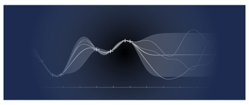


Master thesis

Development of a Multi-phase Optimal Control Software for Aerospace applications (MPOPT)

Faculté Sciences et Techniques de l'Ingénieur, Génie Mécanique,
Laboratoire d'automatique
École Polytechnique Fédérale de Lausanne

Thése de Master de
Devakumar Thammisetty

Supervisée par:

Petr Listov, Doctorial Assistant, Laboratoire d'automatique, EPFL

Prof. Jones Colin Neil, Laboratoire d'automatique, EPFL

Lausanne, EPFL, 2020

Abstract

An open source, extensible pseudo-spectral collocation based multi-phase nonlinear optimal control problem (OCP) solver package named MPOPT is developed as part of the thesis. While a number of OCP solvers already exist, the present package takes advantage of the latest developments in Nonlinear programming problem (NLP) formulation methods. The package written in the Python programming language, transcribes a given OCP into an NLP that is compatible with another open source tool (CasADi) and solves the NLP using various available solver plugins. CasADi is a non-linear optimization tool that is gaining popularity due to its algorithmic differentiation capability. The MPOPT package is validated with multiple test cases covering single-phase, multi-phase OCPs including a standard multi-stage launch vehicle ascent trajectory optimization problem.

The package is further extended with three custom developed adaptive grid refinement techniques to produce robust solutions. A unique feature of the proposed grid adaptation techniques is that the number of collocation nodes remain the same throughout the grid refinement process barring the need for NLP reformulation in between iterations. While the first two proposed grid refinement techniques are iterative in nature, the third method optimizes the segment widths along with the original optimization problem itself. The proposed algorithms have been tested on standard test cases for grid adaption techniques namely moon lander OCP and hypersensitive problem. Although all of the proposed methods are found to be successful in solving the test cases, the choice of the total number of collocation nodes is still a parameter chosen by the user.

Contents

List of Figures

- To my parents and family
For all the love and support over the years

Acknowledgement

First and foremost, I would like to thank *Petr Listov* for being an exceptional assistant for taking his time to guide me through this challenging project. Petr was always around to answer my queries or doubts.

Secondly, I would like to thank *Prof. Colin Jones*, for giving me a chance to work at the Automatic Control Laboratory and for always being accessible for reviewing the progress and for supervising the project. I am thankful for the thorough review of the software and valuable feedback.

Thirdly, I would like to thank *Dr. Rajeev U.P.* of Vikram Sarabhai Space Centre, ISRO for his expert guidance on the practical aspects of optimal control in aerospace systems.

Finally, I take this opportunity to thank my employer, Indian Space Research Organization (ISRO) and express my sincere gratitude to the visionary management of Liquid Propulsion Systems Center (LPSC), ISRO for giving me this exceptional opportunity to study while being part of ISRO. Especially, I would like to thank *Dr. Bijukumar K.S* for being an exceptional mentor, *Mr. N. Jayan* for his continuous guidance and support, *Mr. M.S Suresh* and *Dr. V. Narayanan* for enabling my master studies.

1 Introduction

Optimal control problem solvers derive control laws that optimize given performance index of engineering systems while satisfying the underlying system dynamics and operation constraints. Optimal control problem (OCP) formulations have become increasingly popular across many disciplines of science and engineering due to the increasing complexity of engineering systems and thanks to the advent of computational capabilities of modern computers. Due to the advances in computational control, the OCPs are not only being solved offline but also in real time [1]. Aerospace is one of the field that is known to have stringent requirement in real time computations due to the inherent complexity of the physical systems and robustness requirements. For example, the trajectory of a reentry vehicle which often translates into highly nonlinear, non-convex optimal control problem with stringent constraints needs to be solved in the order of few milliseconds to land safely [2] [3].

