text{for}\, i = 1, ..., N\,, \\
& \left\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\right\|_\infty \le \boldsymbol{\delta} \quad \text{for}\, k = 1\,, \\
& \left\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\right\|_\infty \le \gamma\left\|\boldsymbol{x}^{(k-1)} - \boldsymbol{x}^{(k-2)}\right\|_\infty \;\; \text{for}\, k > 1\,, \\
& \begin{bmatrix} 0.1 \\ 0 \\ -\pi \\ -10 \\ -10 \\ -10 \\ \ln(0.1) \end{bmatrix} \le \begin{bmatrix} r \\ \theta \\ \phi \\ v_r \\ v_\theta \\ v_\phi \\ z \end{bmatrix} \le \begin{bmatrix} 10 \\ 10\pi \\ \pi \\ 10 \\ 10 \\ 10 \\ 0 \end{bmatrix}\,, \\
& \begin{bmatrix} -10 \\ -10 \\ -10 \end{bmatrix} \le \begin{bmatrix} \tau_r \\ \tau_\theta \\ \tau_\phi \end{bmatrix} \le \begin{bmatrix} 10 \\ 10 \\ 10 \end{bmatrix}\,.
\end{aligned}
\tag{3.50}
$$

As reported in Subsection 2.2.1, an SCP algorithm is necessary in order to solve a series of subproblems of the original optimisation problem and progress to a solution, until predefined convergence criteria are met. Algorithm 3.1 displays the logic behind the SCP algorithm adopted in this study, with the problem and criteria outlined in this section.

Algorithm 3.1 Sequential Programming Algorithm

```
1: generate a reference trajectory $\boldsymbol{x}^{(0)}$
2: for $k$ in $k_{\max}$ do
3:     $\boldsymbol{x}^* = \boldsymbol{x}^{(k-1)}$
4:     solve Problem 3.50 to find $\boldsymbol{x}^{(k)}$ and $\boldsymbol{u}^{(k)}$
5:     if $\varepsilon_c \leq 10^{-6}$ & $\varepsilon_z \leq 10^{-4}$ then
6:         optimal solution found: $\boldsymbol{x}^{opt} = \boldsymbol{x}^{(k)}$
7:         $\boldsymbol{u}^{opt} = \boldsymbol{u}^{(k)}$
8:         break
9:     else if $\varepsilon_x \leq 10^{-7}$ then
10:        progress is too small: break
11:    end if
12: end for
```

DPP compliance

In order to allow the problem to be formulated according to DPP convention, as stated in the CVXPYgen introduction of Subsection 2.3.1, certain rules have to be satisfied, and are reported here for reference:

- all parameters, like variables, need to be classified as affine,
- the product of two expressions is affine when at least one of the expressions is constant, or when one of the expressions is parameter-affine and the other is parameter-free.

This allows to define parameters which can be modified without reconstructing the entire problem, significantly speeding up the solution of the SCP as updating the values can be much faster than repeatedly solving a newly defined problem.

As can be seen from the requirements, some of the constraints imposed in the problem formulation need to be redefined to prevent two variables from being allocated in the same expression. This brought to the introduction of a new variable, $\Delta\boldsymbol{x}$, which represents the difference between the current state and reference state:

$$\Delta\boldsymbol{x} = \boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}, \tag{3.51}$$

and the definition of parameters $\boldsymbol{f}$, $\boldsymbol{A}$, $\boldsymbol{x}^*$ to update with the iteration-specific values.

Moreover, in order to impose the convergence of the algorithm and allow it to pass from the defined trust region $\boldsymbol{\delta}$ to a Cauchy sequence, the trust region condition was updated from Equation 3.41 and Equation 3.43 to:

$$\left\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\right\|_{\infty} \leq \boldsymbol{\Upsilon}\,, \tag{3.52}$$

where the newly defined parameter $\boldsymbol{\Upsilon}$ took a value of:

$$\boldsymbol{\Upsilon} = \begin{cases} \boldsymbol{\delta}\,, & \text{if } k = 1 \\ \gamma\left\|\boldsymbol{x}^{(k-1)} - \boldsymbol{x}^{(k-2)}\right\|_{\infty}\,, & \text{otherwise}\,. \end{cases} \tag{3.53}$$

Lastly, another parameter needed to be defined in order to prevent the multiplication of two parametrized expressions, which goes against DPP rules [49]. This was defined as $\exp_z$, which represented the exponential of the reference pseudo-mass of Equation 3.39:

$$\exp_z = \exp(-z^*) \tag{3.54}$$

4 | Numerical Results

This chapter presents the results obtained using the proposed setup and their analysis. Section 4.1 first introduces the different test cases used, explaining their importance and reintroducing the criteria used for the analysis. Section 4.2 then discusses the important choice of parameters such as the value of the trust region factor for each test case and the limit of operations imposed on each solver. A form of validation is given by comparing the results obtained using the proposed algorithm and the ones from literature in Section 4.3. Section 4.4 then provides a detailed analysis of the simulation results corresponding to the evaluation criteria outlined in Section 2.4. Section 4.5 concludes the chapter by analysing the performance of the available IPM solvers in CVXPY and comparing the results obtained from the compiled and non-compiled codes.

The simulations of all test cases 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. To ensure sufficient data collection and mitigate the risk of introducing bias, a Monte Carlo (MC) simulation was employed. This simulation enclosed a total of 100 runs and incorporated a standard deviation of 10% applied to the final position, $\boldsymbol{x}_f$, in order to generate an initial perturbed reference trajectory. While the trajectory produced through this method 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. This strategic approach grants to scrutinise the abilities of the solvers under demanding conditions and draw meaningful insights about their efficiency and limitations.

The reported results represent the median across all MC samples, and, when present, error bars indicate the lower quartile and upper quartile. The simulation setup was carried out using CVXPYgen v0.2.3[1] in conjunction with Python v3.9[2]. The problem formulation utilised CVXPY v1.3[3], and ECOS v2.0.1[4] and SCS v3.2.3[5] were employed to generate the compiled solvers. To ensure that both solvers leveraged the strengths of their respective

[1] Available at: <`https://github.com/cvxgrp/cvxpygen`> (last accessed on 12/10/2023)
[2] Available at: <`https://www.python.org/downloads/`> (last accessed on 04/05/2023)
[3] Available at: <`https://www.cvxpy.org/`> (last accessed on 12/10/2023)
[4] Available at: <`https://github.com/embotech/ecos`> (last accessed on 20/09/2023)
[5] Available at: <`https://github.com/cvxgrp/scs`> (last accessed on 20/09/2023)

method family (see Subsection 2.1.4), the option to use warm starting was enabled for SCS, while ECOS automatically exploited matrix sparsity.

4.1. Experimental Setup

Adopting the SCP scheme presented in Algorithm 3.1, a MC analysis was carried out in GitLab (see Appendix C) where three test cases were used to compare the performance of the solvers. The three scenarios were taken to have an increasing complexity to try and push to the edge the performance of the solvers. These problems include an Earth-to-Mars transfer with BCs taken from [11], a CubeSat transfer from Sun–Earth Lagrange point L_2 to near-Earth asteroid 2000 SG344 (SEL$_2$–NEA) taken from [1], and finally an Earth-to-Venus transfer, also taken from [1]. The simulation parameters for each of the three problems are given in Table 4.1, with the BCs for each case, in spherical coordinates, reported in Table 4.2.

Table 4.1: Simulation parameters for the test cases.

Parameters	**Earth–Mars**	**SEL$_2$–NEA**	**Earth–Venus**
Time of Flight (days)	253	700	1000
Initial Mass (kg)	659.3	22.6	1500
Maximum Thrust (N)	0.55	2.25e-03	0.33
Specific Impulse (s)	3300	3067	3800
Number of Nodes (-)	100	125	150

The number of revolutions for the initial guess generation (Subsection 3.4.4) were taken as in literature to be of 0 for the Earth–Mars case, 2 for the SEL$_2$–NEA transfer and 3 for the Earth–Venus case. The choice of the test cases was made not only to increase the complexity and push the performance of the methods, but also to represent a source of validation for the obtained output and the proposed algorithm.

The first case involves the Earth–Mars trajectory, chosen for its simplicity, featuring a short time of flight and a single revolution. This case acts as a foundational benchmark, enabling a straightforward assessment of the performance of the optimisation algorithm for interplanetary transfers.

Moving beyond the straightforwardness of the Earth–Mars case, the second scenario concerns a SEL$_2$–NEA CubeSat transfer similar to the M-ARGO mission analysis of Ref.

[70]. This scenario introduces more complexity and brings diverse initial and final positions when compared to planet-to-planet transfers. The SEL_2–NEA transfer provides a more comprehensive test bed for evaluating the versatility and adaptability of the solvers. What makes this case harder compared to the Earth–Mars trajectory is the extended time of flight of the transfer combined with the low thrust levels imposed by the limited dimensions of the spacecraft.

Finally, the study extends to the Earth–Venus problem, representing the most complex scenario simulated in the study. With a prolonged time of flight and a higher number of revolutions, the Earth–Venus scenario is aimed to challenge the capacities of the solvers to handle interplanetary missions with extended trajectories, further gauging their effectiveness in real-world applications.

It is noteworthy to mention that at the time of writing CVXPYgen presents a constraint on the number of parameters that can be employed, imposing a maximum allowance of 150 nodes for the minimum-fuel problem presented. In light of this, the number of nodes used for the three different simulations are of 100, 125, and 150, chosen to reflect the increasing times of flight of the transfers.

Table 4.2: Boundary conditions in non-dimensional units.

	Earth–Mars		**SEL_2–NEA**		**Earth–Venus**	
	Initial	Final	Initial	Final	Initial	Final
r (DU)	1.0000e+00	1.5237e+00	9.9568e-01	9.2843e-01	9.9948e-01	7.1860e-01
θ (rad)	0.0000e+00	3.1416e+00	2.3531e+00	1.3670e+01	2.4001e-01	2.0894e+01
ϕ (rad)	0.0000e+00	0.0323e+00	-3.5264e-05	-1.5444e-03	-1.6719e-06	3.8500e-02
v_r (VU)	0.0000e+00	0.0000e+00	8.8856e-03	-4.7179e-02	-1.6986e-02	-1.9924e-03
v_θ (VU)	1.0000e+00	0.8101e+00	4.4338e-05	1.0625e+00	1.0014e+00	1.1831e+00
v_ϕ (VU)	0.0000e+00	0.0000e+00	0.0000e+00	-1.3067e-03	1.5011e-05	5.3326e-02

The results presented cover the analysis of the criteria introduced in Section 2.4, which are reported for completeness in Table 4.3 with a more detailed description.

Table 4.3: Description of criteria used.

Criteria	Measure
Memory	Determined by analysing the size of the compiled code and the memory usage during the execution of the SCP.
Complexity	Represents the total number of iterations and operations to reach a converged solution, and is measured by using the `SolverStats`[6] object of a CVXPY problem.
Efficiency	Describes the computational performance of the solver: $$t_k = \frac{1}{k_f}\sum_{i=1}^{k_f} t_{k_i}\,. \tag{4.1}$$
Computational Burden	Indicates the overall CPU time required to obtain convergence: $$t_{\text{CPU}} = \sum_{i=1}^{k_f} t_{k_i}\,. \tag{4.2}$$
Accuracy	Outlines the error between the re-propagated optimal solution, $\tilde{\boldsymbol{x}}$, and the desired state at final time: $$\epsilon = \left\|\tilde{\boldsymbol{x}}(t_f) - \boldsymbol{x}_f\right\|\,. \tag{4.3}$$
Optimality	Measured by determining the value of the objective function that is being minimised: $$J = \frac{\Delta t}{2}(\tau_{2:N} + \tau_{1:N-1}) + C\sum_{j=1}^{7}\sum_{i=1}^{N}\left\|\boldsymbol{h}_{j,i}\right\| + D\sum_{i=1}^{N}\max(0,\eta_i)\,. \tag{4.4}$$

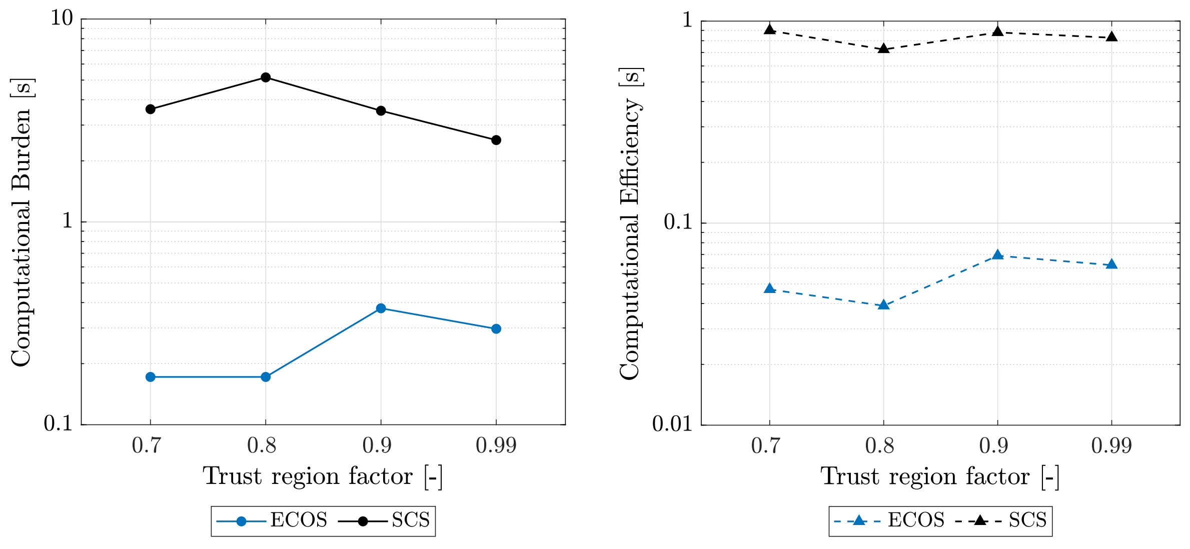
[6]Available at: <`https://www.cvxpy.org/index.html`> (last accessed on 13/11/2023)

Reliability	Constitutes the percentage of converged optimal solutions among the MC samples, measured through: $$R\,(\%) = \frac{\text{simulations} - \text{failures}}{\text{simulations}} \times 100\,. \qquad (4.5)$$
Effectiveness	Reflects the variation of the above criteria with a change in the initial guess or test case.

4.2. Parametric Analysis

Trust Region Factor

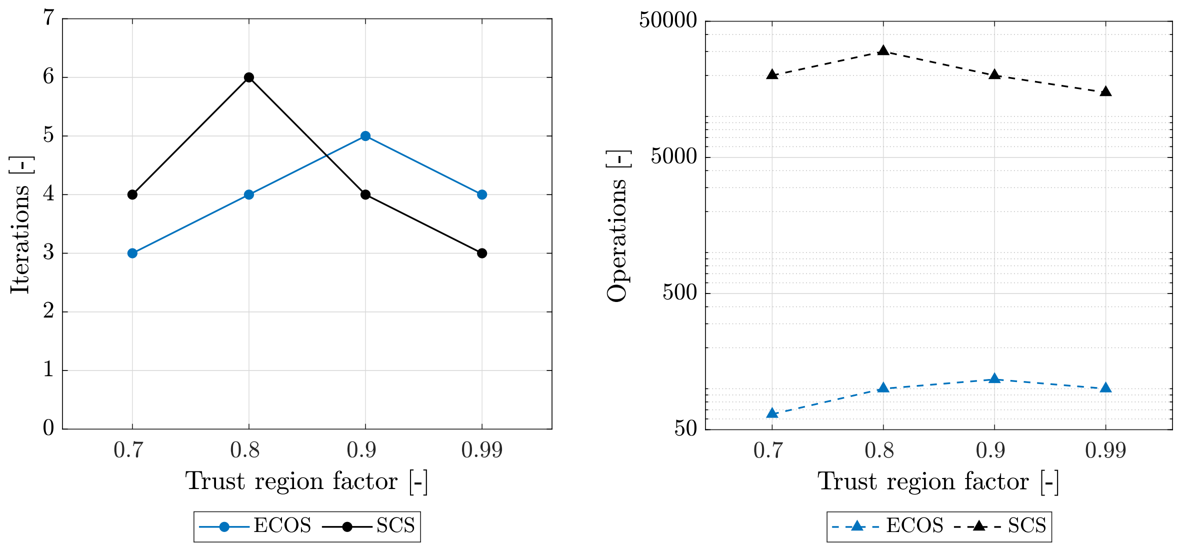
As outlined in Subsection 3.4.2, the trust region factor embodies the intricate interplay between convergence rate, feasibility, and computational efficiency, shaping the outcome of the optimisation algorithm. In order to understand how these values are affected, an analysis was done using the Earth–Mars case, varying the trust region factor and leaving all other parameters constant. This is shown by reporting the solver performance and efficiency in Figure 4.1 and Figure 4.2.



(a) Burden. (b) Efficiency.

Figure 4.1: Effect of the trust region factor on computational loads.

