FAST SEQUENTIAL CONVEX PROGRAMMING WITH RELAXED CONVERGENCE CRITERIA

Alexander Popov[(1)], **Cristiano Contini**[(1)], **Mattia Zamaro**[(1)]

[(1)]*Airbus Defence and Space Ltd.*
Gunnels Wood Rd, Stevenage, SG1 2AS, United Kingdom
+44 1438 773000
alexander.popov@airbus.com
cristiano.contini@airbus.com
mattia.zamaro@airbus.com

ABSTRACT

In this paper, we recommend two related strategies which can enable fast termination of sequential convex programming algorithms. We first suggest a simple Hermite interpolation scheme to represent the state solution, and we propose a broad condition for which the constructed solution can be feasible when implementing the algorithm with relaxed convergence criteria. We also present a mesh refinement framework to ensure that this continuous-time trajectory satisfies the system dynamics to a desired level of accuracy. This motivates an amended stopping condition for sequential convex programming, using an integrated residual term as an additional measure of error in the solution. The benefits of this framework are demonstrated using the case study of guidance for an autonomous lunar lander with six degrees of freedom.

1 INTRODUCTION

In recent years, sequential convex programming (SCP) has emerged as a powerful class of methods for solving nonlinear trajectory optimisation problems [1]. These algorithms solve a series of approximate convex optimisation problems, for which there already exist a variety of fast and robust solution methods [2]. Because of the relatively low computational cost of each iteration, SCP schemes are potentially suitable for implementation on embedded systems with strict running time requirements. A number of realisations of SCP have been proposed, boasting impressive results from numerical experiment. Perhaps the two most prominent examples are SCvx [3] and GuSTO [4], both of which are also supported by rigorous theoretical convergence analysis. However, SCP methods remain relatively immature compared to classical techniques such as direct collocation, and one area that has received very little practical consideration is the question of when an SCP algorithm should be terminated. This can pose a difficulty for the designer, as a compromise must be made between running time and solution accuracy. Additionally, a closely related question is how best to represent the extracted solution.

In this paper, we address these questions by recommending a pair of strategies that can allow for greater flexibility in the design of an SCP algorithm, specifically for implementations where the desired output is a reference state trajectory to be followed using classical control techniques. This represents a wide range of possible applications, such as on-board computation of optimal guidance

trajectories for spacecraft [5]. We argue the case for applying Hermite interpolation to represent the state trajectory solution, and we suggest that this approach can be combined with relatively relaxed stopping criteria on the algorithm, even if this results in some inconsistency between the state solution and the control solution. In particular, we justify this approach in the case when terminal state constraints should be satisfied exactly. We additionally endorse the use of a mesh refinement scheme for a wide range of problems, and we suggest a tailored framework for its incorporation in SCP algorithms. By using such an approach, it can become considerably easier to choose appropriate stopping criteria for a given application.

The remainder of this paper is organised as follows: Section 2 briefly introduces some fundamental concepts in trajectory optimisation and SCP. In Section 3, we discuss methods for representing the continuous-time solution derived from an SCP implementation, and we describe the recommended interpolation scheme. Following this, Section 4 describes the variable-resolution framework using a simple mesh refinement routine. We illustrate the effectiveness of these strategies in Section 5, via a lunar lander case for which we present numerical results. Section 6 concludes the paper.

2 PRELIMINARIES

2.1 Trajectory Optimisation

A trajectory optimisation problem aims to find the optimal evolution of a dynamical system over a time interval $\mathcal{T} \coloneqq [t_0, t_f] \subset \mathbb{R}$, such that a solution minimises a defined cost function while satisfying a set of constraints. Since t_0 and t_f may be variables, the problem is often formulated over the fixed interval $\widetilde{\mathcal{T}} \coloneqq [0, 1]$, with the time parameterised by the non-dimensional variable t. For any $t \in \widetilde{\mathcal{T}}$, the state vector $x(t) \in \mathbb{R}^{n_x}$ of the system is given by the continuous state trajectory $x : \mathbb{R} \to \mathbb{R}^{n_x}$, and the system is controlled by a trajectory of free variables $u : \mathbb{R} \to \mathbb{R}^{n_u}$. The problem is further characterised by a vector of constant parameters $\theta \in \mathbb{R}^{n_\theta}$, which includes the free boundaries of $\mathcal{T}$ if this interval is not fixed. We now consider a general class of trajectory optimisation problems:

$$\min_{x(\cdot),u(\cdot),\theta} \quad \pi\big(x(0), x(1), \theta\big) + \int_0^1 \rho\big(x(t), u(t), \theta, t\big)dt \tag{1a}$$

$$\text{s.t.} \quad \dot{x}(t) - f\big(x(t), u(t), \theta, t\big) = 0, \ \ \forall t \in \widetilde{\mathcal{T}}, \tag{1b}$$

$$\gamma\big(x(t), u(t), \theta, t\big) \geq 0, \ \ \forall t \in \widetilde{\mathcal{T}}, \tag{1c}$$

$$\psi_E\big(x(0), x(1), \theta\big) = 0, \tag{1d}$$

$$\psi_I\big(x(0), x(1), \theta\big) \geq 0. \tag{1e}$$

Throughout this paper, we treat (1) as an optimal control problem, and so u can equivalently be defined as a trajectory of control inputs. The cost function is written in Bolza form, comprising the Mayer term $\pi : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_\theta} \to \mathbb{R}$ and the integral of the running cost $\rho : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_\theta} \times \mathbb{R} \to \mathbb{R}$. We assume the cost function to be continuous, but not necessarily smooth. The dynamical behaviour of the system (i.e., the dynamics) is described by the differential equations (1b), where $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_\theta} \times \mathbb{R} \to \mathbb{R}^{n_x}$ is continuously differentiable. The inequality constraints (1c) are described by the continuously differentiable mapping $\gamma : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_\theta} \times \mathbb{R} \to \mathbb{R}^{n_\gamma}$, and boundary conditions are captured in (1d) and (1e), where $\psi_E : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_\theta} \to \mathbb{R}^{n_E}$ and $\psi_I : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_\theta} \to \mathbb{R}^{n_I}$. The solution to (1) is the pair of optimal trajectories (x^*, u^*), as well as the optimal parameters θ^*.

It is typically only practical to solve (1) with *direct methods*, in which the dynamic problem is

treated as a static problem, with the trajectories (x, u) replaced by the ordered set of variables $\mathcal{V} \coloneqq \{(x(t), u(t)), \forall t \in \widetilde{\mathcal{T}}\} \subset \mathbb{R}^{n_x} \times \mathbb{R}^{n_u}$, and (1b) and (1c) replaced by equivalent constraints on elements of $\mathcal{V}$. This translation is valid because the necessary conditions for optimality are equivalent for both problems, but the nature of $\mathcal{V}$ means that the static problem has an infinite number of variables. This issue is alleviated by instead discretising (x, u) over a *mesh* of $N + 1$ time nodes, $\mathcal{M} \coloneqq \{\tau_0, \tau_1, \ldots, \tau_N\} \subset \widetilde{\mathcal{T}}$, where $0 = \tau_0 < \tau_1 < \ldots < \tau_N = 1$. $\mathcal{M}$ divides $\widetilde{\mathcal{T}}$ into N intervals $\widetilde{\mathcal{T}}_k \coloneqq [\tau_k, \tau_{k+1}],\ k = 0, 1, \ldots, N-1$, and each interval $\widetilde{\mathcal{T}}_k$ has length $h_k \coloneqq \tau_{k+1} - \tau_k$.