To solve such complex and challenging optimal control problems, one needs robust and computationally efficient solvers. While recent advances in nonlinear programming have lead to very efficient NLP solver code implementations [4] [5], there is still a need to develop not only robust but also computationally efficient numerical OCP solvers. Most of the existing robust OCP solvers are either commercial or require a steep learning curve to get started. While addressing some of the aforementioned limitations of existing OCP solvers, the present thesis concerns itself with the development of a new open source, extensible software for solving the multi-phase OCPs using pseudo-spectral collocation method fully utilizing the recent advances in NLP formulation methods [6]. The proposed package offers a perfect platform for the open source community, students, and the researchers likewise to experiment, extend, and customize with the latest developments in the field.

While the literature reveals that there are a rapidly growing list of optimal control problem solvers [1] [7] [8] [9] [10] [11], two of the solvers GPOPS-II, PSOPT are found to be popular choices in optimal control. GPOPS-II, a multi-phase OCP solver, written in MATLAB is based on pseudo-spectral collocation using Legendre-Gauss-Radau collocation points. Inbuilt with various adaptive mesh refinement techniques [12] and extensive documentation of the algorithm and examples of usage, it is a popular choice for commercial usage. The commercial nature of the software not only limits the usage but also prevents it to be extended. Another popular software PSOPT, written in C++ also featuring pseudo-spectral collocation is an open source solver [8]. Although PSOPT is well documented with extensive examples, it requires a steep learning curve to be able to solve new optimal control problems due to its nonintuitive interface.

The present package written in Python programming language is not only open source but also comes with a set of customized grid refinement techniques. Concisely written in Python, the solver is easy to use making it a perfect choice for the student, academic, and research community across educational and public institutions. The software is provided with various examples ranging from simple single-phase OCPs to complex Multi-phase OCPs. The package designed in modular fashion makes the implementation of customized OCP formulations easy with very little effort in programming. Inbuilt with various adaptive grid refinement techniques, the package provides competitive computational capabilities and solves complex problems such as rocket landing and multistage launch vehicle ascent optimization accurately. While python being the popular choice of programming language among open source community, the software is extensible making it accessible to a wider audience.

The thesis is divided into five sections. In the first section, a brief background of the OCP solution methods is discussed along with the scope of the package. Subsequently, a detailed description of the NLP transcription algorithm that is implemented in the package is given in the second section. The third section gives details of the solver implementation and the MPOPT algorithm. The fourth section demonstrates various aspects of the solver through three specific test cases from the literature. Finally, the last section discusses three adaptive grid refinement techniques developed and implemented as part of the thesis, along with descriptive examples.

2 Background

A generic optimal control problem in standard Bolza form is given in Equation 1, includes objective function (J), system dynamics (f), path constraints (g), terminal constraints (h) and box constraints on state and control variables. The OCP solvers find a control law that minimizes the given objective function while satisfying the constraints. In contrast to single-phase OCP defined in Equation 1, multi-phase OCP defined in Equation 2 contains multiple phases. While each phase of a multi-phase OCP can have different dynamics, constraints and objective function, the terminal states of each phase are related to their adjoining phase through event constraints.

Optimal Control Problem in Standard Bolza form (Single-phase)

$$\begin{aligned}
\min_{x,u,t_0,t_f,s} \quad & J = M(x_0,t_0,x_f,t_f,s) + \int_0^{t_f} L(x,u,t,s)dt && \text{(1a)}\\
\text{subject to} \quad & \dot{x} = f(x,u,t,s) && \text{(1b)}\\
& g(x,u,t,s) \leq 0 && \text{(1c)}\\
& h(x_0,t_0,x_f,t_f,s) = 0 && \text{(1d)}
\end{aligned}$$

Methods to solve optimal control problems

Numerical methods to solve OCPs can be broadly classified into two categories namely indirect methods and direct methods [13]. Indirect methods solve OCPs using Pontryagin's Maximum Principle which eventually results in two-point boundary value problem. While these methods are successfully employed in many aerospace applications, they are known to be sensitive to initial guess. In addition, initial guess for co-state and adjoint equations is also difficult to obtain. Another difficulty with indirect methods is that OCPs with path inequality constraints are difficult to solve [14] [13]. Direct methods overcome most of the problems encountered with indirect methods.