These initial figures clearly illustrate how variations in the parameter γ have a pronounced impact on the obtained results. Particularly, when running the ECOS solver, a trust region factor of 0.7 emerges as the optimal choice in terms of computational burden and efficiency. On the other hand, with regards to SCS, the computational efficiency remains relatively consistent across different γ values, with only a marginal decline observed for higher values of the trust region factor. In the context of problem complexity, variations in the number of iterations directly correlate with the number of operations required to reach an optimal solution. The study highlights that, for both solvers, a γ value of 0.7 strikes as a favourable balance between problem complexity and computational behaviour.



(a) Iterations. (b) Operations.

Figure 4.2: Effect of the trust region factor on complexity.

Another parameter that had to be considered, as it is vital for space missions, is the accuracy of the propagated solution, which is reported in Figure 4.3. In the figure, the difference between the desired final radial position, r, and that obtained through a propagation of the optimal solution is reported as it was the leading parameter for the accuracy investigation, with more pronounced differences with respect to the velocity and angles.

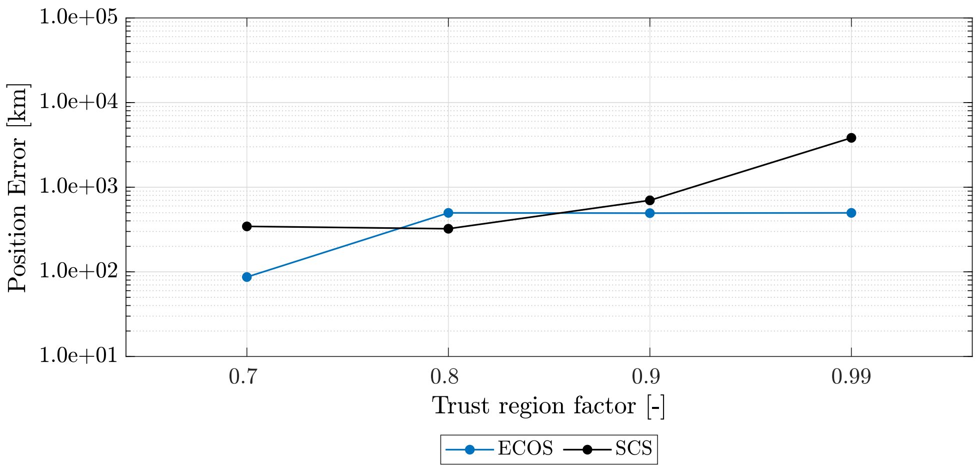


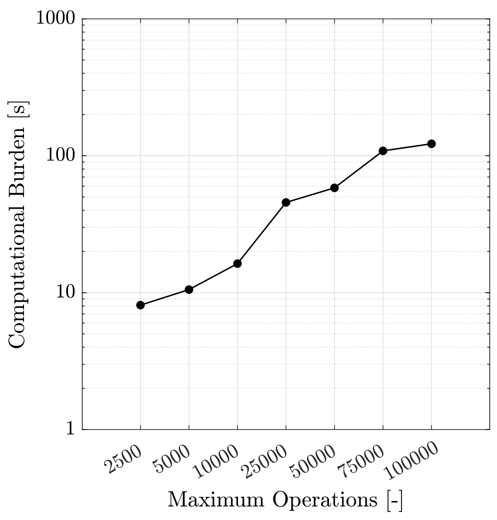
Figure 4.3: Effect of the trust region factor on accuracy.

Ultimately, following this last analysis, the value of γ was chosen to be of 0.7 for the Mars case. This was based on the conclusion that it guaranteed the overall best performance of the solvers while allowing for the highest accuracy. Furthermore, in support of this decision, a consistent behavior was noted when assessing the accuracy across both velocity and spherical angles.

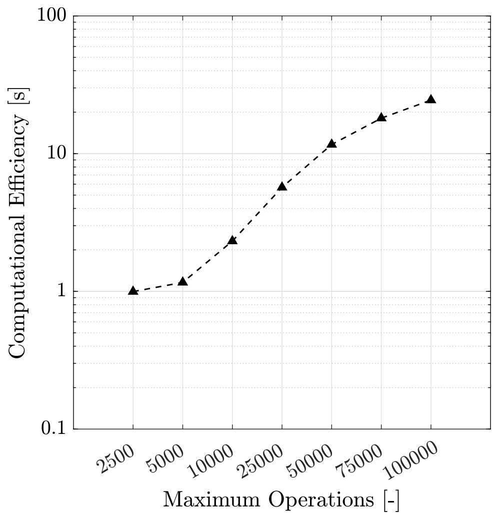
Since the trust region factor is an algorithmic parameter rather than a solver-specific one, the same analysis was conducted to determine an appropriate value of γ for the remaining test cases (see Section A.1). This approach was taken to ensure that both solvers achieved their highest overall performance to grant a fair and equitable comparison between them, which is purely based on the solver type, resulting in a value of 0.8 for the SEL$_2$-NEA case and of 0.99 for the Earth–Venus trajectory.

Operations Limit

Following the initial simulations, a notable difference in the maximum operations admissible (*max_iters*) between the default value of the SCS solver and that imposed by CVXPY was observed. While SCS defaults to 100,000 iterations, CVXPY sets a limit of 2,500 iterations when using the solver. Recognising the significance of this difference, an investigation was carried out to understand how the maximum number of operations affected the obtained solutions. The subsequent findings, illustrated in Figure 4.4, Figure 4.5, and Figure 4.6, outline the impact of varying this parameter for the SEL$_2$–NEA case. It is worth noting that similar trends were observed for the other two test cases, and these results are provided in Section A.2 for reference.

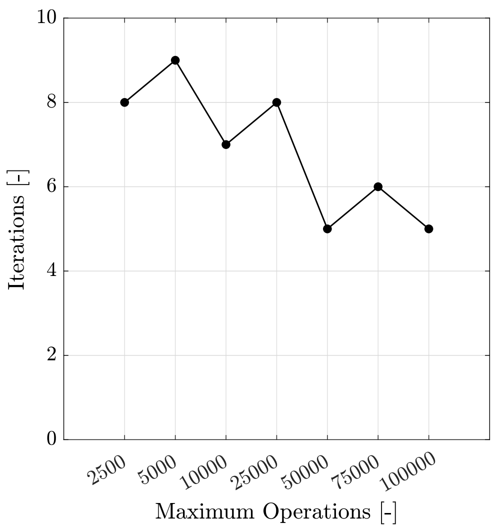


(a) Burden.

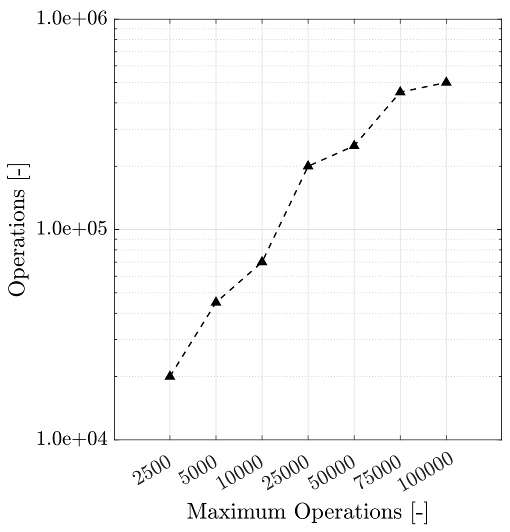


(b) Efficiency.

Figure 4.4: Effect of SCS operations limit on computational loads.



(a) Iterations.



(b) Operations.

Figure 4.5: Effect of SCS operations limit on complexity.

The maximum allowable operations have a noticeable and consistent impact on both the computational burden and efficiency of the solver. In fact, with an increase of this parameter, there is a corresponding increase in both the total computational time and the time per iteration. This trend is closely linked to the behaviour illustrated in Figure 4.5, where a decrease in the number of iterations is observed, leading to an increase in the total number of operations required to reach optimality. This decrease in the number of SCP iterations can be attributed to the ability of the solver to execute more operations before terminating and providing the best optimal solution for successive runs. Nevertheless, the progress made while reducing the number of iterations does not offset the losses in terms of operations, which have a direct impact on the computational load.

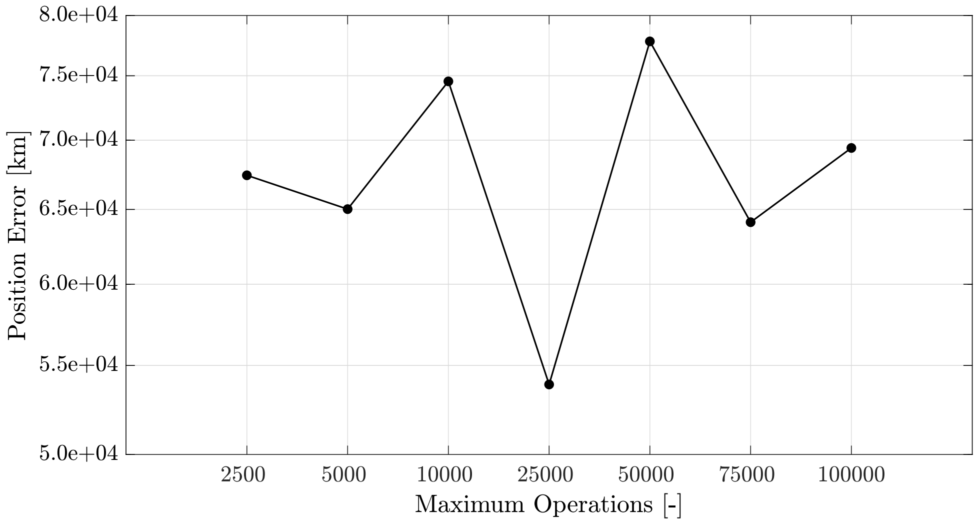


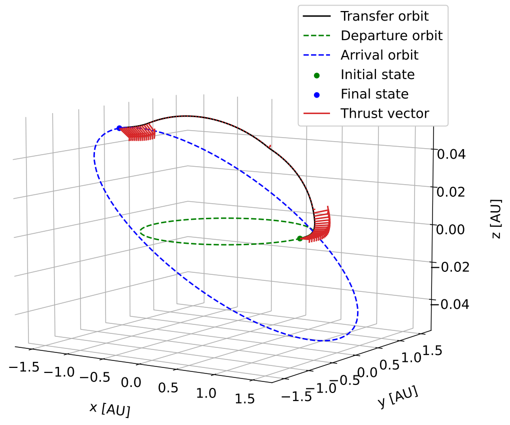
Figure 4.6: Effect of SCS operations limit on accuracy.

While fewer operations may seem preferable from a computational and complexity point of view, it is essential to consider solution accuracy. Figure 4.6 reveals that there is no evident relationship between a variation in *max_ iters* and the accuracy of the solution. Due to these considerations, a value of $2,500$ maximum operations was selected for the SCS solver. This decision reflects the understanding that a marginal improvement in solution accuracy would not be worth the associated computational and operational trade-offs.

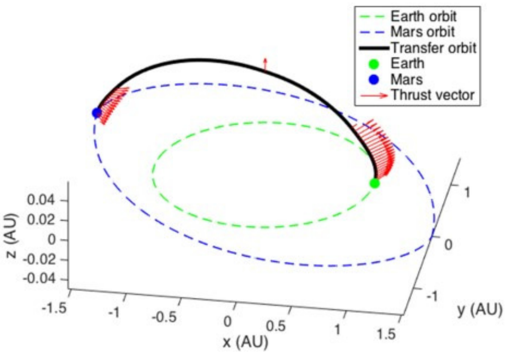
As the limit imposed on the maximum operations for ECOS was the same in the default values of the solver and CVXPY, and because this limit was never reached throughout the iterations, no investigation was carried out on the operations limit for ECOS.

4.3. Results Validation

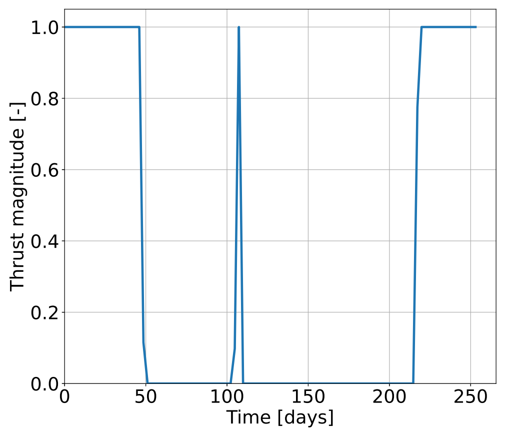
In order to ensure that the data collected is reliable, it is important to verify that the results obtained are close to those presented in the literature. Figure 4.7 presents the trajectories, thrust profiles and corresponding mass changes of the optimal solution from the SCP and the results from the literature, for the Earth–Mars case. As the results for the two convex methods have a negligible difference with regard to the median outputs, the choice to present only the solution from ECOS was taken.



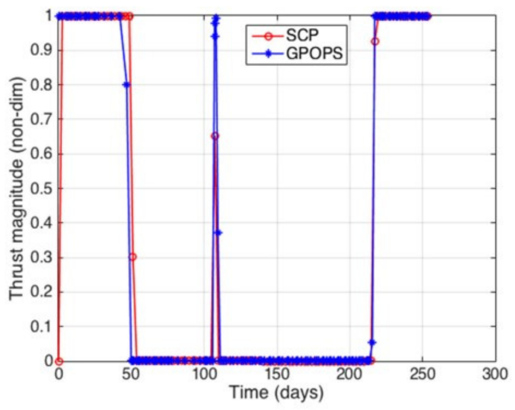
(a) SCP trajectory.



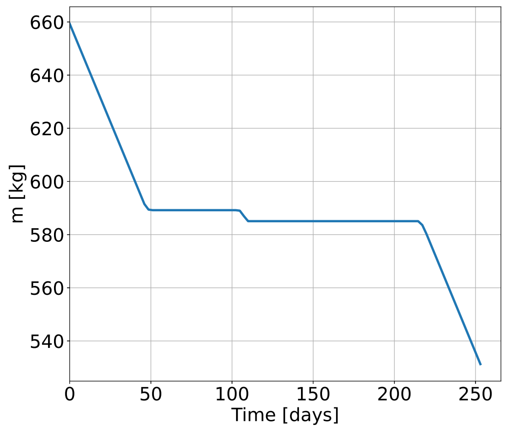
(b) Benchmark trajectory [11].



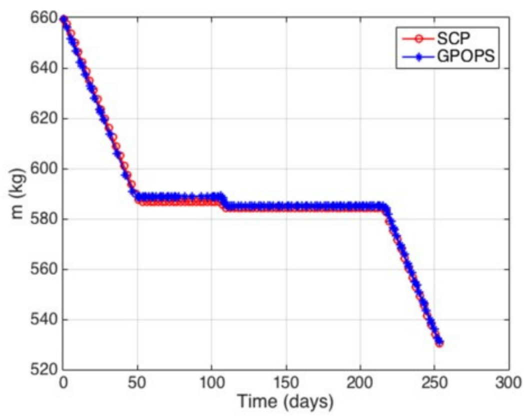
(c) SCP thrust profile.



(d) Benchmark thrust profile [11].



(e) SCP mass profile.



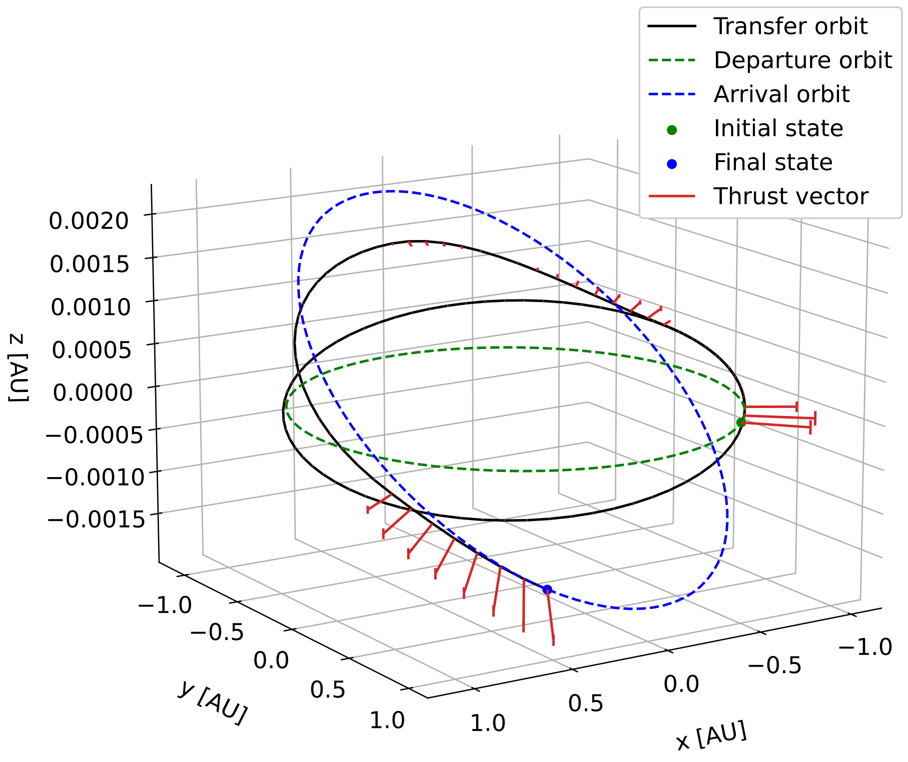
(f) Benchmark mass profile [11].