The process of discretising (1) over a mesh is known as transcription, and it results in a finite-dimensional optimisation problem which can now be solved with efficient computational methods. The transcribed formulation is an approximation of the original trajectory optimisation problem, converging to the equivalent form as $h_k \to 0,\ \forall k$. Since numerical optimisation techniques become increasingly expensive as the number of decision variables grows, a compromise between computational speed and accuracy must always be considered when implementing a transcription routine.

2.2 Sequential Convex Programming

In many instances, the transcribed optimisation problem is non-convex. Such a problem is generally NP-hard [6], [7], and hence it may be unsuitable for embedded applications where running time must be limited. In contrast, many convex optimisation problems can be solved using algorithms with polynomial-time complexity [8]. This property is exploited in sequential convex programming (SCP), in which (1) is transformed into a convex problem via a series of approximations, and this problem is then discretised and solved with fast convex optimisation methods. The convex problem is solved multiple times in sequence, via an update rule that ensures that, with each new iteration, the solution to the approximate problem approaches that of the true problem.

For an optimal control problem to yield a convex optimisation problem, the cost must be convex, along with the functions which define inequality constraints. Additionally, the equality constraints must be affine. For simplicity, we henceforth assume that γ, ψ_E, ψ_I and the cost are all convex functions, and non-convexity is introduced in (1b) through the nonlinearity of f. This simplification enables generalisation to a wide range of SCP methods, though many methods do admit broader classes of problems, to which the work in this paper remains equally applicable. Nevertheless, the form assumed here is very typical in many optimal control applications. Furthermore, there exist techniques to relax non-convex constraints on u into a convex form, while retaining the same optimal solutions as the original problem [9], [10]. As an additional simplification, we assume that the dynamical system is time-invariant, and hence f is not a direct function of time.

Considering the assumptions on (1), a convex approximation of the problem is obtained by linearising f around a reference solution $(\hat{x}, \hat{u}, \hat{\theta})$ using a first-order Taylor approximation:

$$\begin{aligned} f\big(x(t), u(t), \theta\big) &\approx \hat{f}\big(x(t), u(t), \theta\big) \\ &\coloneqq f\big(\hat{x}(t), \hat{u}(t), \hat{\theta}\big) + \frac{\partial f}{\partial x}\big(\hat{x}(t), \hat{u}(t), \hat{\theta}\big)\big(x(t) - \hat{x}(t)\big) \\ &\quad + \frac{\partial f}{\partial u}\big(\hat{x}(t), \hat{u}(t), \hat{\theta}\big)\big(u(t) - \hat{u}(t)\big) + \frac{\partial f}{\partial \theta}\big(\hat{x}(t), \hat{u}(t), \hat{\theta}\big)\big(\theta(t) - \hat{\theta}(t)\big), \end{aligned} \tag{2}$$

where $\frac{\partial f}{\partial \cdot}$ denotes a Jacobian matrix of f. Then the nonlinear equality constraints (1b) can be replaced by the affine condition

$$\dot{x}(t) - \hat{f}\big(x(t), u(t), \theta\big) = 0, \quad \forall t \in \widetilde{\mathcal{T}}. \tag{3}$$

In its current form, the new convex optimal control problem may suffer from two phenomena known as *artificial unboundedness* and *artificial infeasibility*. The former is a consequence of the linearised system deviating too much from the reference trajectory. Not only does excessive deviation decrease the accuracy of the Taylor approximation, but it can also lead to the cost function becoming unbounded below. To avoid this problem, many SCP algorithms introduce a *trust region* to ensure that the trajectory remains sufficiently close to the reference. This additional condition can be enforced by adding trust region constraints to the problem, for example

$$\int_0^1 \|x(t) - \hat{x}(t)\|^2 \, dt \leq \Delta, \tag{4}$$

where $\|\cdot\|$ is the Euclidean norm and Δ is the radius of the trust region. An alternative approach is to augment the cost function with a weighted term which penalises the integral in (4).

Artificial infeasibility occurs when the convex problem is infeasible, even though the original problem is not. This can happen if there is no trajectory which can simultaneously satisfy the naturally convex constraints of (1) and the linearised constraints. Furthermore, the convex problem may be infeasible if the trust region is so small that it does not admit any feasible trajectories. This issue can be handled by introducing *virtual control* to ensure that the linearised dynamics always hold [3], [11]. For example, (3) can be amended to

$$\dot{x}(t) - \hat{f}_{aug}\big(x(t), u(t), \theta, \nu(t)\big) = 0, \;\; \forall t \in \widetilde{\mathcal{T}}, \tag{5}$$

where we define $\hat{f}_{aug}\big(x(t), u(t), \theta, \nu(t)\big) \coloneqq \hat{f}\big(x(t), u(t), \theta\big) + \nu(t)$, the linearised dynamics augmented with the virtual control trajectory $\nu : \mathbb{R} \to \mathbb{R}^{n_x}$. If (5) is enforced, the cost function must be augmented so that $\nu(t)$ is penalised, as a truly feasible solution cannot rely on any virtual control. An alternative remedy for artificial infeasibility is the direct penalisation of state constraint violations in the cost, without augmenting the linearised dynamics [4].

For the reasons described above, all SCP methods rely on an augmented convex trajectory optimisation problem, making use of features such as trust regions and penalty terms in the cost function. Additionally, all such algorithms solve a sequence of these problems, with the aim to converge to a solution to (1). This is often done by combining the augmented problem with a suitable update rule for each new iteration. For example, if the hard trust region constraint (4) is enforced, then Δ is iteratively updated to reflect how much we 'trust' the current approximation: if the algorithm detects that the approximation may remain valid even further from the current reference solution, then Δ may be increased, and in the converse case, the trust region shrinks.

Regardless of the nature of the convex problem and the update rules for algorithm parameters, SCP schemes always compute the linearisations around the solution from the previous iteration. For the first iteration, an initial solution guess is set as the reference triplet. A generic SCP procedure is now outlined in Algorithm 1, where $\mathbf{ACP}^i$ denotes the augmented convex problem at iteration i. The reference solution used in $\mathbf{ACP}^i$ is (x^i, u^i, θ^i), and any other iteration-specific features of the convex problem are described by a set of parameters $\mathcal{P}^i$, which is updated in each iteration according to the routine **UpdateSubproblem**. An example set may be $\mathcal{P}^i \coloneqq \{\Delta^i\}$, where Δ^i is the current trust region radius, with **UpdateSubproblem** representing the corresponding trust region update rule. Note that there do exist SCP algorithms for which $\mathcal{P}^i$ is always empty, in which case **UpdateSubproblem** is redundant. The algorithm terminates according to convergence and feasibility criteria, defined using a set of parameters $\mathcal{E}$. While Algorithm 1 cannot encapsulate all SCP variations, it is able to represent

Algorithm 1 Sequential Convex Programming

Input: Initial solution guess (x^0, u^0, θ^0)
Output: Solution $(x^{i+1}, u^{i+1}, \theta^{i+1})$ to **ACP**^{i} for some $i \in \mathbb{N}$
Data: Initial subproblem parameters $\mathcal{P}^0$, termination parameters $\mathcal{E}$, mesh $\mathcal{M}$

1: $i = 0$
2: **while** $(x^i)_{i\in\mathbb{N}}$ not converged **or** (x^i, u^i, θ^i) infeasible **do**
3: Solve **ACP**^{i} over $\mathcal{M}$ for $(x^{i+1}, u^{i+1}, \theta^{i+1})$
4: $\mathcal{P}^{i+1} =$ **UpdateSubproblem**$(x^{i+1}, u^{i+1}, \theta^{i+1}, x^i, u^i, \theta^i)$
5: $i \leftarrow i + 1$
6: **end while**

a very broad class of methods, and thus it is sufficient as a conceptual introduction. For the remainder of this paper, we assume the following property for Algorithm 1:

Assumption 1 *If* (1) *is feasible, then* $(x^i)_{i\in\mathbb{N}}$ *converges, and the limit is a solution for* (1).