Direct methods transcribe the given OCP into a Nonlinear Programming problem (NLP) using various approximation methods i.e. single shooting, multiple shooting, collocation etc.. The NLP is then solved using various standard NLP solvers. Direct methods have gained popularity due to their ability to solve complex OCPs with ease. While there are different methods for the transcription of OCPs, pseudo-spectral methods have gained attention in direct optimization due to their potential fast convergence for continuous systems with high accuracy [14] [15].

Pseudo-spectral collocation method approximates the state and control variables using a set of orthogonal basis functions. The collocation roots are often chosen from the roots of a class of orthogonal polynomials called Jacobi polynomials. Pseudo-spectral methods are popular in numerical approximation of solutions to complex differential equations since they converge exponentially to the exact solution with increasing number of basis functions [16]. While the pseudo-spectral collocation has become popular choice in solving OCPs, an accurate transcription of continuous time OCP into a NLP is a challenging task. The transcription process itself involves approximation of the states and control trajectories using polynomials. While different choices for the polynomials exist, robust OCP solvers use various adaptive grid refinement techniques [9] to find robust solutions. The process of finding the robust solutions is not only challenging in nature but also a computationally expensive exercise limiting the application of existing OCP solvers in real time.

3 Multi-phase Optimal Control Problem Solver

Continuous time multi-phase OCP is transcribed into a nonlinear programming problem (NLP) using the Pseudo-spectral collocation method. The transcription process involves the approximation of derivative and integral operators. A generic, unique algorithm for the transcription of continuous multi-phase OCP which is valid for not only pseudo-spectral collocation but also for global collocation approximations is developed in this section. While in literature, transcription algorithms are often developed for specific polynomial approximations such as Legendre-Gauss, Legendre-Gauss-Radau, the present algorithm does not make any assumptions on the collocation roots which makes it very generic.

3.1 Continuous Multi-phase OCP

A multi-phase continuous time OCP with N_p number of phases is often represented in standard Bolza form as follows. Let, superscript p represents the phase index, $p \in [1, \dots N_p]$. Further, let $\mathbf{x}^p \in \mathcal{X}^p \subset \mathbb{R}^{n_x}$ represent the state vector of dimension n_x and $\mathbf{u}^p \in \mathcal{U}^p \subset \mathbb{R}^{n_u}$ represent the control vector of dimension n_u. Define $t_0^p \in \mathcal{T}_0^p \subset \mathbb{R}$ and $t_f^p \in \mathcal{T}_f^p \subset \mathbb{R}$ as the start and final times of the phase p respectively. Further, let $t^p \in [t_0^p, t_f^p]$ represent a continuous time variable in phase p. Finally, define $\mathbf{a}^p \in \mathcal{A}^p \subset \mathbb{R}^{n_a}$ as the set of algebraic parameters in phase p. Then continuous multi-phase OCP standard Boltza form is represented by Equation 2.

$$\begin{aligned}
&\min_{\mathbf{x}^p, \mathbf{u}^p, t_0^p, t_f^p, \mathbf{a}^p} && \sum_{p=1}^{N_p} \left[\mathcal{M}^{(p)} \left(\mathbf{x}^p(t_0^p), \mathbf{x}^p(t_f^p), t_0^p, t_f^p, \mathbf{a}^p \right) + \int_{t_0^p}^{t_f^p} \mathcal{L}^{(p)} \left(\mathbf{x}^p, \mathbf{u}^p, t^p, \mathbf{a}^p \right) dt \right] && \text{(2a)}\\
&\text{subject to} && \dot{\mathbf{x}}^p = \mathbf{f}^{(p)}(\mathbf{x}^p, \mathbf{u}^p, t^p, \mathbf{a}^p) && \text{(2b)}\\
& && \mathbf{g}^{(p)}(\mathbf{x}^p, \mathbf{u}^p, t^p, \mathbf{a}^p) \le 0 && \text{(2c)}\\
& && \mathbf{h}^{(p)}(\mathbf{x}^p(t_0^p), \mathbf{x}^p(t_f^p), t_0^p, t_f^p, \mathbf{a}^p) = 0 && \text{(2d)}\\
& && \underline{\mathbf{e}} \le \mathbf{e}(\mathbf{x}^1(t_0^1), \mathbf{x}^1(t_f^1), t_0^1, t_f^1, \mathbf{a}^1, \dots \mathbf{x}^{N_p}(t_0^{N_p}), \mathbf{x}^{N_p}(t_f^{N_p}), t_0^{N_p}, t_f^{N_p}, \mathbf{a}^{N_p}) \le \overline{\mathbf{e}} && \text{(2e)}\\
& && \mathbf{x}^p \in \mathcal{X}^p; \quad \mathbf{u}^p \in \mathcal{U}^p; \quad t_0^p \in \mathcal{T}_0^p; \quad t_f^p \in \mathcal{T}_f^p; \quad \mathbf{a}^p \in \mathcal{A}^p; && \text{(2f)}
\end{aligned}$$