Figure 4.7: Results obtained from the proposed SCP of this thesis (left) and from Ref. [11] (right) for the Earth–Mars case.

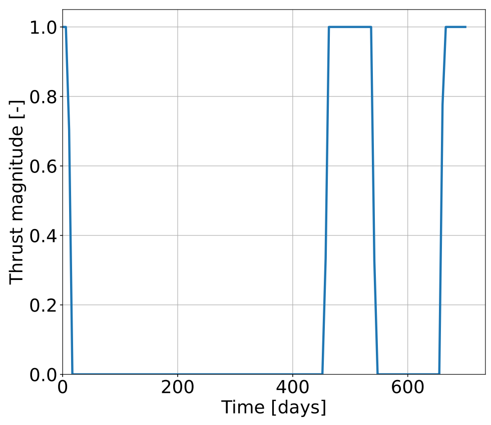
Analysing the two outputs, it can be concluded that the obtained results compare closely to the literature, providing a means of validating the proposed problem formulation and implementation. Of importance is also the standard bang-bang profile common to optimisation techniques. This is reflected in the mass changes of the spacecraft, with constant negative gradients obtained when the propulsion unit is turned on at the maximum thrust.

The final mass of the SCP algorithm presented by Ref. [11] of 530.33 kg is of a similar magnitude to that obtained from this work of 531.29 kg. What stands out from the two solutions is the number of iterations required to reach convergence. In fact, the proposed algorithm took a median of 4 iterations to reach optimal convergance, whereas the benchmark required 13 [11]. Furthermore, the computational toll of the optimisation is of 0.043 s per iteration and 0.344 s total, compared to the values of Ref. [11] of 0.25 s per iteration and 2.5 s total. This being said, it has to be considered that the programs have been run using different software and on different machines, and hence to properly compare the two parameters the SCPs should be run on the same computer. Moreover, no mention of a MC simulation was stated in Ref. [11].

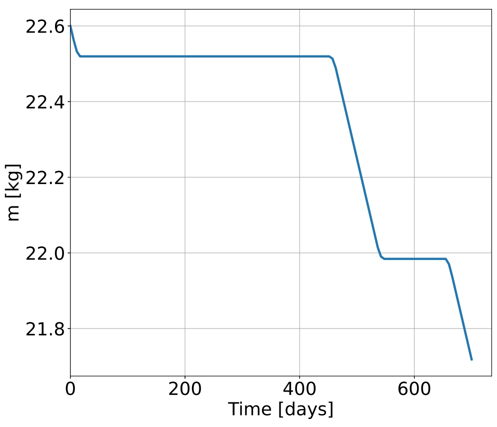
The same verification was made with the SEL_2–NEA case (Figure 4.8) and the Earth–Venus case (Figure 4.9), using as a benchmark the results obtained from Ref. [1], from which the same conclusions were drawn. In fact, similar final masses, thrust profiless and trajectories were observed. As the investigation of discretisation methods, trust regions, nodes and orders of integrating polynomials is not comparable to that of the solver methods, other parameters could not be compared. For a comprehensive exploration of the former, the interested reader is referred to Ref. [1].



(a) SCP trajectory.

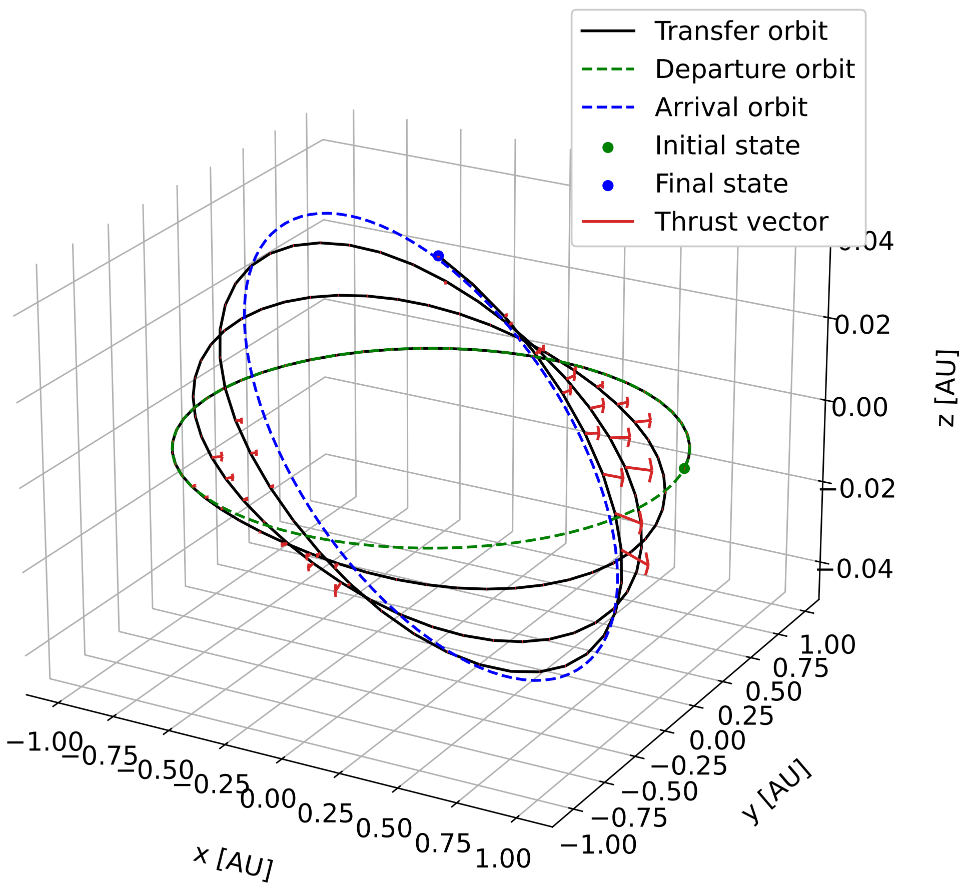


(b) SCP thrust profile.

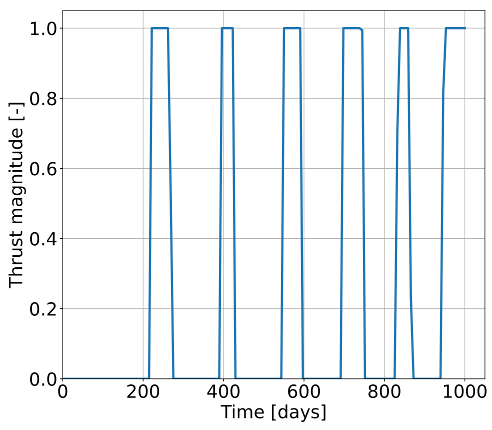


(c) SCP mass profile.

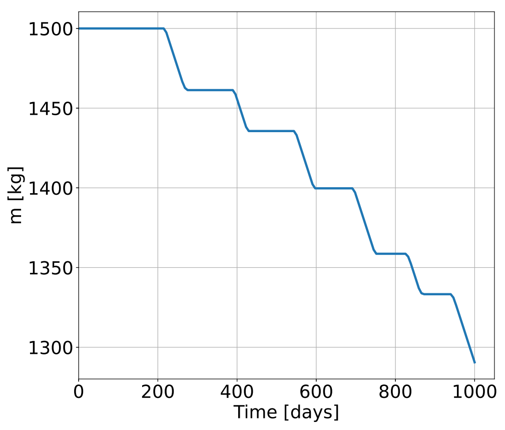
Figure 4.8: Results obtained from the proposed SCP of this thesis for the SEL$_2$–NEA case.



(a) SCP trajectory.



(b) SCP thrust profile.



(c) SCP mass profile.

Figure 4.9: Results obtained from the proposed SCP of this thesis for the Earth–Venus case.

4.4. Trade-off Criteria Analysis

This section presents and investigates the trade-off criteria outline in Table 4.3 for the two solvers and all test cases.

4.4.1. Memory

When comparing the memory characteristics of both solvers, two key aspects were assessed: the size of the compiled code and the memory usage during the execution of the SCP. Analysing the results obtained, it can be observed that with an increase in the problem complexity, and hence in the number of nodes for the trajectory, the dimension of the compiled code increases. Like the nodal increment, this increase is seen to be linear since with the nodes the number of parameters and constraints also increases. ECOS, in particular, generates a compiled code that is approximately 18% larger, indicative of the more complex operations performed by the IPM solvers [4, 30].

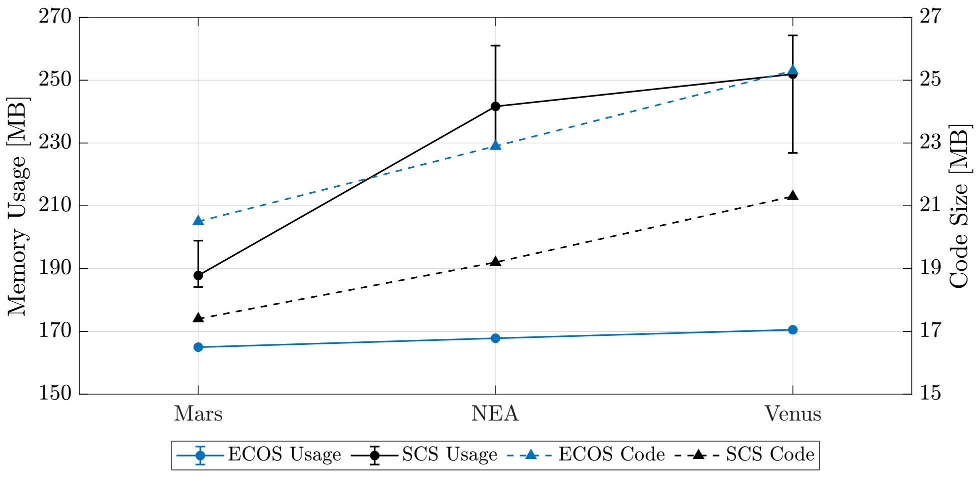


Figure 4.10: Memory of compiled code.

The memory consumption was found by recording the difference in RSS memory size from the start to the end of the optimisation process using the `psutil`[7] library. The results obtained for the two solvers can be seen to differ. Firstly, ECOS exhibits an extremely favourable behaviour, 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. The FOM, instead, requires a greater memory usage resulting in values that increase with problem complexity reaching a maximum of 291.414 MB. Moreover, the difference in consistency is not only in the different cases, but also in the MC samples as there is

[7]Available at: <`https://pypi.org/project/psutil/`> (last accessed on 13/11/2023)

a greater sparseness in range for SCS. This difference is seen to increase with problem, while it remains indifferent and negligible for the IPM. It is essential to note that the dissimilarity in memory usage between solvers can be contingent upon the particular problem at hand and the configurations of the solver settings. Nevertheless, ECOS is renowned for its memory efficiency, and its ability to handle large-scale problems with a relatively low memory footprint [52] is reflected in the presented results.

Table 4.4 indicates the difference in the size of the problems investigated. The variables of a problem in CVXPY are defined as the optimisation variables which are free to vary in order to minimise or maximise an objective function. The parameters, instead, represent constant expressions whose value is specified after the problem creation [49]. As outlined in Subsection 3.4.5, their definition is of extreme importance for the SCP algorithm as they allow to modify the problem at each iteration.

Table 4.4: Problem dimension with test case.

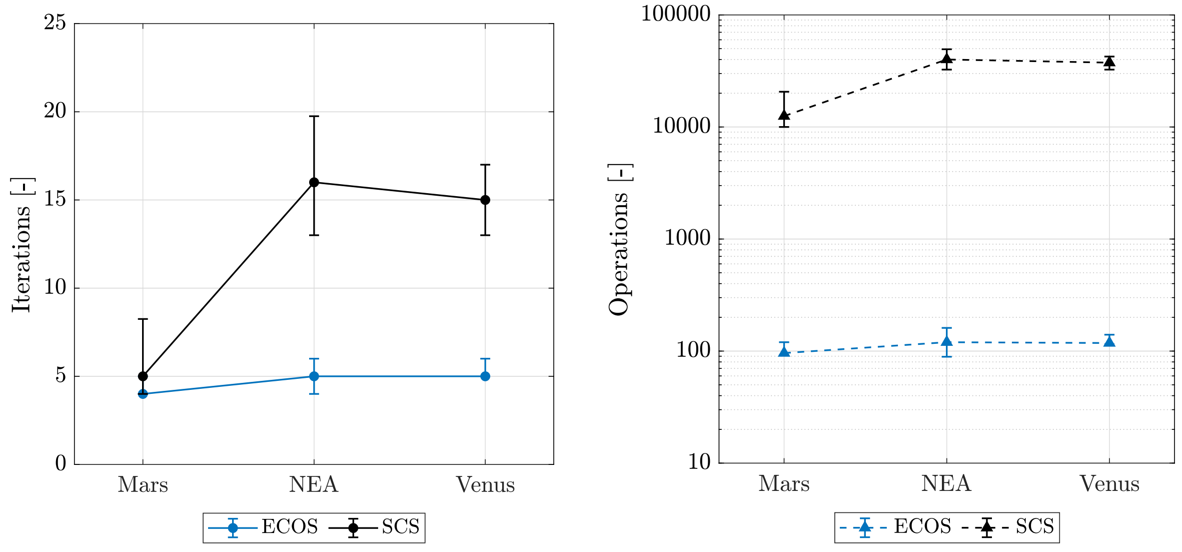
	Earth–Mars	**SEL$_2$–NEA**	**Earth–Venus**
Nodes	100	125	150
Constraints	3720	4645	5570
Variables	2600	3250	3900
Parameters	6407	8007	9607

As aforementioned, the nodal increase has a direct impact on the problem size. The increment in the constraints, variables and parameters is constant between the cases since they are dependent on the number of nodes. The constraints are in fact imposed on the dynamics (with a dependency of $7N$), the lower and upper bounds ($22N$), the control (N), and $\Delta\boldsymbol{x}$ ($7N$). The BCs and trust region, instead, have a constant size of respectively 13 and 7 for each problem as they are independent of the number of nodes. Similarly, the increment in the number of variables is constant, as the variables are directly proportional to the dimensions of $\boldsymbol{u}$, $\boldsymbol{x}$, $\Delta\boldsymbol{x}$, $\boldsymbol{h}$, and η ($4N + 7N + 7N + 7N + N = 26N$). The same can be said for the parameters, with a dependency on $\boldsymbol{f}$, $\boldsymbol{A}$, $\boldsymbol{x}^*$, $\exp_z$, and $\boldsymbol{\Upsilon}$ ($7N + 49N + 7N + N + 7 = 64N + 7$).

4.4.2. Complexity

The complexity of the optimisation solvers can be evaluated with two parameters that are intertwined: the iterations of the SCP and the number of operations needed for the solver to reach optimality. The results obtained for the three cases are reported in Figure 4.11.

The number of operations and iterations required by ECOS to obtain an optimal solution remains fairly similar between the different simulations, showing once again the strong consistency of the IPMs [4, 30]. Regarding SCS, although the SCP iterations are in the same order of magnitude as ECOS, the values of the solver operations are notably higher, differing by two orders of magnitude. Moreover, a difference in the number of iterations and operations can be observed, with a jump in value when passing from the simpler case to the NEA trajectory. The values of the SCS parameters show a small decrease for the Earth–Venus case, highlighting a better performance for the most complex scenario.



(a) Iterations. (b) Operations.

Figure 4.11: Iterations and operations of simulations.

4.4.3. Computational Burden and Efficiency

The computational values reported in Figure 4.12 are obtained by clocking the difference in time between the start and the end of each SCP iteration. The Python `time` module was adopted, using `process_time_ns` to return the value, in fractional nanoseconds, of the sum of the system and user CPU time.

A similar behaviour can be seen for the two methods when analysing the computational efficiency of the solvers with different problems, with an almost linear increase, although the values of the FOM solver remain higher.

The total computational time needed to run the optimisation algorithm is a result of the interplay between the computational efficiency of the solver and the iterations reported in Figure 4.11. As anticipated, both methods exhibit an increase in the computational demands as problem complexity rises. For ECOS, due to the small linear increment in iterations, the time required to reach an optimal solution increases linearly with the complexity of the problem. This increase, however, is small when considering the magnitude of the values, resulting in a remarkably fast solution spanning from 0.344 s for the Earth–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 prominent increase. Overall, the values obtained remain superior for the IPM solver, with more than an order of magnitude improvement for the parameters investigated.

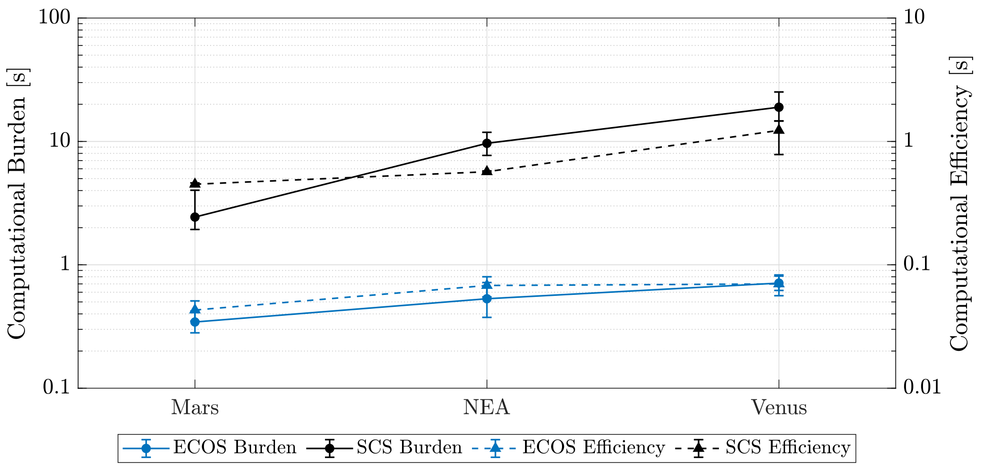


Figure 4.12: Computational efficiency and burden of simulations.