For certain SCP methods, this assumption can be guaranteed by mathematical proof, subject to mild conditions on the original problem [4], [12]. However, various other realisations of SCP have been demonstrated to work well in practice, despite lacking rigorous convergence analysis [1]. Indeed, some of the most impressive practical implementations of SCP are yet to be supported by a strong theoretical foundation.

2.3 Transcribing the Convex Problem

Each **ACP**^{i} is a convex optimal control problem associated with a linear system, and hence it can be transcribed to a convex finite-dimensional optimisation problem, for which there exist many efficient solution algorithms implemented in so-called *solvers*. Therefore, to implement Algorithm 1 on a computer, a mesh $\mathcal{M}$ should be defined for this transcription. The control trajectory u can then be approximated over $\mathcal{M}$, such that it is defined continuously over $\widetilde{\mathcal{T}}$ but explicitly solved only at the $N+1$ mesh nodes. We denote u_k as the control vector at node τ_k, and $\tilde{u} : \mathbb{R} \to \mathbb{R}^{n_u}$ as the approximate control trajectory. There are various ways to construct $\tilde{u}$, though this is typically done by defining N continuous functions $v_k : \mathbb{R} \to \mathbb{R}^{n_u},\ k = 0, 1, \ldots, N-1$, to represent the control trajectory over each mesh interval in $\mathcal{M}$. These functions are often parameterised by the control vectors at the nearby mesh nodes, and so we say that $\tilde{u}$ is the *parameterised control trajectory*. A common parameterisation is *first-order hold (FOH)*, in which $\tilde{u}$ is piecewise affine:

$$\tilde{u}(t) = u_k + \frac{t - \tau_k}{h_k}(u_{k+1} - u_k), \quad \forall t \in \widetilde{\mathcal{T}}_k. \tag{6}$$

Having approximated the control trajectory with (6), an initial value problem (IVP) can then be posed, describing the state trajectory from τ_k to τ_{k+1} via the state transition matrix [13]. Numerically solving the IVP in all mesh intervals yields discrete dynamical constraints

$$x_{k+1} = \hat{f}^d(x_k, u_k, u_{k+1}, \theta), \quad k = 0, 1, \ldots, N-1, \tag{7}$$

where x_k is the transcribed state vector at τ_k, and $\hat{f}^d : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_\theta} \to \mathbb{R}^{n_x}$ is an affine function. To complete the transcription, the inequality constraints are enforced at the mesh nodes only, and the boundary conditions are naturally imposed at τ_0 and τ_N. The cost function is restated in discrete form, e.g., any integral is replaced by an approximate sum.

As well as FOH, there exist other transcription schemes based on direct approximation of u, and the work in this paper is equally applicable to such methods. However, FOH is considered a superior strategy in terms of both running time and accuracy [14], and hence we assume this scheme throughout the remainder of the paper. We conclude by mentioning an alternative discretisation approach: pseudospectral methods [15]. Although these can be combined with a mesh refinement strategy similar to the one described in Section 4, we do not attempt to generalise our work to this class of techniques. We note that pseudospectral methods can have advantages in terms of solution accuracy, but they generally yield a more complex optimisation problem, resulting in the solver taking longer to converge for each SCP iteration [14].

3 REPRESENTING THE CONTINUOUS-TIME SOLUTION

When the convex problem **ACP**^{i} is solved, the immediate result from the solver is a finite set of solution points $\{x_0^{i+1}, x_1^{i+1}, \dots, x_N^{i+1}, u_0^{i+1}, u_1^{i+1}, \dots, u_N^{i+1}, \theta^{i+1}\}$, where $x_k^{i+1} \in \mathbb{R}^{n_x}$ and $u_k^{i+1} \in \mathbb{R}^{n_u}$ are respectively the state and control vectors at τ_k, and $\theta^{i+1} \in \mathbb{R}^{n_\theta}$ is the associated parameter vector. By definition, the FOH discretisation scheme assumed a piecewise affine control trajectory, and so a continuous-time control solution $\tilde{u}^{i+1} : \mathbb{R} \to \mathbb{R}^{n_u}$ can be constructed by linearly interpolating between consecutive elements of the ordered set $u^{i+1} \coloneqq \{u_0^{i+1}, u_1^{i+1}, \dots, u_N^{i+1}\}$. A similar construction for the state solution $\tilde{x}^{i+1} : \mathbb{R} \to \mathbb{R}^{n_x}$ is not naturally available, as $x^{i+1} \coloneqq \{x_0^{i+1}, x_1^{i+1}, \dots, x_N^{i+1}\}$ is constrained according to (7), and the state trajectory x was never parameterised. The remainder of this section considers possible representations of the state solution $\tilde{x}^{i+1}$ recovered from **ACP**^{i}. To inform this discussion, we first introduce generic stopping criteria for SCP algorithms.

3.1 Stopping Criteria for SCP

Assumption 1 tells us that if we set a sufficiently strict convergence criterion for Algorithm 1 such that it terminates in iteration i^*, then the extracted solution $(\tilde{x}^{i^*+1}, \tilde{u}^{i^*+1}, \theta^{i^*+1})$ will be very close to a true solution for (1). A reasonable condition for convergence is

$$\int_0^1 \left\| \tilde{x}^{i+1}(t) - \tilde{x}^i(t) \right\|^2 dt \leq \epsilon_\rho, \tag{8}$$

where ϵ_ρ is some small positive scalar, and the integral can be computed with a numerical quadrature scheme. While a convergence criterion is set to ensure that the solution is 'sufficiently optimal', an exact solution can never be obtained. Therefore, an additional stopping criterion is introduced to ensure that the represented solution at least satisfies the constraints (1b)-(1e) to some tolerance level. The most critical obstacle to achieving this stems from the linearisation of the dynamics: if f is badly approximated with (2), then the solution to the convex problem will be inconsistent with (1). Therefore, a common stopping criterion is an upper bound on the deviation of the propagated state trajectory from the discrete state solution x^{i+1}. This is commonly expressed in terms of the local error over each interval $\widetilde{\mathcal{T}}_k,\ k = 0, 1, \dots, N-1$. For any $\widetilde{\mathcal{T}}_k$, an IVP of the following form is constructed using information from **ACP**^{i}:

$$y' = f\big(y(t), \tilde{u}^{i+1}(t), \theta^{i+1}\big), \quad t \in \widetilde{\mathcal{T}}_k, \quad y(\tau_k) = x_k^{i+1}, \tag{9}$$

where $y(t) \in \mathbb{R}^{n_x}$ and the solution $y^* : \mathbb{R} \to \mathbb{R}^{n_x}$ describes the evolution of the states over the interval. The deviation associated with $\widetilde{\mathcal{T}}_k$ is then computed by comparing x_{k+1}^{i+1} to $y^*(\tau_{k+1})$, and this is then checked against some defined tolerance level. For example, by setting some positive scalar ϵ_δ, the feasibility criterion may take the form

$$\max_k \left\| x_{k+1}^{i+1} - y^*(\tau_{k+1}) \right\| \leq \epsilon_\delta. \tag{10}$$

We now have two types of stopping criterion that should simultaneously be enforced: one indicating sufficient convergence to an optimal solution, and the other ensuring that the solution is sufficiently feasible. If (8) and (10) are to be used in Algorithm 1, then the set of termination parameters is defined as $\mathcal{E} \coloneqq \{\epsilon_\rho, \epsilon_\delta\}$. Considering these features of the general SCP implementation, we now proceed to discuss two different approaches for representing the state solution $\tilde{x}^{i+1}$.