Where $\mathcal{M}^{(p)}, \mathcal{L}^{(p)}$ represents the terminal cost, running cost in phase p often called as Mayer term and Lagrangian respectively. The system dynamics at time t^p are defined by the function $\mathbf{f}^{(p)}$. Likewise, the path constraints in phase p are defined by the function $\mathbf{g}^{(p)}$. Further, the equality constraints on terminal time and state vectors are defined using the function $\mathbf{h}^{(p)}$. In multi-phase formulation, each phase is connected to other phases through their terminal time and state vectors through event constraints represented by the function $\mathbf{e}$ in the standard Bolza formulation in Equation 2.

3.2 Discretization of multi-phase continuous OCP

The discretized approximation of the continuous time multi-phase OCP often results in a NLP given in Equation 3. The NLP is written in terms of matrices $(\mathbf{D}, \mathbf{F}, \mathbf{G}, \mathbf{H}, \mathbf{E}, \mathbf{W}, \mathbf{Q})$ which are defined based on the type of collocation approximation.

$$
\begin{aligned}
&\min_{\mathcal{X},\mathcal{U},\mathcal{T},\mathcal{A},\mathcal{P}} && \sum_{p=1}^{N_p} \left[M^p + \mathbf{W}_p^T \mathbf{Q}^p \right] && && \text{(3a)}\\
&\text{s. t.} && 0 = \mathbf{D}^p \mathbf{X}^p - \mathbf{F}^p \quad = 0 && \forall p = 1, \ldots, N_p && \text{(3b)}\\
& && -\infty = \mathbf{G}^p \qquad \leq 0 && \forall p = 1, \ldots, N_p && \text{(3c)}\\
& && 0 = \mathbf{H}^p \qquad = 0 && \forall p = 1, \ldots, N_p && \text{(3d)}\\
& && \underline{\mathbf{E}}^l = \mathbf{E}^l \qquad = \overline{\mathbf{E}}^l && \forall\, l \in \mathcal{P} && \text{(3e)}\\
& && \mathbf{X}^p \in \mathcal{X}^p; \quad \mathbf{U}^p \in \mathcal{U}^p; && \forall p = 1, \ldots, N_p && \text{(3f)}\\
& && t_0^p \in \mathcal{T}_0^p; \quad t_{N_c^p}^p \in \mathcal{T}_f^p; \quad \mathbf{a}^p \in \mathcal{A}^p; && \forall p = 1, \ldots, N_p && \text{(3g)}
\end{aligned}
$$

The discrete approximation of multi-phase continuous OCP presented in Equation 3 is a unique representation due to the following reasons. Firstly, all terms of the NLP are expressed in terms of matrices to ease the implementation in software. While other formulations express the system dynamics and constraints in matrix form, the objective function is typically given in scalar form with multiple summations. The present formulation introduces two matrices $(\mathbf{W}, \mathbf{Q})$ to construct stage cost in a given phase. Secondly, the transcription doesn't make any assumptions on the segment width in case of multi-segment pseudo-spectral collocation, the segment width is included in the definition of matrices $(\mathbf{W}, \mathbf{F})$ to enable implementation of various grid refinement schemes. In literature, the multi-segment transcription often deals with equal segment width collocation. Thirdly, the transcription holds for major collocation approximations namely global collocation, pseudo-spectral collocation etc. with only two matrices $(\mathbf{W}, \mathbf{D})$ changing from one approximation to another. Lastly, it is a generic representation that holds for single-segment and multi-segment pseudo-spectral collocation for not only a single phase but also multi-phase OCPs. While in literature, generally single segment formulations are extended to multi-segment collocation, the algorithm developed in this section formulates multi-segment approximation as an equivalent single segment approximation with appropriate construction of derivative and integral approximation matrices $(\mathbf{D}, \mathbf{W})$.