4.4.4. Accuracy

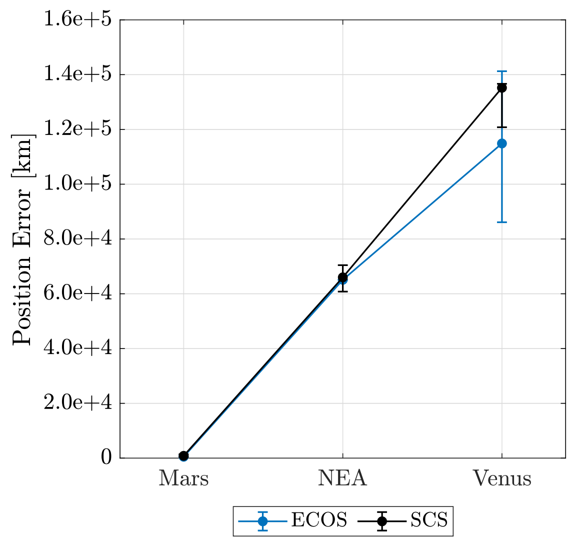
The accuracy of the solution, ϵ, is found by propagating the optimised control vector using the `scipy`[8] integrator `odeint`. As the values of $\boldsymbol{u}$ are known only at the nodes, the control in between the nodes is obtained by interpolating the control history. Due to the discretisation used, there is a difference in solution between the desired final state and the propagated one as seen in Figure 4.13 and Table 4.5, which is calculated through:

$$\epsilon = |\tilde{\boldsymbol{x}}(t_f) - \boldsymbol{x}_f| \,. \tag{4.6}$$

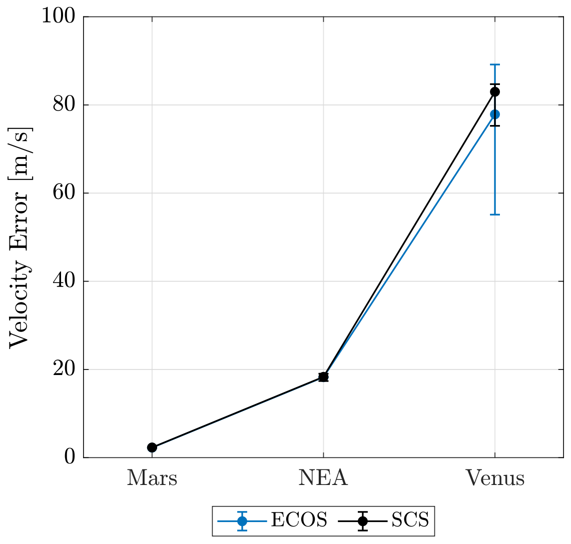
To ensure that the error in the accuracy of the methods, which at first impact might seem high, was due to the discretisation method and not the problem formulation, an investigation was done by increasing the number of nodes, and can be seen in Appendix B. The

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

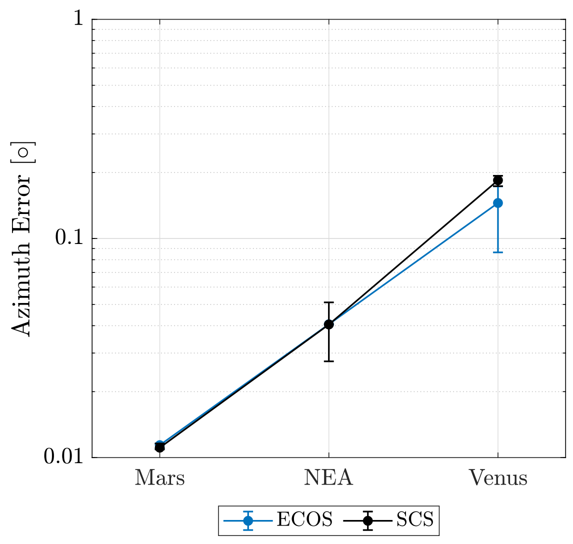
results proved that increasing the nodal number increased the accuracy of the solution, validating the method used.



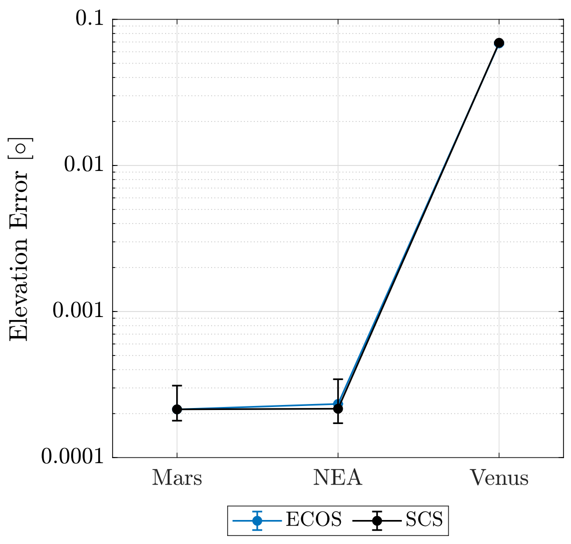
(a) Distance.



(b) Velocity.



(c) Azimuth.



(d) Elevation.

Figure 4.13: Accuracy of the propagated optimal solution.

Although the results for the ECOS solver are better in terms of overall accuracy, they can be treated as comparable to the results of SCS as they are within the same order of magnitude. This is understandable as the accuracy of the optimal solution is dependent on the level of tolerance set in the solver, which is set to the same value of 10^{-8} for both solvers. The tolerance is imposed on the relative and absolute values of the duality gap (see Section 2.1), which represents a measure of the optimality of the solution [22].

Moreover, the accuracy can be seen to decrease with the complexity of the problem, which is attributed to the increase in time of flight. As trajectories become more prolonged, they may involve more complex manoeuvres, multiple revolutions, or extended coasting phases. In addition, small errors in the initial states can lead to larger errors in the final state, when propagated, making it more difficult to find a precise solution. Moreover, the aforementioned interpolation error becomes more significant as the time of flight grows.

Table 4.5: Accuracy in spherical coordinates of the propagated optimal solution.*

	Earth–Mars		**SEL$_2$–NEA**		**Earth–Venus**	
Accuracy	ECOS	SCS	ECOS	SCS	ECOS	SCS
r (km)	497.237	836.414	65096.348	66000.333	114870.838	135196.287
θ (rad)	0.011	0.011	0.041	0.041	0.145	0.184
ϕ (rad)	2.138e-04	2.142e-04	2.334e-04	2.155e-04	6.838e-02	6.911e-02
v_r (m/s)	2.305	2.306	10.924	10.923	57.106	61.059
v_θ (m/s)	0.091	0.092	14.630	14.630	32.501	38.217
v_ϕ (m/s)	0.034	0.034	0.195	0.195	23.223	21.878

*Values reported are median results of 100 MC simulations.

4.4.5. Optimality

As with the accuracy, the values of the optimality (J) of the solvers increase with the time of flight, with very similar values for the IPM and FOM solvers. Recalling the problem definition, the objective of the minimum-fuel trajectory optimisation problem is reflected in minimising a pre-defined function in order to maximise the final mass. It is therefore understandable that a measure of the performance of the solver is both J and the value of the mass at time t_f, which are shown in Table 4.6.

Table 4.6: Optimality of converged solution.*

	Earth–Mars		**SEL$_2$–NEA**		**Earth–Venus**	
	ECOS	SCS	ECOS	SCS	ECOS	SCS
J (-)	1.661222	1.661252	2.391095	2.391100	5.071635	5.071716
$m(t_f)$ (kg)	531.293	531.276	21.718	21.718	1290.568	1290.552

*Values reported are median results of 100 MC simulations.

4.4.6. Reliability

The ability of the solver to consistently reach an optimal solution is paramount when applied to real-life and time-critical applications. The IPM solver shows an extremely positive output under this performance criteria, as the success rate remains high across all cases. Conversely, the success rate of SCS experiences a decrease with an increase in problem complexity. To rigorously challenge the two solvers and push their performance to the limit, the convergence tolerance of the constraints, ε_c, was reduced from an initial value of 10^{-6} to a value of 10^{-7} after determining that it was the leading criterion for convergence. As seen from Figure 4.14, this showed that with a more stringent criterion, the reliability of the solutions diverges between methods.

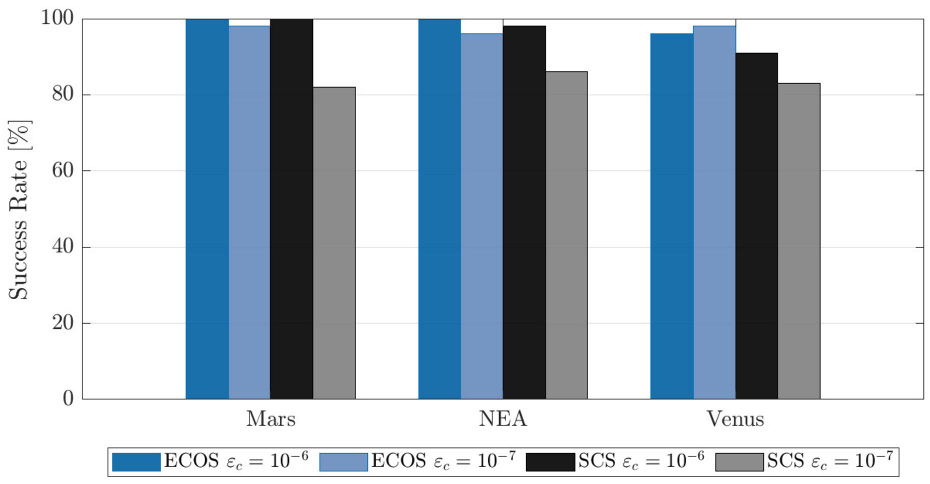


Figure 4.14: Reliability and effectiveness of simulations.

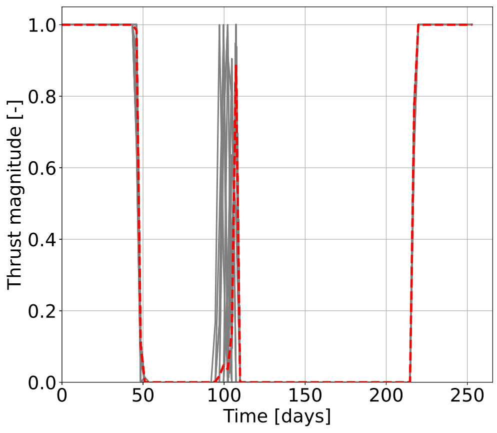
The consistency of ECOS and its ability to reach an optimal solution remains high, whereas SCS once again shows its weakness in terms of problem consistency, with a more strict requirement increasing the SCP runs with no solution. An interesting observation emerges when considering the FOM solver as the success rate across the three cases is significantly influenced by the choice of convergence tolerance. A higher value in ε_c exhibits reduced reliability with problem complexity, while a decreased value yields lower but congruent results. This illustrates the sensitivity of SCS to the chosen convergence tolerance, highlighting the importance of the parameter selection.

4.4.7. Effectiveness

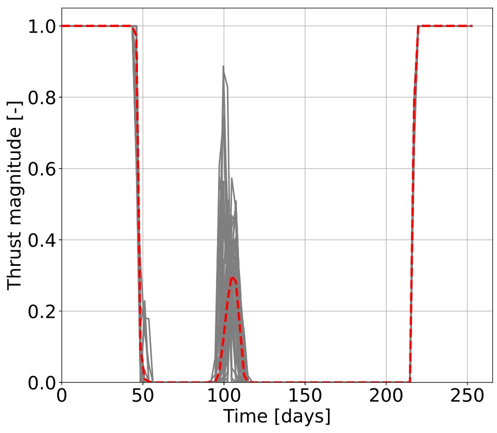
The convergence rate is not the only parameter that has to be considered when judging the consistency of a method, as it is essential to observe how the parameters vary between different runs and from one case to another. The thrust profiles of all converged solutions are displayed in Figure 4.15, where all the profiles of the MC samples are shown in grey. The mean of the thrust magnitude of all converged samples is taken at each node to investigate the consistency of the solution, with the resulting profile reported in red.

When analysing the first two cases, it is interesting to see how ECOS shows a much more resonant behaviour, with consistent bang-bang profiles except for only two instances in the SEL_2–NEA case. What further supports this claim is the mean, which exhibits a congruous profile. In fact, the noise caused by two MC samples in the NEA trajectory is seen to be insignificant when calculating the mean of the profile, proving that the IPM solver obtains consistent and similar bang-bang profiles for that case. On the other hand, the profiles for SCS are seen to be less optimal for the Earth–Mars case, with the plane-change maneuver at 100 days [11] occurring over a greater time span and at a lower magnitude compared to ECOS. This is not the case for the second trajectory, where bang-bang profiles are indeed observed, but with a much lower consistency between themselves. In fact, several conflicting profiles can be seen, causing the mean to exhibit noise.

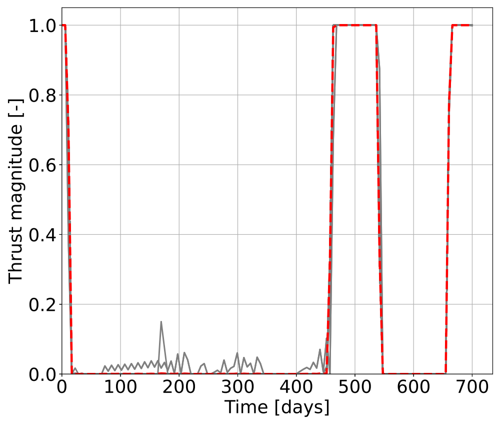
A much different conclusion can be drawn when examining the more complex Earth–Venus case. The IPM solver finds two solutions for the minimum-fuel problem, one with an initial fuel burn at around 60 days and one without. Although both solutions are valid as they grant similar final masses and show the bang-bang profile, this behaviour highlights a limit in the consistency of outputs for more complex scenarios for ECOS. As a result, this causes the mean solution to exhibit a less bang-bang-like profile compared to the first two cases, as seen in Figure 4.15e. In contrast, SCS exhibits a much more favourable behaviour, with congruous bang-bang profiles being seen through all of the converged solutions and hence in the mean profile.



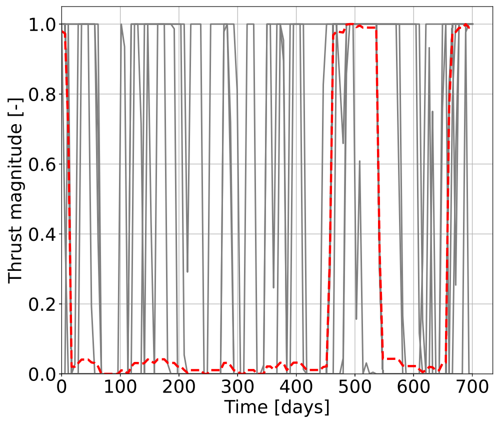
(a) Earth–Mars ECOS.



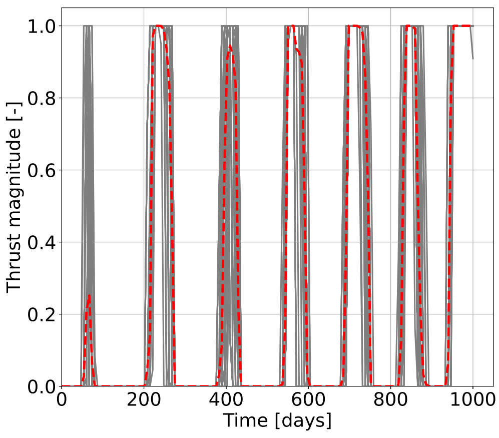
(b) Earth–Mars SCS.



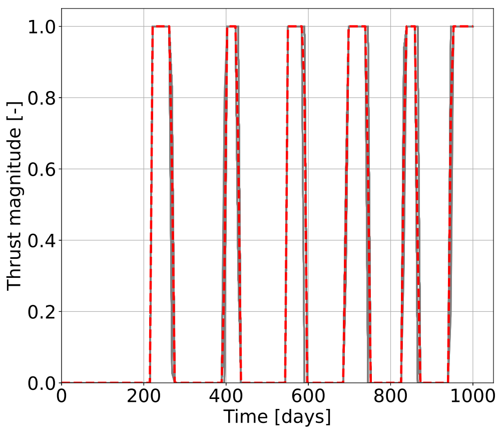
(c) SEL_2–NEA ECOS.



(d) SEL_2–NEA SCS.



(e) Earth–Venus ECOS.



(f) Earth–Venus SCS.

Figure 4.15: Output of the thrust profiles obtained using ECOS (left) and SCS (right). Grey lines: converged solutions, red line: mean profile.