3.2 Solving an IVP for the State Trajectory

To extract $\tilde{x}^{i+1}$, a possible strategy is to propagate the original nonlinear dynamics over $\widetilde{\mathcal{T}}$ (or even over the true time interval $\mathcal{T}^{i+1}$, obtained by scaling the problem using information from θ^{i+1}). This propagation is performed by solving an IVP of the form

$$z' = f\big(z(t), \tilde{u}^{i+1}(t), \theta^{i+1}\big), \;\; t \in \widetilde{\mathcal{T}}, \;\; z(0) = x_0, \tag{11}$$

where $z(t) \in \mathbb{R}^{n_x}$ and the solution $z^* : \mathbb{R} \to \mathbb{R}^{n_x}$ forms $\tilde{x}^{i+1}$. It is typically not possible to solve (11) analytically, and so a closed-form expression for $\tilde{x}^{i+1}$ cannot be obtained. However, accurate numerical methods can be employed, such as the fourth-order Runge-Kutta scheme. This can be implemented over any user-defined mesh of sufficient resolution.

There is a potential pitfall in recovering $\tilde{x}^{i+1}$ by simulating the nonlinear dynamical evolution according to $\tilde{u}^{i+1}$. In applications which demand very low computation time, such as in certain embedded systems, it may not be feasible for an SCP algorithm to run until very strict stopping criteria are satisfied. This motivates setting more relaxed termination conditions which can be met after only a few iterations. However, if ϵ_ρ and ϵ_δ are increased, then the state trajectory derived from (11) will deviate from the discrete solution points encapsulated in x^{i+1}. Because the constraints (1c) were only enforced explicitly at the mesh nodes in $\mathbf{ACP}^i$, deviation from these $N+1$ solution points may lead to violation of state inequality constraints along the trajectory. Additionally, the solution $\tilde{x}^{i+1}$ will no longer satisfy the terminal conditions described in (1d) and (1e). While (1c) may be formulated such that the system can afford to violate the nominal constraints by some small margin, it may not be acceptable to deviate from the terminal conditions. For example, this may be the case when generating a trajectory to a precise target, such as for planetary landing with pinpoint accuracy.

This disadvantage motivates the adoption of a scheme that can construct $\tilde{x}^{i+1}$ such that it exactly satisfies the boundary conditions of (1). We now describe a natural approach that can achieve this, subject to the condition that there are no equality constraints on the terminal control input, only on the terminal state vector. This assumption holds true for many optimal control problems in practice.

3.3 Hermite Interpolation of the Discrete State Solution

To construct a state trajectory that exactly satisfies the terminal conditions, we simply ensure that $\tilde{x}^{i+1}$ interpolates the points in x^{i+1}. This also guarantees that any path inequality constraints on the states will be satisfied at the nodes of $\mathcal{M}$. As a sensible strategy, we adopt Hermite interpolation to define the following approximation:

$$\tilde{x}^{i+1} \coloneqq q_k(t; x_k^{i+1}, x_{k+1}^{i+1}, f_k^{i+1}, f_{k+1}^{i+1}), \;\; \forall t \in \widetilde{\mathcal{T}}_k, \;\; k = 0, 1, \ldots, N-1, \tag{12}$$

where $f_k^{i+1} \coloneqq f(x_k^{i+1}, u_k^{i+1}, \theta^{i+1})$ and q_k is a cubic polynomial. For clarity, the parameters are henceforth ignored in the notation, and each polynomial is written as $q_k(t)$. We define for some $\eta \in \mathbb{R}$ the following four Hermite polynomials, which are the basis functions used to construct q_k:

$$\begin{aligned} H_{00}(\eta) &\coloneqq 2\eta^3 - 3\eta^2 + 1, & H_{01}(\eta) &\coloneqq -2\eta^3 + 3\eta^2, \\ H_{10}(\eta) &\coloneqq \eta^3 - 2\eta^2 + \eta, & H_{11}(\eta) &\coloneqq \eta^3 - \eta^2. \end{aligned} \tag{13}$$

We now define $q_k(t), \forall t \in \widetilde{\mathcal{T}}_k$, for $k = 0, 1, \ldots, N-1$:

$$q_k(t) \coloneqq H_{00}\big(s(t)\big)x_k^{i+1} + H_{10}\big(s(t)\big)h_k f_k^{i+1} + H_{01}\big(s(t)\big)x_{k+1}^{i+1} + H_{11}\big(s(t)\big)h_k f_{k+1}^{i+1}, \tag{14}$$

where we use $s(t) \coloneqq \frac{t-\tau_k}{h_k} \in [0,1]$. The function q_k satisfies the following four conditions on the boundaries of $\widetilde{\mathcal{T}}_k$:

$$\begin{aligned} q_k(\tau_k) &= x_k^{i+1}, & q_k(\tau_{k+1}) &= x_{k+1}^{i+1}, \\ q_k'(\tau_k) &= f_k^{i+1}, & q_k'(\tau_{k+1}) &= f_{k+1}^{i+1}. \end{aligned} \tag{15}$$

In summary, applying Hermite interpolation over all N mesh intervals results in a continuously differentiable trajectory $\tilde{x}^{i+1}$ which passes through all points in the set x^{i+1}. Furthermore, the time-derivative of $\tilde{x}^{i+1}$ is exactly consistent with the dynamical equations of the system at the mesh nodes. This scheme has the advantage of yielding a state trajectory which exactly satisfies the terminal conditions in (1). However, this comes at a price. Since $\tilde{x}^{i+1}$ has been constructed independently of $\tilde{u}^{i+1}$ and a corresponding nonlinear simulation procedure, it will generally not be possible for the system to follow $\tilde{x}^{i+1}$ by implementing $\tilde{u}^{i+1}$. While consistency between the two trajectories can be approached as the SCP algorithm converges to an exact solution, we have proposed Hermite interpolation for the case when the algorithm may need to use relatively relaxed stopping criteria.