The algorithm used for the construction of matrices $(\mathbf{D}, \mathbf{F}, \mathbf{G}, \mathbf{H}, \mathbf{E}, \mathbf{W}, \mathbf{Q})$ for multi-segment pseudo-spectral collocation in the package and various terms in NLP are described in this section.

Multi-segment pseudo-spectral collocation

Let the domain of each phase (p) be split into N_s^p number of segments. Define d_s^p as the degree of the approximating polynomials for state and control vector in segment s of phase p. Let, N_c^p represent the total number of discrete collocation nodes in a given phase p excluding the starting node. Then, N_c^p is related to the polynomial degree d_s^p, given by Equation 4.

$$N_c^p = \sum_{s=1}^{N_s^p} d_s^p \tag{4}$$

Definition of optimization variables

Let $\mathcal{X} = \left\{\mathbf{X}^p; \quad \forall p = 1, \ldots N_p\right\}$ represent the set of discretized state variables. Further, let $\mathbf{x}_i^p$, $\mathbf{u}_i^p$, t_i^p represent the state vector, control vector and discrete time variable respectively at i^{th} collocation node in phase p. By construction $i \in [0, 1, \dots N_c^p]$. Further, let the state vector at collocation node i, of dimension n_x be represented with scalar entries as $\mathbf{x}_i^p = [x_i^1 \dots x_i^{n_x}]^T$.

$$\mathbf{X}^p = \begin{bmatrix} \mathbf{x}_0^p & \mathbf{x}_1^p & \dots & \mathbf{x}_{N_c^p}^p \end{bmatrix}^T = \begin{bmatrix} x_0^1 & \dots & x_0^{n_x} \\ x_1^1 & \dots & x_1^{n_x} \\ \vdots & \ddots & \vdots \\ x_{N_c^p}^1 & \dots & x_{N_c^p}^{n_x} \end{bmatrix}^p \tag{5}$$

Let control vector at collocation node i in phase p be given by $\mathbf{u}_i^p = [u_i^1 \dots u_i^{n_u}]^T$. Let $\mathcal{U} = \left\{\mathbf{U}^p; \quad \forall p = 1, \dots N_p\right\}$ be a set of discretized control vectors over all phases. Define the control vector ($\mathbf{U}^p$) for a given phase p as follows,

$$\mathbf{U}^p = \begin{bmatrix} \mathbf{u}_0^p & \mathbf{u}_1^p & \dots & \mathbf{u}_{N_c^p}^p \end{bmatrix}^T = \begin{bmatrix} u_0^1 & \dots & u_0^{n_u} \\ u_1^1 & \dots & u_1^{n_u} \\ \vdots & \ddots & \vdots \\ u_{N_c^p}^1 & \dots & u_{N_c^p}^{n_u} \end{bmatrix}^p \tag{6}$$

Define $\mathcal{T} = \left\{\left(t_0^p, t_{N_c^p}^p\right); \quad \forall p = 1, \dots N_p\right\}$ be a set of discretized start and terminal times over all phases. Further, let $\mathcal{A} = \left\{a^p; \quad \forall p = 1, \dots N_p\right\}$ be a set of algebraic parameters to be optimized. Multi-phase formulations require definition of relation between terminal variables of different phases. Let, $\mathcal{P} = \left\{(p_i, p_j) \mid p_i, p_j \in [1, \dots, N_p]; p_i > p_j\right\}$ define the set of predefined connecting phase pairs indices.