In order to analyse how the criteria in the previous sections behave when a different problem is posed, through MC sampling, or when a different test case is considered, all of the reported data needs to be analysed. The standard deviation of the solution shown in Table 4.7, along with the values of the lower and upper quartiles displayed in the previous graphs, help to understand the variation of the results obtained for the three cases. This information is of importance as it is a measure of the consistency of the solver in achieving the same solutions with a change in MC sample, or test case, in the same way as Figure 4.15 was for the thrust profiles.

Table 4.7: Standard deviation of propagated solution.

	Earth–Mars		**SEL$_2$–NEA**		**Earth–Venus**	
Standard deviation	ECOS	SCS	ECOS	SCS	ECOS	SCS
Memory usage (MB)	0.024	25.282	0.031	43.708	0.037	170.374
Iterations (-)	0.934	5.613	1.526	7.506	1.331	6.281
Operations (-)	34	14033	127	18766	41	15703
Burden (s)	0.122	2.722	0.426	4.539	0.257	11.444
Efficiency (s)	0.011	0.037	0.032	0.019	0.015	0.343
Accuracy r (km)	837.091	2440.396	2455.559	10686.106	87515.234	7839.907
Accuracy v (m/s)	1.036	1.028	6.135	6.190	27.066	18.019
J (-)	3.935e-04	8.464e-04	1.203e-02	3.025e-01	5.102e-03	2.532e-04
Mass (g)	27.164	59.074	4.360	106.440	191.998	11.448

From the presented data it can be concluded that the sparseness of the solutions is higher for the FOM when compared to the IPM except for one instance. The area in which SCS behaves better is in the accuracy of the solution for the more complex Earth–Venus case, and in its variation in optimality, with the mass having a smaller variation of one order of magnitude compared to ECOS. This is consistent with the thrust profiles analysed in Figure 4.15f, which showed a much more congruent behaviour for SCS with respect to the two conflicting solutions found by the IPM solver. Another criterion that stands out is the difference in memory usage. The values are extremely consistent for ECOS while having a greater divergence for SCS, which was found to increase with problem complexity. The behaviour of the variation of the accuracy can be seen to be reflective of the pattern observed in Subsection 4.4.4, with an increase in the time of flight of the trajectory increasing the standard deviation, apart from the radial error of SCS. Moreover, the SEL$_2$–NEA case proved to be more challenging for the solvers than anticipated, as shown in the higher variation in operations and iterations required by the two solvers compared to the Earth–Venus case, reflective of Figure 4.11.

4.5. CVXPY Analysis

To investigate the solutions obtained with ECOS, some simulations were made using the other available IPM solvers in CVXPY for the Earth–Mars trajectory. Due to the absence of their implementation in CVXPYgen the compiled code could not be generated. Although the computational performance of the solvers in CVXPY is not comparable with that in CVXPYgen, other parameters that are not dependant on the run time are and can be seen in the Table 4.8. Since the non-compiled codes require a greater run time, the number of MC samples was reduced to 30 in order to gather the required data within a reasonable amount of time. All the simulations carried out with the IPM solvers converged to an optimal solution, highlighting the consistency of these methods in terms of accuracy and complexity. Additionally, the memory usage of the solvers can also be seen to be of the same order of magnitude, proving once again their consistency. The number of operations required to reach convergence for the MOSEK and CPLEX solvers is not reported due to the inability of CVXPY to obtain that information.

Table 4.8: Analysis of solvers in CVXPY (Earth–Mars case).*

	MOSEK	GUROBI	CLARABEL	COPT	CPLEX	NAG
Memory usage (MB)	201.285	184.191	175.094	183.393	167.174	162.705
Iterations (-)	4	4	4	4	4	4
Operations (-)	-	65	99	97	-	119
Burden (s)	43.422	40.211	41.383	39.859	63.445	66.633
Efficiency (s)	10.996	10.104	10.025	9.934	15.521	16.147
Accuracy r (km)	672.352	497.567	486.116	497.709	496.243	531.293
Accuracy v (m/s)	2.251	2.299	2.284	2.289	2.264	2.230
J (-)	1.661386	1.661224	1.661222	1.661234	1.661225	1.661224
Success rate (%)	100	100	100	100	100	100

*Values reported are median results of 30 MC simulations.

With regards to the FOM solvers, SCS is currently the only one that supports SOCPs in CVXPY, and hence no comparison could be made with solvers of the same family. However, the simulations made with the CVXPY software using SCS remain of importance as they can be used, combined with the ECOS runs, to support the advantage of using a compiled code instead of a non-compiled one, as highlighted in Table 4.9.

Table 4.9: Comparison of compiled and non-compiled code (Earth–Mars case).*

	ECOS		SCS	
	CVXPY	CVXPYgen	CVXPY	CVXPYgen
Memory usage (MB)	172.604	164.961	174.340	187.797
Iterations (-)	4	4	4	5
Operations (-)	94	96	100000	12500
Burden (s)	45.836	0.344	55.898	2.438
Efficiency (s)	10.736	0.043	13.354	0.450
Accuracy r (km)	497.493	497.237	723.729	836.414
Accuracy v (m/s)	2.285	2.268	2.294	2.303
J (-)	1.661222	1.661222	1.663951	1.661252
Success rate (%)	100	100	100	100

*Values reported are median results of 30 MC simulations.

Understandably, as the problem formulation along with the convergence criteria and solver tolerance settings are kept the same, the accuracy levels obtained using CVXPY and CVXPYgen are of the same order. Moreover, the memory consumption during the optimisation process is seen to remain fairly similar between the two software. What is of importance, instead, is the significant saving in both the computational burden and efficiency when passing from the raw code to the compiled one, with a greater improvement observed for ECOS. During this analysis, it was noted that for SCS to obtain comparable results the maximum number of operations had to be raised to $10,000$ using CVXPY.

5 | Conclusions and Future Work

5.1. Conclusion

In this work, a convex approach has been applied to the classical minimum-fuel problem of low-thrust trajectory optimisation in order to assess the performance of convex solvers for onboard interplanetary missions. The method adopted involved analysing three different test cases, with increasing problem complexity and with different initial and final objectives, to compare predefined trade-off criteria and grant a complete and unbiased assessment. These scenarios comprised the Earth-to-Mars, SEL_2-to-NEA, and Earth-to-Venus transfer problems taken from literature and which played a role in validating the proposed algorithm and the results obtained. The solvers used for this study have been selected as ECOS (for the interior-point method) and SCS (for the first-order method) as they represent the pinnacle of their solver family at the time of writing.

In order to allow for the solvers to perform at their best capabilities, an investigation has been done into the trust region factor adopted for each test case, along with the maximum number of allowable operations for each solver. The outcome of this survey resulted in a higher maximum number of allowable operations for SCS compared to ECOS. The trust region factor, used to keep the solution in the neighbourhood of the reference trajectory and improve convergence of the SCP, was found to be reflective of the increasing difficulty of the trajectories, as tighter constraints were imposed for the simpler cases.

Results from the simulations show a clear difference in performance between the two solvers. When analysing the size of the compiled code a favourable result emerges for the FOM solver, with a smaller code size of 18% for SCS compared to ECOS, owning to the simpler operations required by FOMs. In contrast, the analysis of the memory usage shows that while ECOS exhibits a quasi-constant memory consumption between cases, SCS demonstrates a much more prominent increase with problem complexity.

A difference is also noted in the complexity of the solvers, with the ECOS requiring a fairly low and consistent number of operations to converge to an optimal solution. On the other hand, although SCS shows a similar, consistent, behaviour with an increase in problem complexity, the number of operations remains two orders of magnitude higher.

With regards to the accuracy and optimality of the solutions, the two methods are comparative, as these criteria are reflective of the same level of tolerance imposed on the duality gap of the solution. Significantly, the accuracy of the solution decreases with the complexity of the problem due to the extended time of flight which leads to a more pronounced interpolation error.

The reliability results in being superior for the IPM, with tighter convergence criteria on the constraints highlighting the discrepancies between the solvers. In fact, the success rate of ECOS is consistently high for all test cases, remaining above 96%, while it exhibits a decreasing behaviour for SCS, reaching as low as 82% for the Earth–Venus trajectory. In contrast, 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 has more difficulty in doing so, especially for the second case. The Earth–Venus trajectory, instead, results in extremely consistent bang-bang profiles for the FOM solver, whereas the IPM solver obtains two conflicting solutions. Hence, although the success rate of SCS is lower, the results for the more complex scenario are more consistent in terms of trajectory compared to ECOS.

Another parameter that is of key importance in real-life and decision-making scenarios like deep-space applications, where a fast response can provide significant overall savings for the mission, is the computational toll of the process. The comparison with CVXPY and CVXPYgen demonstrates the powerful benefits of using a compiled code compared to a non-compiled one. Results from the latter show an increase in computational burden and efficiency for both solvers, with values that remain below one second for ECOS and an order of magnitude greater for SCS, for the trajectories investigated. However, considering the deep-space low-thrust application, where transfer orbits take in the order of hundreds of days, even the run time for SCS is negligible when compared to the total time of flight of the trajectory.

Finally, the consistency of the two methods under study has also been analysed by taking into account how the criteria above change with different cases and MC samples. It was found that ECOS exhibits a behaviour that is much more consistent, with a small linear decrease in performance with an increase in problem complexity. SCS shows a similar relation with problem complexity while displaying a much greater variation in results between different runs, with the exception of the accuracy of most complex case. These findings are reflected and supported by the aforementioned thrust profile behaviour.

In conclusion, the trajectories investigated show a similar result in terms of accuracy and optimality. 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.

5.2. Future Work

Future developments for the comparison of convex solvers for low-thrust trajectory optimisation go hand in hand with the development process of CVXPYgen. This thesis has proven the great versatility and benefits of using this software to produce compiled code. Nevertheless, CVXPYgen is a library that is still under heavy development, and this can be exploited to fill the gaps of the presented study such as the number of solvers adopted or the complexity of the test cases analysed.

Under a solver point of view, the introduction of CLARABEL [1] (see Subsection 2.3.2) as the main solver for CVXPYgen is underway, and is supposed to be released in the next version of the software. This will allow to study the behaviour of another interior-point method and probe the results obtained. Although already proven with CVXPY, the use of an additional compiled code will verify or refute the results obtained with ECOS from a computational point of view. Another interesting aspect to expand upon would be the addition of more first-order methods that support second-order cone constraints. Moreover, active-set methods could be added to CVXPYgen to test their performance and determine whether they would be suitable for onboard applications.

The performance of the methods can also be pushed by investigating problems of larger sizes through the introduction of a greater number of nodes. This can be done by switching software or waiting for the release of the next version of CVXPYgen, which is intended to vastly improve in this aspect. Due to the novel nature of CVXPYgen, a limitation has been found in this study for the maximum number of parameters available for the problems, resulting in an upper limit of 150 nodes. This led to the selection of the Earth-to-Venus case as the most complex trajectory for providing solutions comparable to the literature. Analysing the behaviour of the methods with more intricate trajectories like

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

the Earth-to-Dionysus one could be of interest to understand the methods better and verify their use in onboard applications. The analysis of a more complex scenario can be extremely useful in understanding if the FOM solver behaves better in terms of thrust-profile consistency and accuracy when compared to the IPM solver for more complex trajectories, as observed in this study.

Another prospective direction for future research involves evaluating solver performance in scenarios with different dynamics, such as the planetary landing problem. The synthesis of the literature review and this study demonstrate the profound utility of convex methods across a variety of space applications. Leveraging these methods holds the promise of advancing our understanding and exploration of the solar system.

List of Figures

2.1 Taxonomy of optimisation technique families. 20

3.1 Spherical reference frame adopted. 32
3.2 Control constraint in 2D space. 40

4.1 Effect of the trust region factor on computational loads. 55
4.2 Effect of the trust region factor on complexity. 56
4.3 Effect of the trust region factor on accuracy. 57
4.4 Effect of SCS operations limit on computational loads. 58
4.5 Effect of SCS operations limit on complexity. 58
4.6 Effect of SCS operations limit on accuracy. 59
4.7 Output of results for the Earth–Mars case. 60
4.8 Output of results for the SEL$_2$–NEA case. 62
4.9 Output of results for the Earth–Venus case. 63
4.10 Memory of compiled code. 64
4.11 Iterations and operations of simulations. 66
4.12 Computational efficiency and burden of simulations. 67
4.13 Accuracy of the propagated optimal solution. 68
4.14 Reliability and effectiveness of simulations. 70
4.15 Comparison of thrust profiles. 72

A.1 Effect of the trust region factor on performance for the NEA case. 100
A.2 Effect of the trust region factor on performance for the Venus case. 101
A.3 Effect of SCS operations limit on performance for Earth–Mars. 103
A.4 Effect of SCS operations limit on performance for Earth–Venus. 104

B.1 Effect of nodal number on the accuracy of the solution. 106
B.2 Difference between the SCP propagated and output solutions. 107
B.3 Equivalence of control constraint relaxation for the three test-cases. 108

C.1 Screenshot of GitLab repository. 109

List of Tables

2.1 Comparison of convex solution methods. 17
2.2 Comparison of characteristics of convex solution methods. 18

3.1 Physical constants. 34

4.1 Simulation parameters for the test cases. 52
4.2 Boundary conditions in non-dimensional units. 53
4.3 Description of criteria used. 54
4.4 Problem dimension with test case. 65
4.5 Accuracy in spherical coordinates of the propagated optimal solution.* . . 69
4.6 Optimality of converged solution.* 69
4.7 Standard deviation of propagated solution. 73
4.8 Analysis of solvers in CVXPY (Earth–Mars case).* 74
4.9 Comparison of compiled and non-compiled code (Earth–Mars case).* . . . 75

Nomenclature

Physical properties

α	thrust angle	[rad]
$\boldsymbol{\delta}$	trust region	[m, rad, rad, m/s, m/s, m/s, kg]
$\boldsymbol{r}$	position	[m, rad, rad]
$\boldsymbol{u}$	control	[kgm/s^2, rad, rad]
$\boldsymbol{x}$	state	[m, rad, rad, m/s, m/s, m/s, kg]
Δt	time step	[s]
μ	gravitational constant	[m^3/s^2]
ϕ	azimuth angle	[rad]
τ	control variable	[m/s^2]
θ	elevation angle	[rad]
DU	distance unit	[m]
FU	thrust unit	[kgm/s^2]
g	gravitational acceleration	[m/s^2]
m	mass	[kg]
MU	mass unit	[kg]
r	radial distance	[m]
T	thrust	[kgm/s^2]
t	time	[s]
TU	time unit	[s]
v	velocity	[m/s]

VU	velocity unit	[m/s]
z	pseudo-mass	[–]

Other parameters

$\boldsymbol{\lambda}$	Lagrange multiplier	[–]
$\boldsymbol{\nu}$	search parameter	[–]
$\boldsymbol{\phi}$	barrier function	[–]
$\boldsymbol{\Upsilon}$	Cauchy parameter	[–]
$\boldsymbol{A}$	dynamics derivative	[–]
$\boldsymbol{a}$	polynomial constant	[–]
$\boldsymbol{B}$	control matrix	[–]
$\boldsymbol{b}$	trinomial constant	[–]
$\boldsymbol{c}$	binomial constant	[–]
$\boldsymbol{d}$	monomial constant	[–]
$\boldsymbol{f}$	dynamics	[–]
$\boldsymbol{g}$	constraint function	[–]
$\boldsymbol{H}$	Hessian	[–]
$\boldsymbol{h}$	dynamics slack variable	[–]
$\boldsymbol{L}$	augmented Lagrange function	[–]
$\boldsymbol{l}$	Lagrange dual function	[–]
$\boldsymbol{Q}$	feasible set	[–]
$\boldsymbol{S}$	$\boldsymbol{s}$ diagonal matrix	[–]
$\boldsymbol{s}$	slack variable	[–]
$\boldsymbol{y}$	Lagrange multiplier of equality constraint	[–]
$\boldsymbol{Z}$	$\boldsymbol{z}$ diagonal matrix	[–]
$\boldsymbol{z}$	Lagrange multiplier of inequality constraint	[–]
χ	duality measure	[–]

ϵ	accuracy	[−]
η	control slack variable	[−]
γ	trust region factor	[−]
ι	path parameter	[−]
κ	scaling factor	[−]
$\mathbb{R}$	real	[−]
ρ	splitting-method penalty term	[−]
σ	centring parameter	[−]
τ	control variable	[−]
ε	convergence criteria	[−]
ζ	barrier parameter	[−]
C	dynamics weight factor	[−]
c	dynamisc constant	[−]
D	control weight factor	[−]
J	objective function	[−]
N	nodes	[−]
R	reliability	[−]

Subscripts/Superscripts

$*$	reference
ϕ	elevation component
θ	azimuth component
0	initial
aff	affine-scaling
c	constraint
e	escape
f	final

i	node
k	SCP iteration
max	maximum
r	radial component
x	state
z	pseudo-mass

Acronyms

ADMM	Alternating Direction Method of Multipliers
ASM	Active Set Method
AU	Astronomical Unit
BC	Boundary Condition
CPU	Computational
DCP	Disciplined Convex Programming
DPP	Disciplined Parameterised Programming
DU	Distance Unit
FOH	First-Order-Hold
FOM	First Order Method
FU	Thrust Unit
IPM	Interior Point Method
KKT	Karush-Kuhn-Tucker
LGL	Legrendre-Gauss-Lobatto
LP	Linear Program
MC	Monte Carlo
MPC	Model Predictive Control
MU	Mass Unit
NEA	Near-Earth-Asteroid

NLP	Non-Linear Program
QCQP	Quadratically constrained quadratic program
QP	Quadratic Program
RSS	Resident Set Size
SCP	Sequential Convex Program
SDP	Semidefinite Program
SEL	Sun-Earth Lagrange
SOC	Second-Order Cone
SOCP	Second-Order Cone Program
TU	Time Unit
VU	Velocity Unit

Bibliography

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

[2] Tsiotras Panagiotis and Mesbahi Mehran. Toward an algorithmic control theory. *Journal of Guidance, Control, and Dynamics*, 40:1–3, 02 2017. `doi:10.2514/1.G002754`.