In order to permit early termination of Algorithm 1 while still enforcing exact terminal constraints on the state trajectory, we must ensure that the extracted solution $\tilde{x}^{i+1}$ can be followed by the system. The relation between $\tilde{x}^{i+1}$ and $\tilde{u}^{i+1}$ can be characterised by the perturbed dynamics

$$\dot{\tilde{x}}^{i+1}(t) = \phi\big(\tilde{x}^{i+1}(t), \tilde{u}^{i+1}(t), \theta^{i+1}, w(t)\big) \coloneqq f\big(\tilde{x}^{i+1}(t), \tilde{u}^{i+1}(t), \theta^{i+1}\big) + w(t), \tag{16}$$

where $w : \mathbb{R} \to \mathbb{R}^{n_x}$ describes the residual of (1b) for the solution $(\tilde{x}^{i+1}, \tilde{u}^{i+1}, \theta^{i+1})$, and $\phi : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_\theta} \times \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$. Therefore, $\tilde{x}^{i+1}$ will instead require the control trajectory $u_s = \tilde{u}^{i+1} + \upsilon$, for some $\upsilon : \mathbb{R} \to \mathbb{R}^{n_u}$. Depending on the nature of the system, u_s can either be determined entirely with a feedback control policy, or it can be formed by combining the nominal control solution with a feedback controller, as in tube-based model predictive control [16]. To guarantee the existence of a feasible u_s, we must tighten the constraints on u in (1). That is, if $\mathcal{U}$ defines the set of feasible control trajectories, then the posed problem (1) should enforce $u \in \mathcal{U}_R \subset \mathcal{U}$, where the contracted set $\mathcal{U}_R$ is chosen such that $u_s \in \mathcal{U}$. Equivalently, $\mathcal{U}_R$ is associated with a suitable robust control invariant set for the uncertain system $\dot{x}(t) = \phi\big(x(t), u(t), \theta, w(t)\big)$ [17], [18]. A generic theoretical framework for determining $\mathcal{U}_R$ is beyond the scope of this paper, though this set may often be approximated using Monte Carlo methods.

3.4 Measuring Error in the Hermite Interpolation Scheme

A convenient consequence of defining $\tilde{x}^{i+1}$ with a piecewise polynomial is that the time-derivative of the constructed solution can be obtained analytically. This allows exact computation of the dynamical evolution of the represented state trajectory, which we denote by the function $\tilde{f}(t)$:

$$\tilde{f}(t) \coloneqq q_k'(t), \quad \forall t \in \widetilde{\mathcal{T}}_k, \quad k = 0, 1, \ldots, N-1, \tag{17}$$

where each q_k' is a quadratic function in t, parameterised by $(x_k^{i+1}, x_{k+1}^{i+1}, f_k^{i+1}, f_{k+1}^{i+1})$. For an exactly feasible solution, the dynamics of $\tilde{x}^{i+1}$ will be equivalent to the nonlinear system dynamics evaluated over the continuous-time solution $(\tilde{x}^{i+1}, \tilde{u}^{i+1}, \theta^{i+1})$. However, in the realistic case where $\tilde{x}^{i+1}$ and $\tilde{u}^{i+1}$ cannot both satisfy (1b) simultaneously, there will be some non-zero error

$$\lambda(t) = \tilde{f}(t) - f\big(\tilde{x}^{i+1}(t), \tilde{u}^{i+1}(t), \theta^{i+1}(t)\big), \quad \forall t \in \widetilde{\mathcal{T}}. \tag{18}$$

Since $\tilde{x}^{i+1}$ is the assumed state trajectory solution for (1), the residual λ directly assesses the accuracy to which the differential equations in (1b) are satisfied for the triplet $(\tilde{x}^{i+1}, \tilde{u}^{i+1}, \theta^{i+1})$. For each state, enumerated with index $j = 1, \ldots, n_x$, we can subsequently assess the *absolute local error* $\mu_{j,k}$ over a particular interval $\widetilde{\mathcal{T}}_k$:

$$\mu_{j,k} \coloneqq \int_{\tau_k}^{\tau_{k+1}} |\lambda_j(t)| dt, \tag{19}$$

where $\lambda_j(t)$ is the corresponding component of the vector $\lambda(t) \in \mathbb{R}^{n_x}$. As a more useful measure, we define the *relative local error* ξ_k of the solution over $\widetilde{\mathcal{T}}_k$ [19]:

$$\xi_k = \max_j \frac{\mu_{j,k}}{w_j + 1}, \tag{20}$$

where w_j is a weighting to account for the relative magnitude of each state.

4 VARIABLE-RESOLUTION SEQUENTIAL CONVEX PROGRAMMING

Thus far, we have paid little attention to the choice of $\mathcal{M}$. In fact, the mesh itself can significantly influence the algorithm performance. A coarse mesh will result in a relatively small number of optimisation variables in $\mathbf{ACP}^i$, and so very little time is spent in the solver. Since the solver step of Algorithm 1 dominates the running time in each iteration, it may be very desirable to accelerate this phase. However, excessively large steps h_k will result in lower solution accuracy.

Ideally, we use the smallest possible mesh that can achieve a desired level of accuracy. The optimal mesh in this sense is not known a priori, and it may even vary between different realisations of (1) for the same system. This motivates the implementation of an *adaptive mesh refinement (AMR)* scheme. AMR is a popular class of heuristic techniques for efficient and accurate numerical trajectory optimisation [19], [20]. The idea is very simple: solve a transcribed problem on a given mesh, then assess the accuracy of the solution. In regions with excessive local error, refine the mesh by adding more nodes and then transcribe the problem over the new mesh. This procedure is repeated until some error tolerance is met. For each new iteration, the problem can be warm-started using the previous solution, thus reducing computation time.

AMR fits very naturally into the SCP framework, as Algorithm 1 is already designed to repeatedly solve a finite-dimensional optimisation problem, and the solution obtained in iteration i is already stored for use in iteration $i + 1$. Algorithm 2 presents a general procedure for SCP combined with AMR, using the **UpdateMesh** routine to implement mesh refinement. It is natural to combine the Hermite representation described in Section 3 with the relative local error as defined in (20). Then **UpdateMesh** adds n_s points in any interval for which $\xi_k > \epsilon_\lambda$, where ϵ_λ is a small positive scalar and $n_s \in \mathbb{N}$ is set by the designer. Note that this routine will not change the mesh if the measured error is satisfactory over every interval. To guarantee an upper bound on the relative local error in the solution, ϵ_λ is added to the set $\mathcal{E}$.

The set of mesh refinement parameters $\mathcal{R}$ contains not only n_s, but also the positive scale factor σ, which is used to define a relaxed convergence tolerance $\bar{\epsilon}_\rho \coloneqq \sigma\epsilon_\rho$. The purpose of $\bar{\epsilon}_\rho$ is to detect if the solution is sufficiently close to convergence that the mesh refinement routine can be triggered. This condition is recommended, as the first few SCP iterations can yield very bad solutions, and so they should not be used to assess which regions of $\widetilde{\mathcal{T}}$ require higher resolution. Indeed, if mesh refinement is implemented from the first iteration, then the mesh may quickly become excessively fine, thus slowing the iterations significantly. Algorithm 2 represents a generic framework which can be

Algorithm 2 Variable-Resolution Sequential Convex Programming

Input: Initial solution guess (x^0, u^0, θ^0)
Output: Solution $(x^{i+1}, u^{i+1}, \theta^{i+1})$ to **ACP**^{i} for some $i \in \mathbb{N}$
Data: $\mathcal{P}^0$, $\mathcal{E}$, initial mesh $\mathcal{M}^0$, mesh refinement parameters $\mathcal{R}$

$i = 0$
while $(x^i)_{i\in\mathbb{N}}$ not converged **or** (x^i, u^i, θ^i) infeasible **do**
  Solve **ACP**^{i} over $\mathcal{M}^i$ for $(x^{i+1}, u^{i+1}, \theta^{i+1})$
  if $(x^i)_{i\in\mathbb{N}}$ converged with relaxed tolerance **then**
    $\mathcal{M}^{i+1} =$ **UpdateMesh**$(x^{i+1}, u^{i+1}, \theta^{i+1})$
  else
    $\mathcal{M}^{i+1} = \mathcal{M}^i$
  end if
  $\mathcal{P}^{i+1} =$ **UpdateSubproblem**$(x^{i+1}, u^{i+1}, \theta^{i+1}, x^i, u^i, \theta^i)$
  $i \leftarrow i + 1$
end while

customised to the needs of the particular application. In our numerical experiments, we found that it is often sufficient to set $n_s = 1$, and using a larger value can rapidly increase the size of **ACP**^{i}. Additionally, we recommend setting a very coarse initial mesh $\mathcal{M}^0$. Combined with the parameter σ, this results in a small number of very fast iterations at the start of the algorithm, such that the convex formulation can already represent (1) reasonably well after very little computation time.