Mayer term approximation (M)

Now, Mayer term (M^p) corresponding to phase p can be written using the discrete variables at phase starting and terminal times, given by Equation 7.

$$M^p = \mathcal{M}^{(p)}\left(\mathbf{x}_0^p, \mathbf{x}_{N_c^p}^p, t_0^p, t_{N_c^p}^p, \mathbf{a}^p\right) \tag{7}$$

Running cost approximation ($\mathcal{L}$)

Let h_s^p be the segment width fraction for segment s in phase p, defined such that the actual width of segment s is given by Equation 8.

$$\text{Actual segment width} = \left(t_{N_c^p}^p - t_0^p\right) \times h_s^p \qquad \text{and} \tag{8}$$

$$\sum_{s=1}^{N_s^p} h_s^p = 1; \qquad \forall\, p = 1, \dots, N_p; \qquad 0 < h_s^p \leq 1 \tag{9}$$

Further, let r_i^p be the segment width fraction corresponding to i^{th} collocation node in phase p. Then, r_i^p is related to h_s^p, computed using Equation 10.

$$r_i^p = h_s^p; \qquad \text{if} \qquad \sum_{j=1}^{s-1} d_j^p \leq i < \sum_{j=1}^{s} d_j^p; \qquad \forall\, p \in [1, \dots, N_p] \tag{10}$$

Pseudo-spectral collocation basis is often defined in the interval [-1, 1]. Defining a variable $\tau \in [-1, 1]$ simplifies the approximation of integral operator. Let's define a variable $\tau \in [-1, 1]$ which maps start time and final time of each segment to -1 and +1 respectively. Further, let (t^s_{start}, t^s_{end}) represent start time and end time of a given segment s respectively. Then, absolute time in any given phase p can be computed using Equation 11.

$$t^p(\tau) = \frac{t^s_{\text{end}} - t^s_{\text{start}}}{2}\tau + \frac{t^s_{\text{end}} + t^s_{\text{start}}}{2} \tag{11}$$

$$t^s_{\text{start}} = \left(t^p_{N^p_c} - t^p_0\right) \times \sum_{i=1}^{s-1} h^p_i \tag{12}$$

$$t^s_{\text{end}} = \left(t^p_{N^p_c} - t^p_0\right) \times \sum_{i=1}^{s} h^p_i \tag{13}$$

Then, the integral operator in continuous time OCP is approximated by Equation 14 as follows.

$$\int_{t^p_0}^{t^p_f} \mathcal{L}^{(p)}\left(\mathbf{x}^p, \mathbf{u}^p, t^p, \mathbf{a}^p\right) dt = \sum_{s=1}^{N^p_s} \frac{\left(t^p_{N^p_c} - t^p_0\right) \times h^p_s}{2} \int_{-1}^{1} \mathcal{L}^{(p)}\left(\mathbf{x}^p(\tau), \mathbf{u}^p(\tau), t^p(\tau), \mathbf{a}^p\right) d\tau \tag{14a}$$

$$= \sum_{s=1}^{N^p_s} \frac{\left(t^p_{N^p_c} - t^p_0\right) \times h^p_s}{2} \sum_{i=0}^{d^p_s} w_{i,s}\mathcal{L}^{(p)}\left(\mathbf{x}^p_{is}, \mathbf{u}^p_{is}, t^p_{is}, \mathbf{a}^p\right) \tag{14b}$$

$$= \mathbf{W}^T_p \mathbf{Q}^p \tag{14c}$$

Where, the matrix $\mathbf{Q}^p$, for given phase index p is given by Equation 15.

$$Q^p = [q_i]_{(N^p_c+1)\times 1}; \qquad q_i = \frac{\left(t^p_{N^p_c} - t^p_0\right) \times r^p_i \times \mathcal{L}^{(p)}\left(\mathbf{x}^p_i, \mathbf{u}^p_i, t^p_i, \mathbf{a}^p\right)}{2} \qquad \forall i = 0, 1, \ldots, N^p_c \tag{15}$$

Further, the composite quadrature weights matrix ($\mathbf{W}^p$) is computed based on the selected quadrature scheme and collocation nodes. Specifically, if the quadrature weights corresponding to a set of given collocation points $\{\tau_1 \ldots \tau_{d^p_s}\}$ in segment s are denoted by $[\mathbf{w}_s]_{d^p_s\times 1}$ for all segments except the starting segment and the weights in first segment are defined by $[\mathbf{w}_1]_{(d^p_s+1)\times 1}$ including the starting node, then the composite quadrature weights matrix is given by Equation 16.