[3] Morante David, Sanjurjo Rivo Manuel, and Soler Manuel. A survey on low-thrust trajectory optimization approaches. *Aerospace*, 8(3), 03 2021. `doi:10.3390/AEROSPACE8030088`.

[4] Danylo Malyuta, Yue Yu, Purnanand Elango, and Behçet Açikmeşe. Advances in trajectory optimization for space vehicle control. *Annual Reviews in Control*, 52:282–315, 2021. `doi:10.1016/j.arcontrol.2021.04.013`.

[5] Nurre Nicholas and Taheri Ehsan. Comparison of indirect and convex-based methods for low-thrust minimum-fuel trajectory optimization. In *2022 AAS/AIAA Astrodynamics Specialist Conference*, Charlotte, NC, USA, 08 2022.

[6] E. Trélat. Optimal control and applications to aerospace: Some results and challenges. *Journal of Optimization Theory and Applications*, 154(3):713–758, 09 2012. `doi:10.1007/s10957-012-0050-5`.

[7] Liu Xinfu, Lu Ping, and Pan Binfeng. Survey of convex optimization for aerospace applications. *Astrodynamics*, 1(1):23–40, 2017. `doi:10.1007/s42064-017-0003-8`.

[8] Danylo Malyuta, Taylor P. Reynolds, Michael Szmuk, Thomas Lew, Riccardo Bonalli, Marco Pavone, and Behcet Acikmese. Convex optimization for trajectory generation, 06 2021. `arXiv:2106.09125`.

[9] Morelli Andrea Carlo, Hofmann Christian, and Topputo Francesco. Robust low-thrust trajectory optimization using convex programming and a homotopic approach.

IEEE Transactions on Aerospace and Electronic Systems, 58(3):2103–2116, 11 2022. `doi:10.1109/TAES.2021.3128869`.

[10] Zhenbo Wang and Michael Grant. Constrained trajectory optimization for planetary entry via sequential convex programming. *Journal of Guidance, Control, and Dynamics*, 40:1–13, 07 2017. `doi:10.2514/1.G002150`.

[11] Zhenbo Wang and Michael J. Grant. 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`.

[12] Danylo Malyuta et al. Starship landing scp example. Link: `https://github.com/dmalyuta/scp_traj_opt`. Last access 08 08 2023.

[13] NASA. Nasa tipping point partnership with blue origin to test precision lunar landing technologies. Link: `https://www.nasa.gov/directorates/spacetech/NASA_Tipping_Point_Partnership_to_Test_Precision_Lunar_Landing_Tech/`. Last access on 08 08 2023.

[14] Boyd Stephen P and Vandenberghe Lieven. *Convex optimization.* Cambridge university press, 2004.

[15] Nocedal Jorge and Wright Stephen J. *Numerical optimization.* Springer, 1999.

[16] Nababithi Goswami, Supriyo K. Mondal, and Swapan Paruya. A comparative study of dual active-set and primal-dual interior-point method. *IFAC Proceedings Volumes*, 45(15):620–625, 07 2012. `doi:10.3182/20120710-4-SG-2026.00029`.

[17] Bartlett R.A., Wachter A., and Biegler L.T. Active set vs. interior point strategies for model predictive control. *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat)*, 6:4229–4233, 06 2000. `doi:10.1109/ACC.2000.877018`.

[18] Lau Mark S. K., Yue S. P., Ling K. V., and Maciejowski J. M. A comparison of interior point and active set methods for fpga implementation of model predictive control. *European Control Conference (ECC)*, 1:156–161, 08 2009. `doi:10.23919/ECC.2009.7074396`.

[19] Maximilian Schaller, Goran Banjac, Steven Diamond, Akshay Agrawal, Bartolomeo Stellato, and Stephen Boyd. Embedded code generation with CVXPY. *IEEE Control Systems Letters*, 6:2653–2658, 2022. `doi:10.1109/LCSYS.2022.3173209`.

[20] Wong Elizabeth and Philip E. Gill. *Active-Set Methods for Quadratic Programming.* Phd thesis, University Of California, San Diego, 2011.

[21] Bemporad Alberto. Lecture notes in model predictive control - quadratic programming and explicit mpc, 2023. Scuola IMT Alti Studi Lucca.

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

[23] John T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming.* Society for Industrial and Applied Mathematics, second edition, 01 2010. `doi:10.1137/1.9780898718577`.

[24] Benyamin Ghojogh, Ali Ghodsi, Fakhri Karray, and Mark Crowley. Kkt conditions, first-order and second-order optimization, and distributed optimization: Tutorial and survey, 10 2021. `arXiv:2110.01858`.

[25] Irvin Lustig. *A Practical Approach to Karmarkar's Algorithm.* Systems Optimization Laboratory, Department of Operations Research, Stanford University, 06 1985.

[26] Karmarkar Narendra. A new polynomial-time algorithm for linear programming-II. *Combinatorica*, 4:373–395, 12 1984. `doi:10.1007/BF02579150`.

[27] Karmarkar Narendra. Seminar presentation at stanford university. 01 1985.

[28] Mehrotra Sanjay. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2:575–601, 11 1992. `doi:10.1137/0802028`.

[29] Pavel Dvurechensky, Shimrit Shtern, and Mathias Staudigl. First-order methods for convex optimization. *EURO Journal on Computational Optimization*, 9:100015, 2021. `doi:10.1016/j.ejco.2021.100015`.

[30] H.J. Ferreau, S. Almér, R. Verschueren, M. Diehl, D. Frick, A. Domahidi, J.L. Jerez, G. Stathopoulos, and C. Jones. Embedded optimization methods for industrial automatic control. *IFAC-PapersOnLine*, 50(1):13194–13209, 2017. `doi:10.1016/j.ifacol.2017.08.1946`.

[31] Goldfarb Donald. *Advances in Optimization and Numerical Analysis.* Springer Netherlands, 01 1992.

[32] Karl Heinx Borgwardt. Probabilistic analysis of the simplex method. In *DGOR/NSOR*, Berlin, Heidelberg, 1988. Springer. `doi:10.1007/978-3-642-73778-7$\_$148`.

[33] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. *J. ACM*, 51:385–463, 07 2001. `doi:10.1145/380752.380813`.

[34] Cimini Gionata and Bemporad Alberto. Exact complexity certification of active-set methods for quadratic programming. *IEEE Transactions on Automatic Control*, 62(12):6094–6109, 12 2017. doi:10.1109/TAC.2017.2696742.

[35] Daniel Arnström and Daniel Axehill. A unifying complexity certification framework for active-set methods for convex quadratic programming. *IEEE Transactions on Automatic Control*, 67(6):2758–2770, 06 2022. doi:10.1109/TAC.2021.3090749.

[36] Cristóbal Guzmán and Arkadi Nemirovski. On lower complexity bounds for large-scale smooth convex optimization. *Journal of Complexity*, 31(1):1–14, 02 2015. doi:10.1016/j.jco.2014.08.003.

[37] Liu Xinfu, Lu Ping, and Pan Binfeng. Survey of convex optimization for aerospace applications. *Astrodynamics*, 1:23–40, 09 2017. doi:10.1007/s42064-017-0003-8.

[38] Lars Blackmore, Behçet Açikmese, and Daniel P. Scharf. Minimum-landing-error powered-descent guidance for mars landing using convex optimization. *Journal of Guidance Control and Dynamics*, 33(4):1161–1171, 07 2010. doi:10.2514/1.47202.

[39] Behçet Açikmese and Scott R. Ploen. Convex programming approach to powered descent guidance for mars landing. *Journal of Guidance Control and Dynamics*, 30(5):1353–1366, 09 2007. doi:10.2514/1.27553.

[40] Robin M. Pinson and Ping Lu. Rapid generation of optimal asteroid powered descent trajectories via convex optimization. In *AAS/AIAA Astrodynamics Specialist Conference*, Vail, Colorado, USA, 08 2015.

[41] Pinson M. Robin and Lu Ping. Trajectory design employing convex optimization for landing on irregularly shaped asteroids. In *AAS/AIAA Astrodynamics Specialist Conference*, Long Beach, California,USA, 09 2016. doi:10.2514/6.2016-5378.

[42] Ping Lu and Xinfu Liu. Autonomous trajectory planning for rendezvous and proximity operations by conic optimization. *Journal of Guidance Control and Dynamics*, 36(2):375–389, 03 2013. doi:10.2514/1.58436.

[43] Xinfu Liu. *Autonomous trajectory planning by convex optimization.* Phd thesis, Iowa State University, 2013. doi:10.31274/etd-180810-3525.

[44] John Carson, Michelle Munk, Ronald Sostaric, Jay Estes, Farzin Amzajerdian, James Blair, David Rutishauser, Carolina Restrepo, Alicia Dwyer-Cianciolo, George Chen, and Teming Tse. The SPLICE project: Continuing NASA development of GNC technologies for safe and precise landing. In *AIAA Scitech 2019 Forum*, San Diego, California, USA, 01 2019. doi:10.2514/6.2019-0660.

[45] A. Zanelli, A. Domahidi, J. Jerez, and M. Morari. Forces nlp: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs. *International Journal of Control*, 93(1):1–26, 04 2017. `doi:10.1080/00207179.2017.1316017`.

[46] Boris Houska, Jaochim Ferreau, and Mortiz Diehl. ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 03 2011. `doi:10.1002/oca.939`.

[47] Boris Houska, Jaochim Ferreau, and Mortiz Diehl. An Auto-Generated Real-Time Iteration Algorithm for Nonlinear MPC in the Microsecond Range. *Automatica*, 47(10):2279–2285, 10 2011. `doi:10.1016/j.automatica.2011.08.020`.

[48] Pablo Zometa, Markus Kogel, and Rolf Findeisen. μAO-MPC: A free code generation tool for embedded real-time linear model predictive control. In *American Control Conference (ACC)*, Washington D.C., USA, 06 2013. `doi:10.1109/ACC.2013.6580668`.

[49] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 04 2016.

[50] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 10 2018. `doi:0.1080/23307706.2017.1397554`.

[51] Maximilian Schaller, Goran Banjac, Steven Diamond, Akshay Agrawal, Bartolomeo Stellato, and Stephen Boyd. Embedded code generation with CVXPY. ACCESS Seminar, 03 2022.

[52] Alexander Domahidi, Eric Chu, and Stephen Boyd. ECOS: An SOCP solver for embedded systems. In *2013 European Control Conference, ECC 2013*, 07 2013. `doi:10.23919/ECC.2013.6669541`.

[53] Martin Andersen, Joachim Dahl, and Lieven Vandenberghe. CVXOPT: A python package for convex optimizatio. version 1.3.1. `https://cvxopt.org/documentation/index.html`, 03 2023. Last access 25 04 2023.

[54] Dongdong Ge, Qi Huangfu, Zizhuo Wang, Jian Wu, and Yinyu Ye. Cardinal Optimizer (COPT) user guide. https://guide.coap.online/copt/en-doc, 2023. Last access 21 04 2023.

[55] Abhishek Goud Pandala, Yanran Ding, and Hae-Won Park. qpswift: A real-time

sparse quadratic program solver for robotic applications. *IEEE Robotics and Automation Letters*, 4(4):3355–3362, 10 2019. `doi:10.1109/LRA.2019.2926664`.

[56] Paul Goulart and Yuwen Chen. Clarabel: A library for optimization and control. Link: `https://oxfordcontrol.github.io/ClarabelDocs/stable/`, 2021. Last access on 08 08 2023.

[57] Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 06 2016. `doi:10.1007/s10957-016-0892-3`.

[58] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 12 2020. `doi:10.1007/s12532-020-00179-2`.

[59] Antoine Bambade, Sarah El-Kazdadi, Adrien Taylor, and Justin Carpentier. PROX-QP: Yet another Quadratic Programming Solver for Robotics and beyond. In *RSS 2022 - Robotics: Science and Systems*, New York, United States, 06 2022. URL: `https://hal.inria.fr/hal-03683733`.

[60] Michael Garstka, Mark Cannon, and Paul Goulart. COSMO: A conic operator splitting method for convex conic problems. *Journal of Optimization Theory and Applications*, 190(3):779–810, 2021. `doi:10.1007/s10957-021-01896-x`.

[61] Yue Yu, Purnanand Elango, and Behçet Açı kmeşe. Proportional-integral projected gradient method for model predictive control. *IEEE Control Systems Letters*, 5(6):2174–2179, 2021. `doi:10.1109/LCSYS.2020.3044977`.

[62] Hans Joachim Ferreau, Hans Georg Bock, and Moritz Dieh. An online active set strategy to overcome the limitations of explicit MPC. *International Journal of Robust and Nonlinear Control*, 18(8):816–830, 01 2008. `doi:10.1002/rnc.1251`.

[63] Hans Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Georg Bock, and Moritz Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 12 2014. `doi:10.1007/s12532-014-0071-1`.

[64] Daniel Arnström, Alberto Bemporad, and Daniel Axehill. A dual active-set solver for embedded quadratic programming using recursive LDL^T updates. *IEEE Transactions on Automatic Control*, 67(8):4362–4369, 2022. `doi:10.1109/TAC.2022.3176430`.

[65] Q. Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10:119–142, 04 2018. doi:10.1007/s12532-017-0130-5.

[66] Donald Goldfarb and Ashok Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, 27:1–33, 09 1983. doi:10.1007/BF02591962.

[67] Mattias Fält and Pontus Giselsson. QPDAS: Dual Active Set Solver for Mixed Constraint Quadratic Programming. In *IEEE 58th Conference on Decision and Control (CDC)*, Nice, France, 12 2019. doi:10.1109/CDC40024.2019.9029900.

[68] John T. Betts. Very low-thrust trajectory optimization using a direct sqp method. *Journal of Computational and Applied Mathematics*, 120(1):27–40, 08 2000. doi:10.1016/S0377-0427(00)00301-0.

[69] Yuanqi Mao, Michael Szmuk, Xiangru Xu, and Behcet Acikmese. Successive convexification: A superlinearly convergent algorithm for non-convex optimal control problems. *arXiv preprint*, 2019. doi:10.48550/arXiv.1804.06539.

[70] Francesco Topputo, Yang Wang, Carmine Giordano, Vittorio Franzese, Hannah Goldberg, Franco Perez-Lissi, and Roger Walker. Envelop of reachable asteroids by M-ARGO CubeSat. *Advances in Space Research*, 67(12):4193–4221, 2021. doi:10.1016/j.asr.2021.02.031.

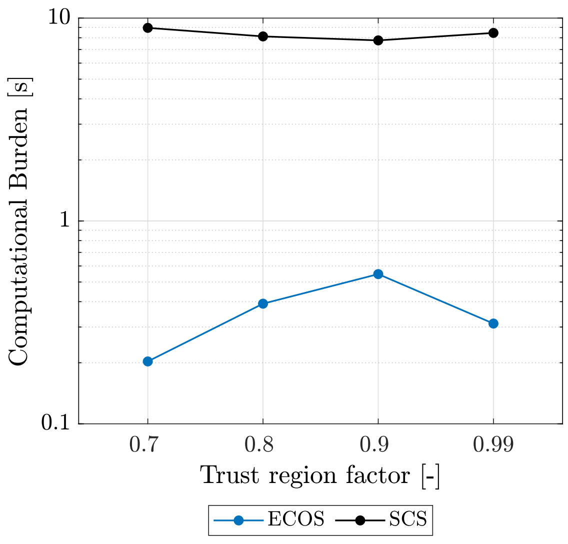
A | Parametric Investigation

A.1. Trust Region Factor

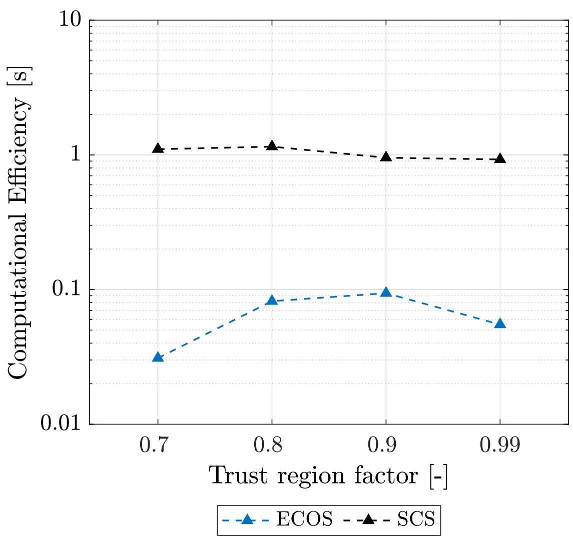
As the trust region factor γ depends on the algorithm adopted and is not a parameter which is specific to the solver, the effort was made to investigate its effect in all cases and for all solvers used. This was done to try and guarantee the best possible result and solver performance for ECOS and SCS, in order to compare the most optimal solutions. To complete the investigation reported in Section 4.2 for the Earth–Mars trajectory, the remaining cases are illustrated below. The parameters reported in the figures are a representations of the criteria introduced in Section 2.4 and described in Table 4.3.