5 EXAMPLE: 6-DOF LUNAR LANDER

To demonstrate the framework of variable-resolution SCP with Hermite interpolation of the discrete state solution, we now present the case study of optimal descent guidance for a lunar lander.

5.1 Description of the Dynamical System

We consider a rigid lander with a main vertical thruster and sixteen small reaction control thrusters, such that the overall propulsion system can control the vehicle in six degrees of freedom (6-DoF). The control thrusters are distributed in sets of four ('quads') around the outside of the lander, such that each quad consists of two opposing vertical thrusters and two orthogonal lateral thrusters. We define the state vector $x(t) \coloneqq \begin{bmatrix} m & \iota_B^T & r_R^T & v_R^T & p^T & q_s & \omega_B^T \end{bmatrix}^T \in \mathbb{R}^{17}$, where m is the spacecraft mass and $\iota_B \coloneqq [I_{xx} \quad I_{yy} \quad I_{zz}]^T$ defines the non-zero components of the inertia matrix. The vectors r_R and v_R respectively represent the spacecraft position and velocity in an inertial frame Ψ_R centred at the landing site, such that z_R points upwards in the local vertical direction. ω_B denotes the angular velocity of the vehicle in the body-fixed frame Ψ_B, and $p \in \mathbb{R}^3$ and $q_s \in \mathbb{R}$ collectively form the unit quaternion q_R describing the attitude of Ψ_B relative to Ψ_R. We also define the control vector $u(t) \coloneqq \begin{bmatrix} F_T & \delta_1 & \delta_2 & \cdots & \delta_{16} \end{bmatrix}^T \in \mathbb{R}^{17}$, where F_T is the thrust level of the main thruster, and each δ_i defines the thrust from a given reaction control thruster. We hence define the system dynamics

$$\dot{x}(t) = g\big(x(t), u(t)\big), \tag{21}$$

where the function g is constructed according to classical laws of dynamics and kinematics, combined with models for the variation of the mass properties, which we define now. The mass $m(t)$ of the vehicle decreases as propellant is used, and this is described by

$$\dot{m}(t) = -\alpha\Big(F_T(t) + \sum_i \delta_i(t)\Big), \tag{22}$$

where α is a scalar constant characterising the efficiency of the propulsion system. Similarly, the evolution of the inertia components follows

$$i_B(t) = -\Big(F_T(t) + \sum_i \delta_i(t)\Big)\beta, \tag{23}$$

where β is a constant vector, related to α and the position of the propellant tank in Ψ_B.

5.2 Problem Formulation

For the system described above, we now formulate a free-final-time trajectory optimisation problem for pinpoint lunar landing. The system dynamics are enforced over the fixed interval $\widetilde{\mathcal{T}}$ by adapting (21) into the form of (1b), where θ represents the final time t_f. We now describe the remainder of the problem over the true time interval $\mathcal{T}$, which is parameterised by θ. For all $t \in \mathcal{T}$, we set the control bounds $F_T(t) \in [F_{T,min}, F_{T,max}]$ and $\delta_i(t) \in [\delta_{min}, \delta_{max}],\ i = 1, 2, \ldots, 16$. The mass of the spacecraft can never be lower than the dry mass (the mass of the vehicle without any propellant) and so we enforce $m(t) \geq m_{dry},\ \forall t$. Since the altitude of the lander can never be negative, we also set $r_R^T(t)e_3 \geq 0,\ \forall t$, where $e_3 \coloneqq [0 \ \ 0 \ \ 1]^T$. Furthermore, we constrain the angular rate of the body according to $\|\omega_B(t)\| \leq \omega_{B,max}$, and we set ϕ_{max} to denote the maximum angle that the spacecraft may tilt off vertical. We introduce boundary equality constraints $x(0) = x_0$ and $Ex(t_f) = x_f$, where $x_f \in \mathbb{R}^{13}$ defines the stationary, upright, non-rotating spacecraft at the target point (i.e., the origin in Ψ_R), and $E \coloneqq [0_{13\times 4} \ \ I_{13}]$ ensures that the terminal mass properties remain free. In this problem, we choose to minimise propellant usage, which is equivalent to maximising the spacecraft mass at the end of the trajectory. Hence we define the cost function as

$$J\big(x(t_f)\big) \coloneqq -[1 \quad 0_{1\times 16}]\, x(t_f). \tag{24}$$

The trajectory optimisation problem can now be stated as follows: minimise (24) subject to the system dynamics, the inequality constraints and the boundary conditions. We note that the inequality constraints, the boundary conditions and the cost function can all be expressed as convex mappings, and so the problem is only non-convex because of the nonlinear dynamics.

5.3 Numerical Results

To test our algorithmic framework, we have implemented Algorithm 2 in Julia with the conic solver ECOS [21]. We use the penalty trust region method (PTR), a recent SCP algorithm that has demonstrated very good practical performance, despite so far lacking a rigorous convergence analysis [11]. PTR combines virtual control with soft trust region constraints, in which the trust region radius itself is a variable in $\mathbf{ACP}^i$. To drive the algorithm towards a solution, this variable is penalised in the augmented cost function. For each extracted solution $(\tilde{x}^{i+1}, \tilde{u}^{i+1}, \theta^{i+1})$, three numbers are computed to check the stopping criteria: a convergence measure ζ_ρ, a local propagation error measure ζ_δ, and a measure relating to the integrated residual of the dynamics ζ_λ. The algorithm terminates when $\zeta_\rho \leq \epsilon_\rho$, $\zeta_\delta \leq \epsilon_\delta$ and $\zeta_\lambda \leq \epsilon_\lambda$.

Properties of the sample spacecraft are listed in Table 1, and the parameters of the implemented algorithm are shown in Table 2. Using these parameters, the algorithm was tested on various versions of the problem, where each version is distinguished by the initial position and attitude states $r_{R,0}$, $v_{R,0}$, $q_{R,0}$ and $\omega_{B,0}$ (the initial mass properties are always given by the wet values in Table 1). An encouraging observation was that the algorithm performed similarly for a wide range of test problems. To illustrate the behaviour of the framework, we present some results for a single problem, noting that the performance is representative of that observed in many other realisations of the

case study. Specifically, we use the following initial conditions, stated in SI units where applicable: $r_{R,0} = [300 \;\; 200 \;\; 500]^T$, $v_{R,0} = [-1 \;\; 1 \;\; 1]^T$, $q_{R,0} = (0, 0, 0, 1)$, based on scalar-last convention, and $\omega_{B,0} = [0.2 \;\; 0.2 \;\; 0.1]^T$ (in rad/s). These conditions may be practically unrealistic, but they result in a more interesting problem on which to demonstrate the algorithm.