$$\mathbf{W}^p = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_{N^p_s} \end{bmatrix}^p_{(N^p_c+1)\times 1} \tag{16}$$

System dynamics matrix (F)

The matrix corresponding to the system dynamics (F) is constructed as follows,

$$\mathbf{F}^p = \left[\mathbf{f}^{pT}_i\right]_{(N^p_c+1)\times n_x} \qquad \forall p = 1, \ldots, N_p \tag{17}$$

$$\mathbf{f}^p_i = \frac{\left(t^p_{N^p_c} - t^p_0\right) \times r^p_i \times \mathbf{f}^{(p)}\left(\mathbf{x}^p_i, \mathbf{u}^p_i, t^p_i, \mathbf{a}^p\right)}{2} = [f^1, \ldots, f^{n_x}]^T \qquad \forall i = 0, 1, \ldots N^p_c \tag{18}$$

In short, the system dynamics matrix in phase p can be written as in Equation 19.

$$\mathbf{F}^p = \begin{bmatrix} f^1_0 & \cdots & f^{n_x}_0 \\ f^1_1 & \cdots & f^{n_x}_1 \\ \vdots & \ddots & \vdots \\ f^1_{N^p_c} & \cdots & f^{n_x}_{N^p_c} \end{bmatrix}^p \tag{19}$$

Path constraints matrix (G)

The matrix corresponding to path constraints (G) is constructed as follows. Let n_{pc}^p be the number of path constraints at a given collocation node in phase p. Then,

$$\mathbf{G}^p = \left[\mathbf{g}_i^{pT}\right]_{(N_c^p+1)\times n_{pc}^p}; \qquad \forall p = 1,\dots,N_p \tag{20}$$

$$\mathbf{g}_i^p = \mathbf{g}^{(p)}\left(\mathbf{x}_i^p, \mathbf{u}_i^p, t_i^p, \mathbf{a}^p\right) = [g^1,\dots,g^{n_{pc}^p}]^T \qquad \forall i = 0,1,\dots N_c^p \tag{21}$$

In short, the path constraints matrix in phase p can be written as in Equation 22.

$$\mathbf{G}^p = \begin{bmatrix} g_0^1 & \cdots & g_0^{n_{pc}} \\ g_1^1 & \cdots & g_1^{n_{pc}} \\ \vdots & \ddots & \vdots \\ g_{N_c^p}^1 & \cdots & g_{N_c^p}^{n_{pc}} \end{bmatrix}^p \tag{22}$$

Terminal constraints matrix (H)

The terminal constraints matrix (**H**) can be construction as follows. Let n_{tc}^p be the number of terminal constraints in phase p. Define $n_{tc} = \sum_{p=1}^{N_p} n_{tc}^p$. Then,

$$\mathbf{H}^p = \left[\mathbf{h}_p{}^T\right]_{n_{tc}\times 1}; \qquad \forall p = 1,\dots,N_p \tag{23}$$

$$\mathbf{h}_p^T = \mathbf{h}^{(p)}\left(\mathbf{x}_0^p, \mathbf{x}_{N_c^p}^p, t_0^p, t_{N_c^p}^p, \mathbf{a}^p\right) = [h^1,\dots,h^{n_{tc}^p}]^T \tag{24}$$

In short, the terminal constraints matrix in phase p can be written as in Equation 25.

$$\mathbf{H}^p = \begin{bmatrix} h_0^1 & \cdots & h_0^{n_{tc}} \\ h_1^1 & \cdots & h_1^{n_{tc}} \\ \vdots & \ddots & \vdots \\ h_{N_c^p}^1 & \cdots & h_{N_c^p}^{n_{tc}} \end{bmatrix}^p \tag{25}$$

Event constraints matrix (E)