For the SEL$_2$-NEA case, as shown in Figure A.1, the effect that the trust region factor has on the computational toll for SCS is almost negligible, whereas a value of 0.7 guarantees the best computational burden for ECOS. Analysing the complexity of the operations done, one can see that a parameter of 0.8 proves to be extremely beneficial to SCS, reducing the number of operations solved by one order of magnitude compared to a value of γ of 0.7 or 0.9. The accuracy, instead, results to be unaffected for ECOS, while it exhibits a decreasing behaviour for SCS as the parameter is increased. A value of 0.8 was therefore chosen for the NEA.

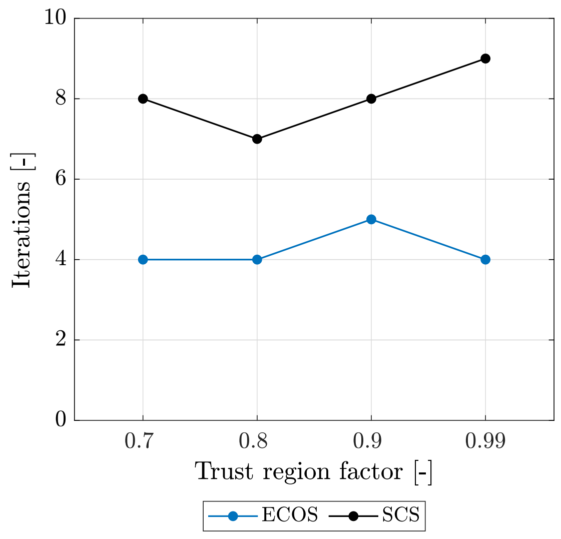
The Earth-to-Venus trajectory optimisation in Figure A.2 presents different results from the previous two. ECOS shows similar outputs for the different values of γ. The same can be said for SCS from the computational and operational point of view. The two values that seemed more promising were of 0.8 and 0.99, due to the lower computational burden for ECOS. However, after analysing the accuracy of the solutions obtained, it was clear that a value of 0.99 was more suitable due to the similar behaviour of the two solvers.



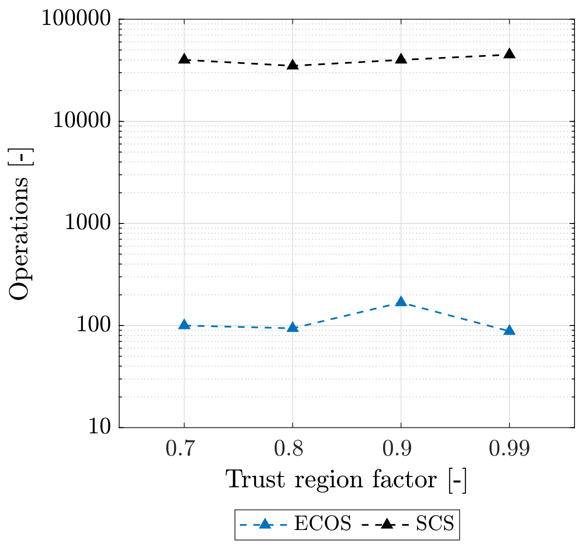
(a) Burden.



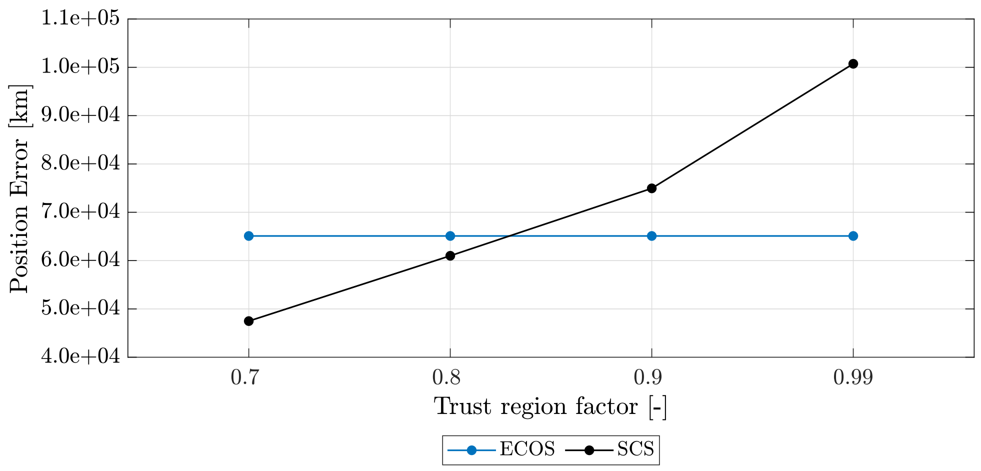
(b) Efficiency.



(c) Iterations.

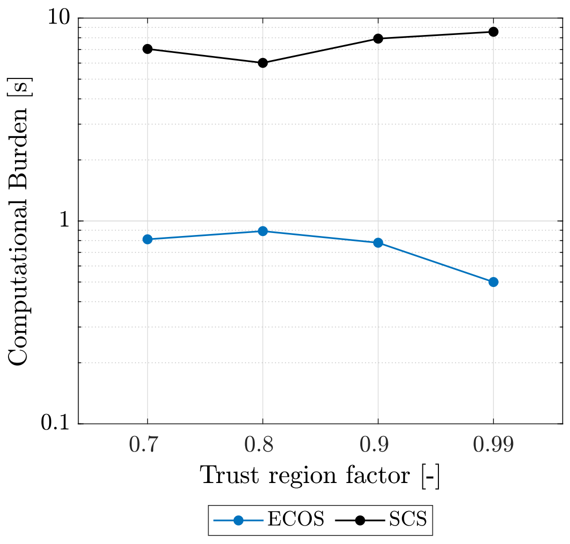


(d) Operations.

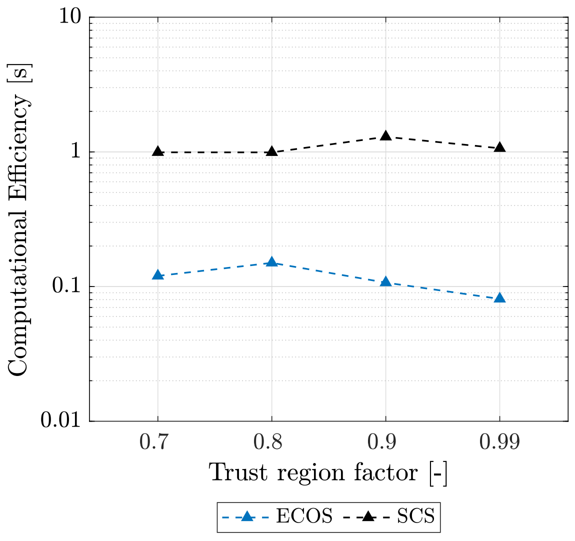


(e) Accuracy.

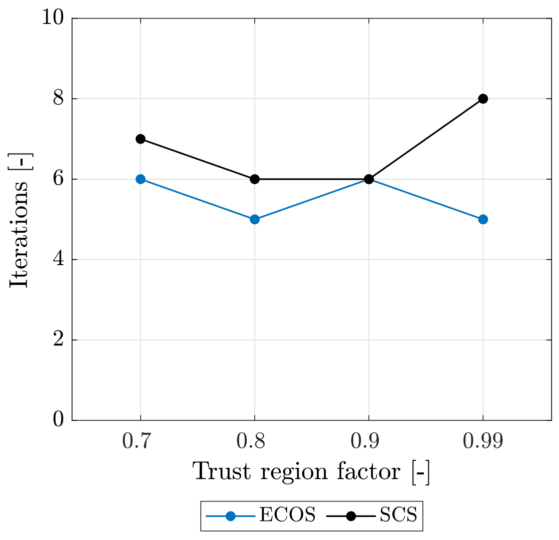
Figure A.1: Effect of the trust region factor on performance for the NEA case.



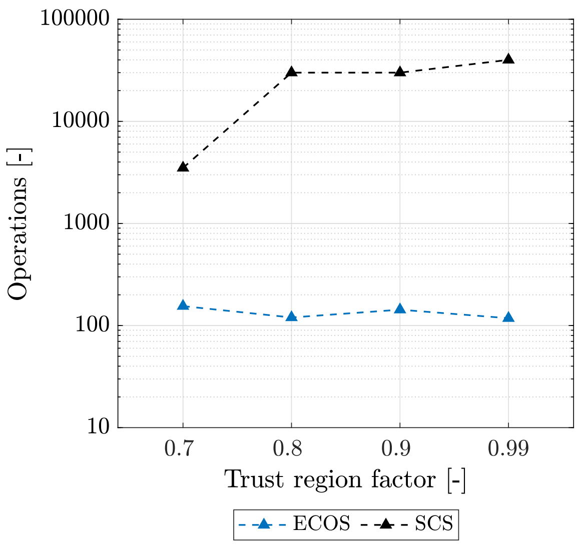
(a) Burden.



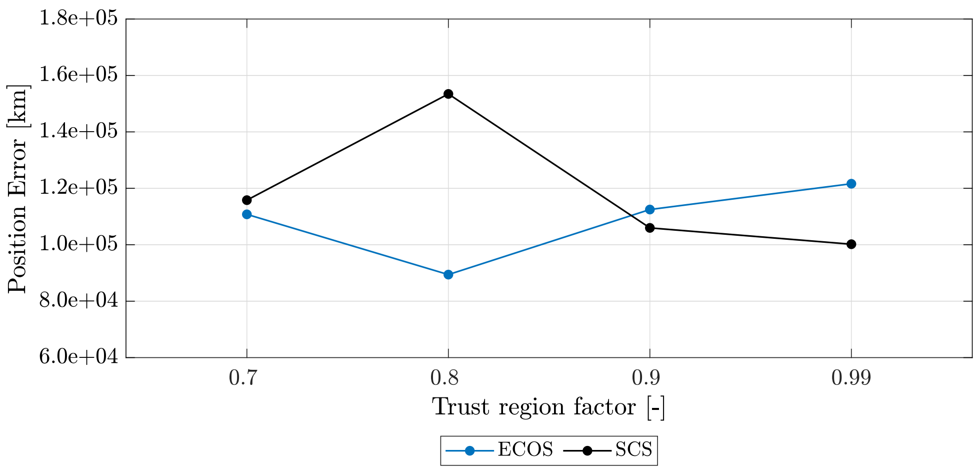
(b) Efficiency.



(c) Iterations.



(d) Operations.



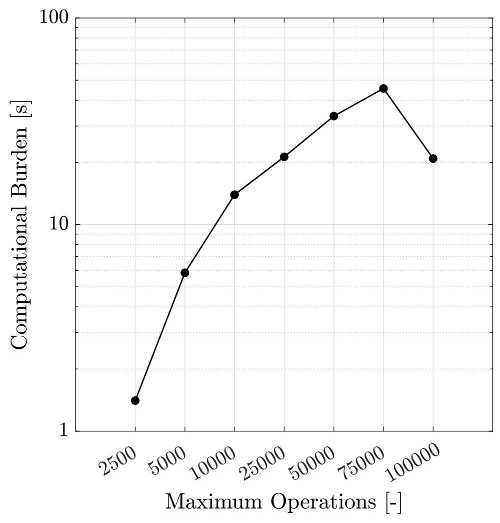
(e) Accuracy.

Figure A.2: Effect of the trust region factor on performance for the Venus case.

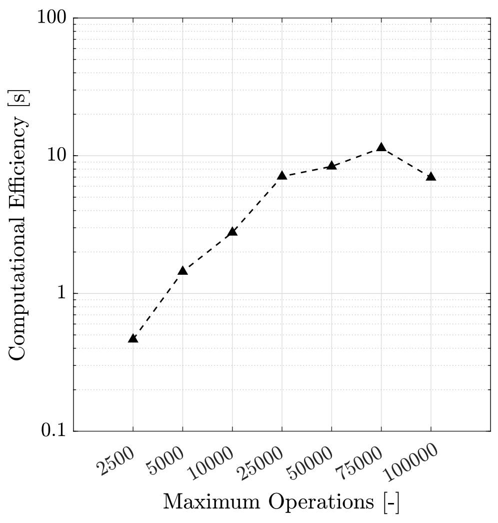
A.2. SCS Operations

As mentioned in Section 4.2, the investigation on the maximum number of solver iterations was carried out for all three test cases. This section of the appendix reports the findings for the Earth-to-Mars (Figure A.3) and Earth-to-Venus (Figure A.4) cases which show the similar behaviour of the parameters as the SEL$_2$-to-NEA example. The parameters reported in the figures are a representation of the criteria introduced in Section 2.4 and described in Table 4.3.

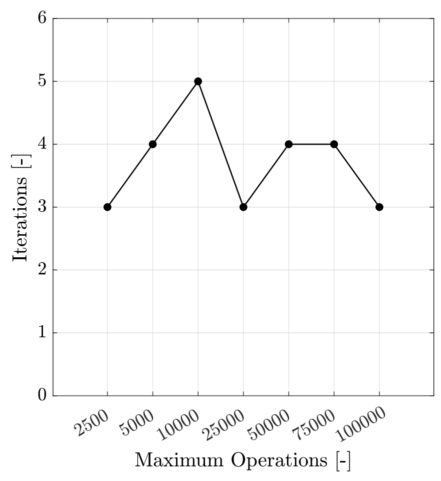
The behaviour of the computational load is similar for all cases, with an increase in values observed with an increase in allowable operations. Only the Earth-to-Mars case shows a minor improvement in the criteria at the last value compared to the previous one. The same can be said for the number of total operations required to reach convergence for SCS (Figure A.3d and Figure A.4d). Moreover, the accuracy of the solution once again shows an unclear behaviour, with a limit of 5000 operations for the SCS solver resulting in the best solution in terms of accuracy for the Mars case, and the second-best for the Venus case. Due to the worsening behaviour of the computational load and complexity with allowable operations and lack of a clear relationship for the accuracy of the maximum number of operations was set to 2500 as explained in Section 4.2.



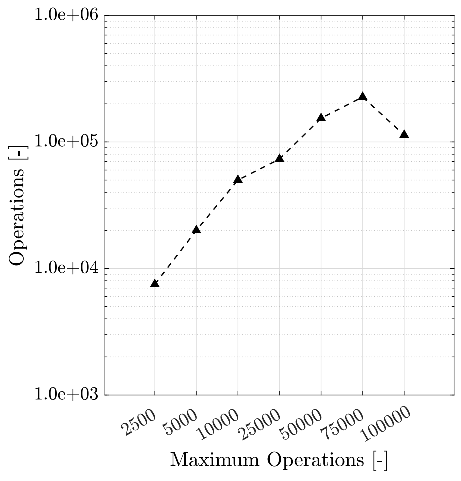
(a) Burden.



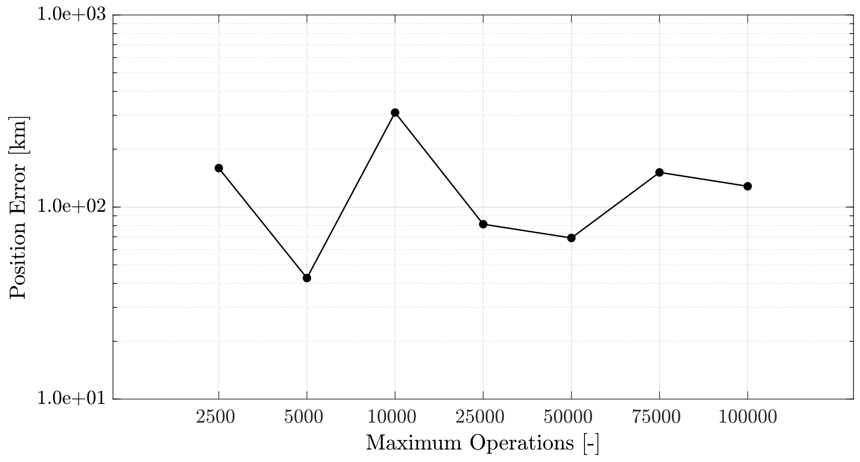
(b) Efficiency.



(c) Iterations.

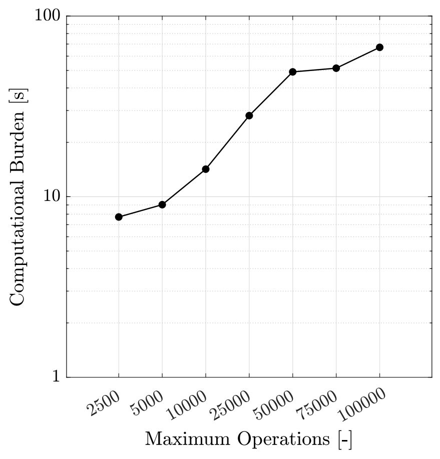


(d) Operations.