Table 1: Spacecraft parameters

Parameter	Unit	Value
m_{wet}	kg	5000
m_{dry}	kg	3000
$\iota_{B,wet}$	kgm^2	$5000 \cdot [1 \;\; 1 \;\; 1]^T$
$\iota_{B,dry}$	kgm^2	$4000 \cdot [1 \;\; 1 \;\; 1]^T$
α	s/m	3e-4
β	sm	1e-4 $\cdot [1 \;\; 1 \;\; 1]^T$
$F_{T,min}$	kN	2
$F_{T,max}$	kN	10
δ_{min}	kN	0
δ_{max}	kN	0.2
$\omega_{B,max}$	rad/s	1
ϕ_{max}	$^\circ$	45

Table 2: Algorithm parameters

Parameter	Value
ϵ_ρ	1e-3
ϵ_δ	1e-2
ϵ_λ	1e-3
σ	1e2
n_s	1
$\mathcal{M}^0$	$\{0, 0.2, 0.4, 0.6, 0.8, 1\}$

Figure 1 plots the interpolated state trajectory solution, with a simplified lander shown at certain points along the path to illustrate the attitude. The progress of the algorithm is presented in Table 3, where m_p is the mass (in kilograms) of propellant used in the latest solution, t_{CPU} denotes the computation time (in seconds) to solve **ACP**^{i}, and N represents the number of intervals in the latest mesh. Tests were performed on an HP EliteBook with Intel Core i5 2.6 GHz CPU.

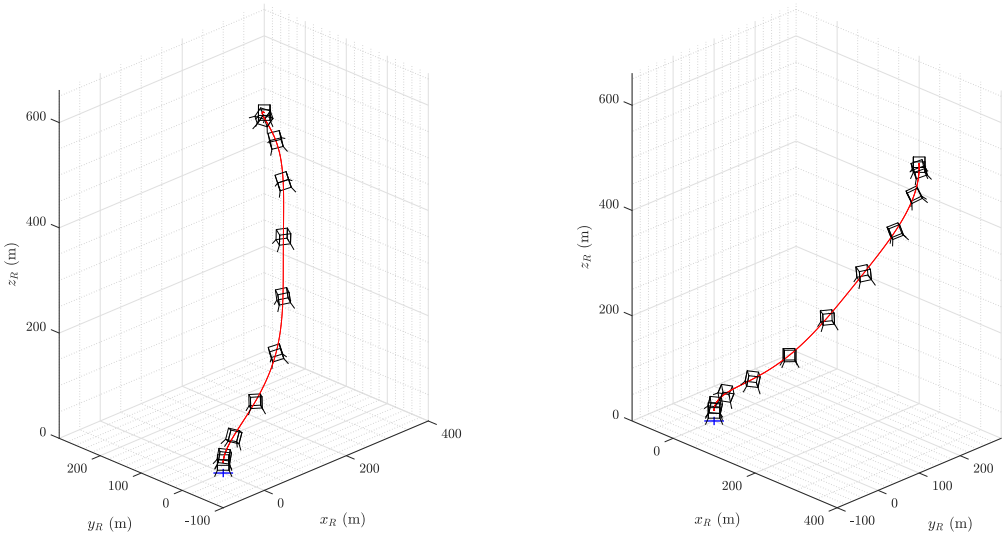

Figure 1: The interpolated state trajectory solution, as seen from two different angles.

We note that, despite high error ζ_λ in the first few iterations, no points are added to the mesh, as $\zeta_\rho > \sigma\epsilon_\rho$. Once **ACP**^{5} is solved, this convergence criterion is met, and so the mesh is refined in all intervals. After two subsequent updates to the mesh, the resolution is now sufficiently high to render ζ_λ acceptable, indicating that the polynomial interpolation between mesh nodes generates a sufficiently feasible continuous-time trajectory. In the remaining six iterations, ζ_ρ and ζ_δ continue

Table 3: Progression of the algorithm

i	m_p	ζ_ρ	ζ_δ	ζ_λ	N	t_{CPU}
0	0.8355	9.202	154.7	1.000	5	0.0179
1	295.4	0.5900	345.1	2.930	5	0.0182
2	191.6	1.447	759.2	3.548	5	0.0237
3	221.1	0.8105	408.4	0.5460	5	0.0246
4	178.4	0.1201	137.5	0.2900	5	0.0158
5	183.4	0.05591	30.66	0.1623	5	0.0138
6	177.5	6.425e-3	3.785	0.01766	10	0.0591
7	174.9	2.475e-3	0.4126	1.883e-3	20	0.1084
8	172.8	1.712e-3	0.02919	8.001e-4	21	0.0979
9	171.0	1.506e-3	0.01422	7.646e-4	21	0.0991
10	169.4	1.356e-3	0.01148	7.319e-4	21	0.0931
11	167.9	1.230e-3	9.785e-3	7.147e-4	21	0.1264
12	166.6	1.090e-3	8.452e-3	7.643e-4	21	0.1526
13	165.5	9.152e-4	7.174e-3	8.074e-4	21	0.1216

to decrease, as the linearised dynamics are still not representing the true system well enough. As the algorithm progresses, we can be increasingly confident that a feasible solution can be recovered (particularly if we formulate the original problem with tightened control constraints), and we do not attempt to keep iterating until an optimal solution is obtained. Indeed, observing the changing m_p in Table 3, its value is still decreasing substantially by the time the algorithm is terminated at $i = 13$. It is reasonable to assume that we can then extract a 'near-optimal' solution, and we note that exact optimality is generally not required in real applications. A final observation from Table 3: for a given mesh size N, the computation time t_{CPU} is relatively consistent. This tends to be a great benefit of employing a convex solver, and it is particularly useful if this algorithm needs to be customised and tuned for an embedded application with constraints on computation time.

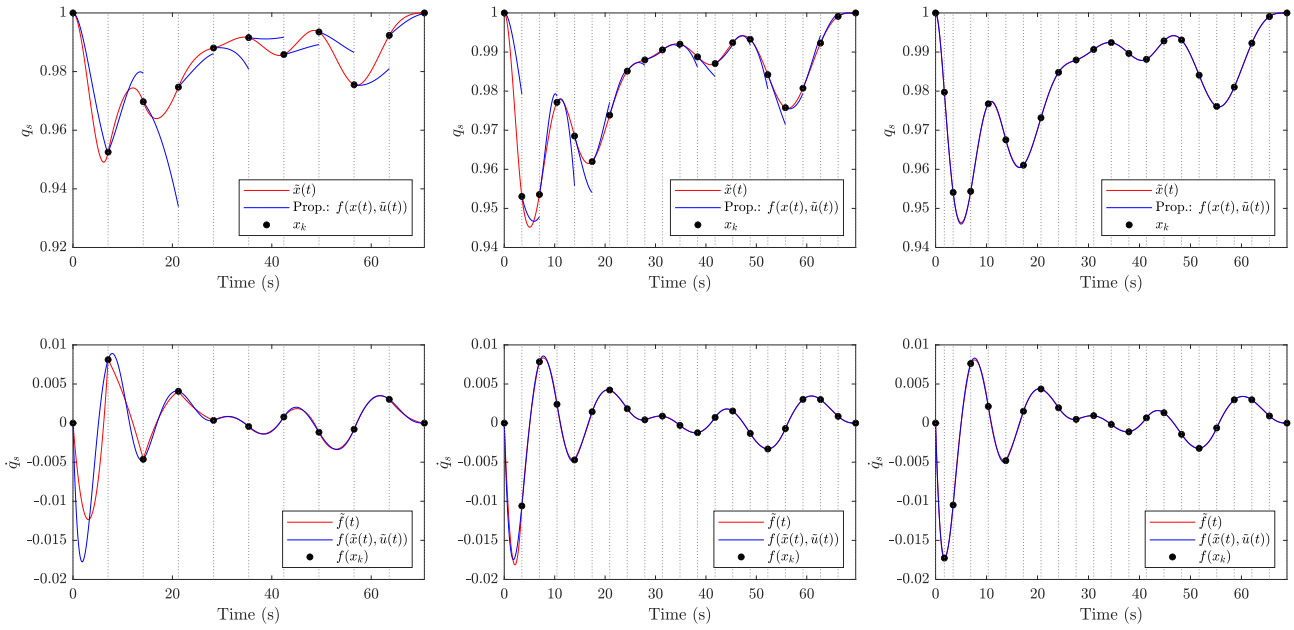

Figure 2: Feasibility of the extracted trajectory of q_s for (left to right) $i = 6, 7, 8$.

Figure 2 illustrates three intermediate solutions for q_s, the scalar component of the quaternion q_R. The top row compares the Hermite state solution with local propagation of the state trajectory using the nonlinear dynamics, as shown in (10), and the bottom row compares the gradient of the Hermite state solution with the evaluation of the nonlinear dynamics along the solution. This is equivalent to the comparison presented in (18). The figure shows this data for $i = 6, 7, 8$, thus providing an effective illustration of the variable-resolution framework.

To observe the effects of mesh refinement, we can consider just the bottom row of plots: the inaccuracy evident in the leftmost figure justifies adding points over the entire mesh, and the middle plot exhibits the resulting improvement in the integrated error measure. However, there remains a visible mismatch between the two curves, particularly within the first mesh interval. Hence only this interval is refined now, resulting in a more accurate non-uniform mesh at the next iteration. We now complement these remarks by considering the top row of plots, illustrating the error derived from solving the local IVP (9) in each mesh interval. The clear improvement over the three plots is a result of the Jacobian linearisation (2) yielding a more representative convex approximation of (1). This follows the gradual progression of the SCP procedure, since the assumed reference solution continues to converge towards some unique pair of state and control trajectories. With each new iteration, this converging behaviour shrinks the region over which the Taylor approximation must extend, hence reducing the deviation from the nonlinear dynamics of the system. It should be emphasised that the simultaneous refining of the mesh does also promote faster convergence, as is evident from the substantial decrease in ζ_ρ and ζ_δ for the iterations shown in the figure. However, the two errors ζ_δ and ζ_λ remain subtly different metrics: ζ_δ measures the accuracy of the discrete-time linearised dynamics (7) as an approximate model of the continuous-time system, whereas ζ_λ measures how well the polynomial interpolant (14) can represent the nonlinear dynamics between mesh nodes.

6 CONCLUSION

We have summarised some generic concepts of sequential convex programming, based on which we have motivated the use of Hermite polynomial basis functions to directly interpolate the finite set of state solution points found by the convex solver. In particular, this has been justified for applications where exact satisfaction of the terminal state conditions is imperative. By employing two different error measures to assess feasibility of the solution, it may be possible to terminate the SCP algorithm with a weaker condition on the convergence of the sequence of solutions. This strategy can be particularly effective if the original control constraints can be tightened such that the interpolated state trajectory can be followed with a perturbed control trajectory. We have subsequently described a generic implementation of mesh refinement with SCP, enabling a guaranteed level of accuracy in the solution while avoiding introducing an excessive number of optimisation variables into the transcribed convex problem. Finally, we have demonstrated some attractive features of this framework using the case study of optimal descent guidance for a lunar lander.

REFERENCES

[1] D. Malyuta, T. P. Reynolds, M. Szmuk, *et al.*, "Convex optimization for trajectory generation: A tutorial on generating dynamically feasible trajectories reliably and efficiently," *IEEE Control Systems Magazine*, vol. 42, no. 5, pp. 40–113, 2022.

[2] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[3] Y. Mao, M. Szmuk, and B. Açıkmeşe, "Successive convexification of non-convex optimal control problems and its convergence properties," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, 2016, pp. 3636–3641.

[4] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "GuSTO: Guaranteed sequential trajectory optimization via sequential convex programming," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6741–6747.

[5] D. Malyuta, Y. Yu, P. Elango, and B. Açıkmeşe, "Advances in trajectory optimization for space vehicle control," *Annual Reviews in Control*, vol. 52, pp. 282–315, 2021.

[6] K. G. Murty and S. N. Kabadi, "Some NP-complete problems in quadratic and nonlinear programming," *Mathematical Programming*, vol. 39, pp. 117–129, 1987.

[7] S. A. Vavasis, "Quadratic programming is NP," *Information Processing Letters*, vol. 36, pp. 73–77, 1990.

[8] Y. Nesterov and A. Nemirovskii, *Interior-Point Polynomial Methods in Convex Programming*. Society for Industrial and Applied Mathematics, 1994.

[9] B. Açikmeşe and L. Blackmore, "Lossless convexification of a class of optimal control problems with non-convex control constraints," *Automatica*, vol. 47, pp. 341–347, 2011.

[10] B. Açikmeşe, J. M. Carson, and L. Blackmore, "Lossless convexification of non-convex control bound and pointing constraints of the soft landing optimal control problem," *IEEE Transactions on Control Systems Technology*, vol. 21, pp. 2104–2113, 2013.

[11] M. Szmuk and B. Açikmeşe, "Successive convexification for 6-DoF Mars rocket powered landing with free-final-time," in *2018 AIAA Guidance, Navigation, and Control Conference*, American Institute of Aeronautics and Astronautics, 2018.

[12] R. Bonalli, T. Lew, and M. Pavone, "Analysis of theoretical and numerical properties of sequential convex programming for continuous-time optimal control," *IEEE Transactions on Automatic Control*, pp. 1–16, 2022.

[13] J. Hespanha, *Linear Systems Theory*. Princeton Press, 2018.

[14] D. Malyuta, T. P. Reynolds, M. Szmuk, M. Mesbahi, B. Açikmeşe, and J. M. Carson, "Discretization performance and accuracy analysis for the powered descent guidance problem," in *AIAA SciTech Forum*, 2019.

[15] D. Garg, M. Patterson, W. W. Hager, A. V. Rao, D. A. Benson, and G. T. Huntington, "A unified framework for the numerical solution of optimal control problems using pseudospectral methods," *Automatica*, vol. 46, no. 11, pp. 1843–1851, 2010.

[16] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, 2nd ed. Nob Hill Publishing, 2019.

[17] F. Blanchini, "Set invariance in control," *Automatica*, vol. 35, no. 11, pp. 1747–1767, 1999.

[18] E. Kerrigan, "Robust constraint satisfaction: Invariant sets and predictive control," Ph.D. dissertation, University of Cambridge, 2000.

[19] J. T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Society for Industrial and Applied Mathematics, 2010.

[20] S. Jain and P. Tsiotras, "Trajectory optimization using multiresolution techniques," *Journal of Guidance, Control, and Dynamics*, vol. 31, no. 5, pp. 1424–1436, 2008.

[21] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *2013 European Control Conference (ECC)*, 2013, pp. 3071–3076.