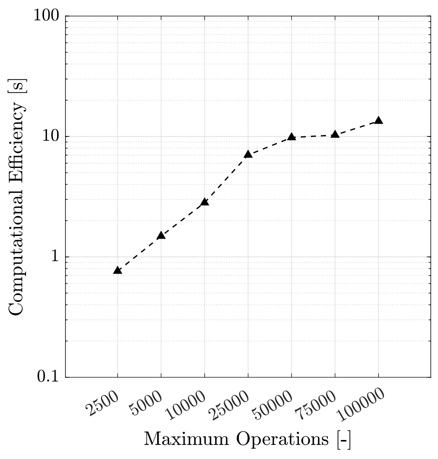


(e) Accuracy.

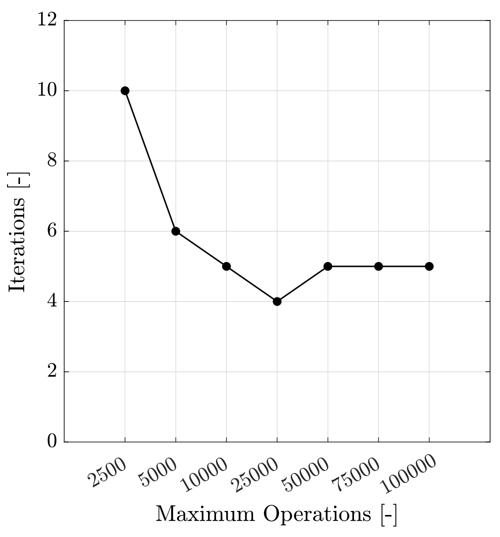
Figure A.3: Effect of the maximum number of allowable SCS operations on performance for the Mars case.



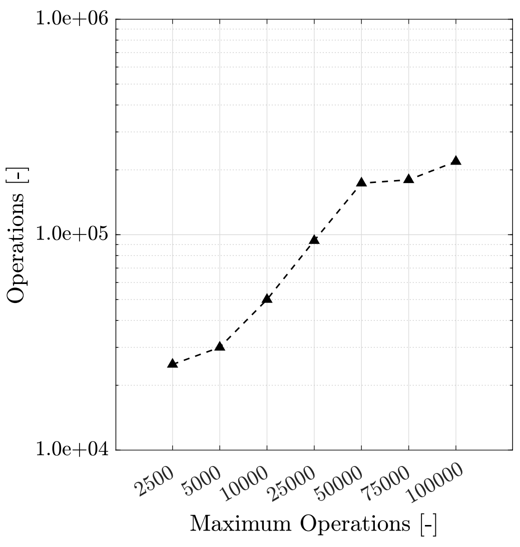
(a) Burden.



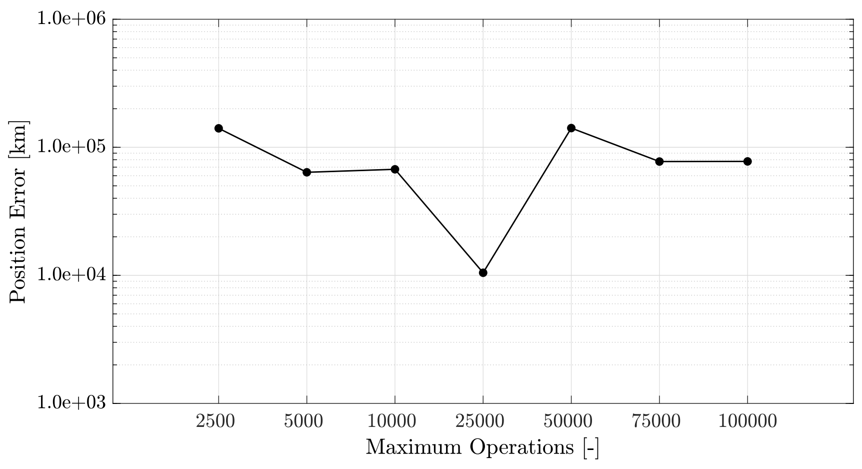
(b) Efficiency.



(c) Iterations.



(d) Operations.



(e) Accuracy.

Figure A.4: Effect of the maximum number of allowable SCS operations on performance for the Venus case.

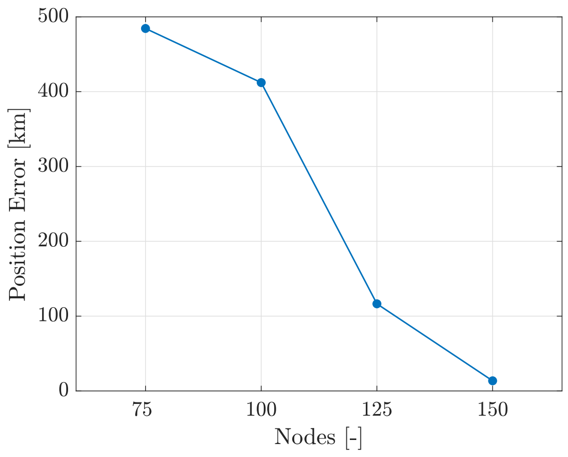
B | Verification and Validation

In order to ensure that the results obtained were correct, a series of investigations and comparisons were performed. This chapter explores and presents said tests together with the results.

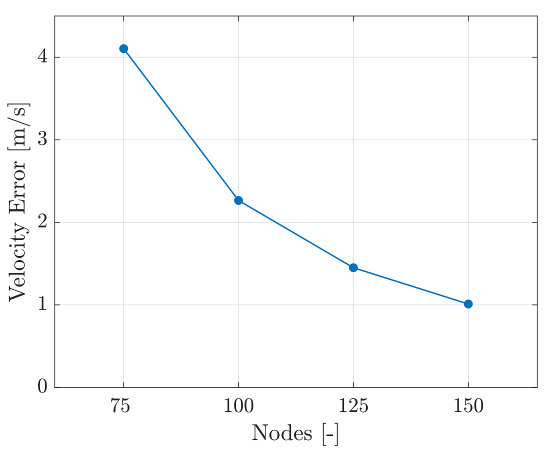
B.1. Accuracy Verification

Initially after obtaining the first results the nodes of the problems were varied progressively to ensure that the high accuracy observed was due to the number of nodes used during the trajectory optimisation and not due to an erroneous problem definition. As the accuracy of the solution (see Table 4.3 for definition) depends on the state of the propagated solution, one has to analyse the radial position, velocity and angles to have a complete evaluation (see Figure B.1). The accuracy reported is obtained by propagating the optimal control and comparing the difference between the desired final state, $\boldsymbol{x}_f$, and the propagated solution at time t_f.

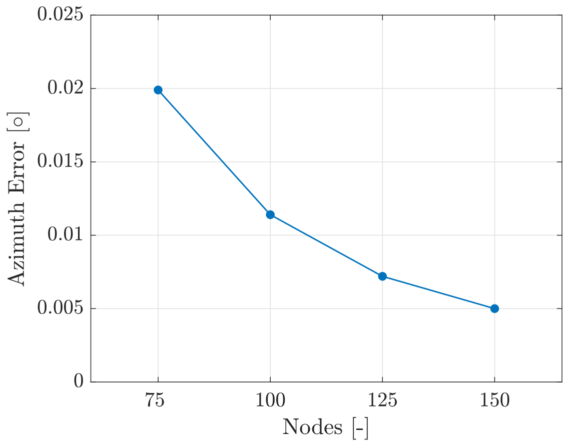
As it is possible to see from the figure below, increasing the number of nodes improves the accuracy of the solution. The pattern observed is similar for the velocity and angles. Although the relationship between the accuracy in the radial position of the spacecraft and the number of nodes differs from the other parameters, an improvement in the values can still be seen. This validates that the low accuracy obtained, especially in the r, is attributed to the node number and not a wrong formulation of the convex problem.



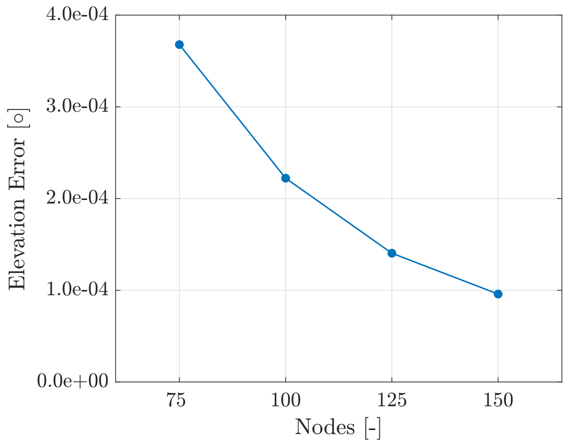
(a) Distance.



(b) Velocity.



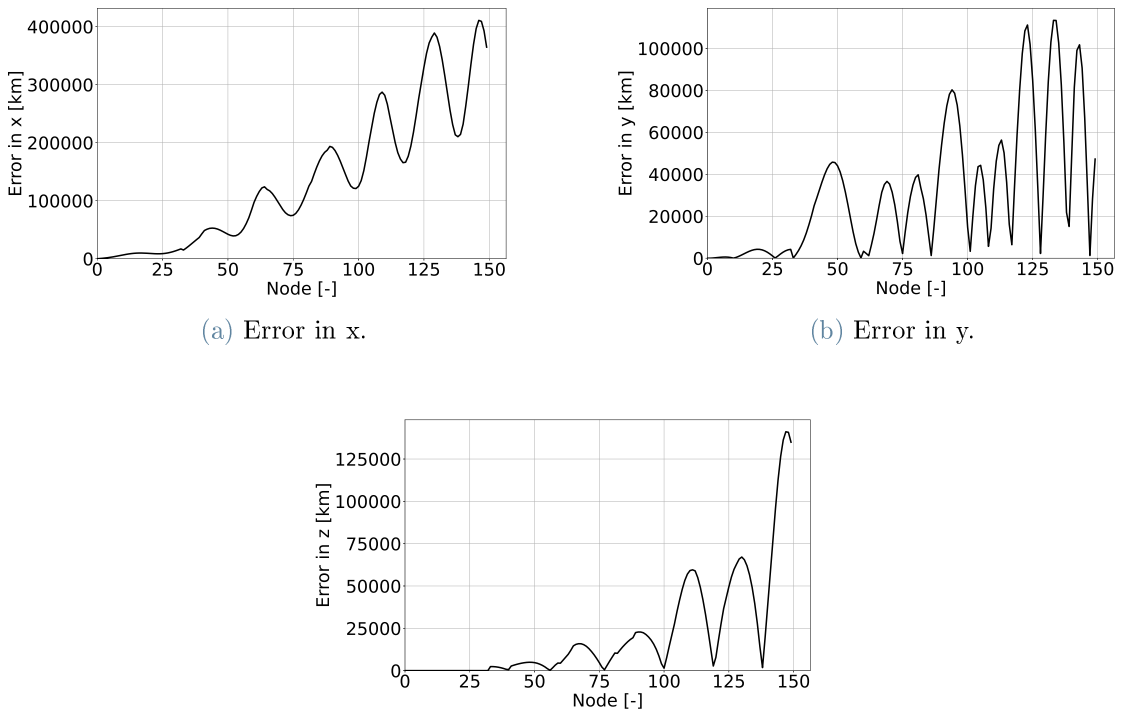
(c) Azimuth.



(d) Elevation.

Figure B.1: Effect of the number of nodes on the accuracy of the propagated solution for the Earth-Mars case using ECOS.

Moreover, by representing the accuracy of the radial distance in Cartesian coordinates, it is clear that the difference in the obtained optimal solution ($\boldsymbol{x}^{opt}$) and the one propagated using the optimal control is due to a propagation error. In fact, the absolute of the error increases with the node i, as shown in Figure B.2.

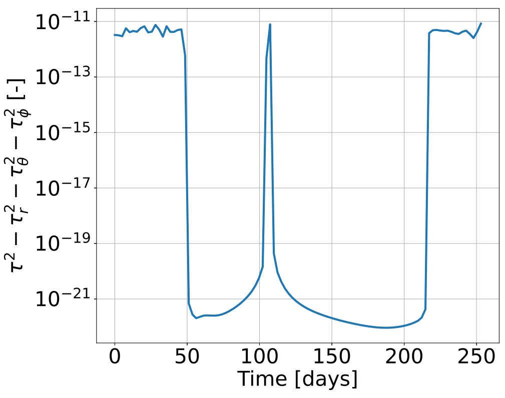


(a) Error in x. (b) Error in y. (c) Error in z.

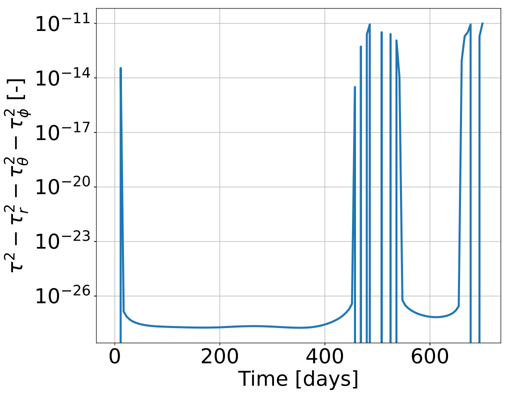
Figure B.2: Difference between the propagated optimal solution and the SCP trajectory output for the Venus case.

B.2. Control Validation

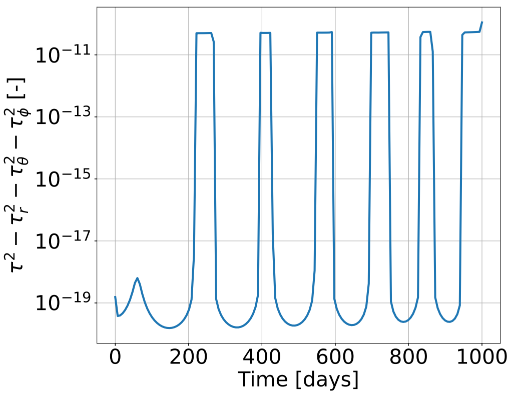
Using the obtained optimal control from the SCP algorithm, it is possible to plot the control relaxation by taking the difference between the left-hand side and the right-hand side of Equation 3.30. Figure B.3 demonstrates that this difference stays extremely close to zero, and therefore the constraint is at the very upper limit of the inequality. This proves that the transformation of the constraint to a convex form respects the equality of Equation 3.18, with the relaxed control constraint remaining active throughout the whole optimised trajectory [11].



(a) Earth-Mars.



(b) SEL_2-NEA.



(c) Earth-Venus.

Figure B.3: Equivalence of control constraint relaxation for the three test-cases.

C | GitLab Repository

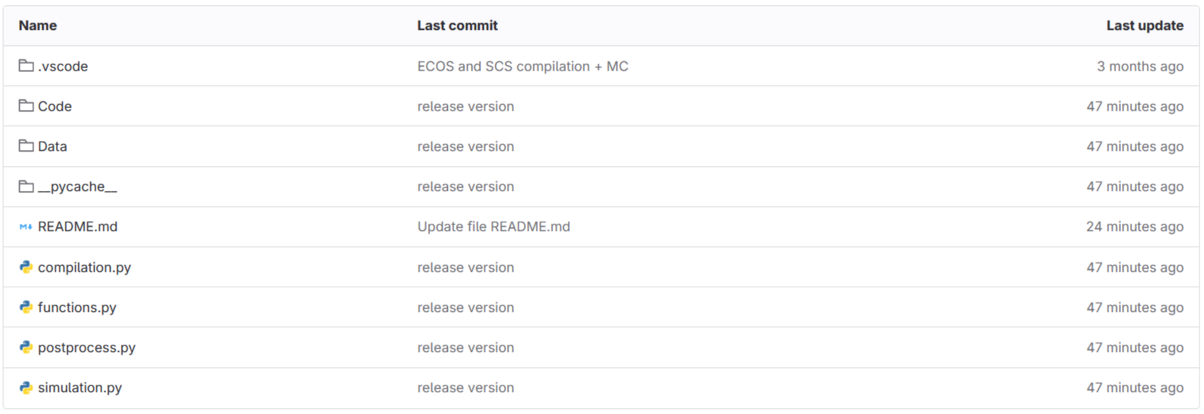
This appendix provides reference to the GitLab repository associated with this thesis. The repository contains all the source code, data sets, and additional materials used in the research discussed in the main text. Readers can access and download these resources to gain a deeper understanding of the research methodology and to reproduce the experiments conducted and the results presented.

GitLab Repository URL: `https://gitlab.com/leogrillo/cvx_thesis`.

The GitLab repository (Figure C.1) contains the following resources:

- Source code for the experimental software;
- Data-sets used for analysis;
- Documentation on how to set up data files;
- Documentation on how to run the experiments.

To access the resources in this GitLab repository, visit the provided URL. Detailed instructions on how to use the code are available in the `README.md` file within the repository.



Name	Last commit	Last update
.vscode	ECOS and SCS compilation + MC	3 months ago
Code	release version	47 minutes ago
Data	release version	47 minutes ago
__pycache__	release version	47 minutes ago
README.md	Update file README.md	24 minutes ago
compilation.py	release version	47 minutes ago
functions.py	release version	47 minutes ago
postprocess.py	release version	47 minutes ago
simulation.py	release version	47 minutes ago

Figure C.1: Screenshot of GitLab repository.