The event constraints matrix (**E**) which connects different phases is constructed as follows. Let, $\underline{e}_x^l, \underline{e}_u^l, \underline{e}_t^l$ be the lower bound of discontinuity in states, controls and time across the linking phases pair $l \in \mathcal{P}$. Let, $\overline{e}_x^l, \overline{e}_u^l, \overline{e}_t^l$ represent the upper bound of discontinuity in states, controls and time variables respectively. Let n_e be the number of event constraints per phase pair $l \in \mathcal{P}$. Then,

$$\mathbf{E} = \left[\mathbf{e}^l\right]_{n_e\times 1}; \qquad \forall\, l \in \mathcal{P} \tag{26}$$

$$\mathbf{e}^l = \begin{bmatrix} \mathbf{x}_{N_c^{p_i}}^{p_i} - \mathbf{x}_0^{p_j} \\ \mathbf{u}_{N_c^{p_i}}^{p_i} - \mathbf{u}_0^{p_j} \\ t_f^{p_i} - t_0^{p_j} \end{bmatrix} \qquad \forall\, (p_i, p_j) = l \in \mathcal{P} \tag{27}$$

The lower bound and upper bound on the event constraints are as follows,

$$\underline{\mathbf{E}} = \begin{bmatrix} \underline{\mathbf{e}}^l \end{bmatrix}_{n_e \times 1}; \qquad \underline{\mathbf{e}}^l = \begin{bmatrix} \underline{\mathbf{e}}_x^l \\ \underline{\mathbf{e}}_u^l \\ \underline{\mathbf{e}}_t^l \end{bmatrix} \qquad \forall\, l \in \mathcal{P} \tag{28}$$

$$\overline{\mathbf{E}} = \begin{bmatrix} \overline{\mathbf{e}}^l \end{bmatrix}_{n_e \times 1}; \qquad \overline{\mathbf{e}}^l = \begin{bmatrix} \overline{\mathbf{e}}_x^l \\ \overline{\mathbf{e}}_u^l \\ \overline{\mathbf{e}}_t^l \end{bmatrix} \qquad \forall\, l \in \mathcal{P} \tag{29}$$

Derivative operator approximation (D)

Finally, the composite derivative interpolation matrix ($\mathbf{D}^p$) is constructed depending on the type of collocation approximation as given below.

Interpolation of states and controls in a single segment :

Let, the interpolation polynomial at i^{th} collocation node be represented by $\phi_i(\tau)$ where the collocation node indices are given by $i = 0, 1, \ldots d_s^p$ and the domain of the interpolating polynomial defined by $\tau \in [-1, 1]$. Further, let $\mathbf{x}_i, \mathbf{u}_i$ represent the discrete time state and control vectors at collocation node i in the given segment s. Then, by construction, state and control vectors in a given segment s can be approximated using interpolation polynomials using Equation 30.

$$\mathbf{x}^s(\tau) = \sum_{i=0}^{d_s^p} \mathbf{x}_i \phi_i(\tau); \qquad \mathbf{u}(\tau) = \sum_{i=0}^{d_s^p} \mathbf{u}_i \phi_i(\tau) \tag{30}$$

Define collocation nodes in a given segment s by $\tau_0, \tau_1 \ldots \tau_{d_s^p}$. The collocation methods satisfy dynamics and constraints exactly at the collocation nodes. Therefore, the interpolating polynomials have unique properties given in Equation 31.

$$\phi_i(\tau_i) = 1; \qquad \phi_i(\tau_j) = 0; \qquad \forall \quad i, j = 0, \ldots d_s^p; i \neq j \tag{31}$$

Choice of orthogonal polynomial basis :

Lagrange polynomials constructed as below are natural choice of interpolating polynomials in collocation.

$$l_i(\tau) = \prod_{j=0; j \neq i}^{d_s^p} \frac{\tau - \tau_j}{\tau_i - \tau_j} \tag{32}$$

In pseudo-spectral collocation, orthogonal polynomials offer a better approximation of integral and derivative operators and converge exponentially to the actual solution with respect to the degree of the approximating polynomial.

Interpolation of derivatives of states and controls in a single segment :

Lets assume that a collocation basis has been chosen and the interpolating polynomials are represented as $\phi_i(\tau)$ corresponding to i^{th} collocation node in segment s. Then, the derivative of the state and control in a given segment s is given by Equation 33.

$$\dot{\mathbf{x}}^s(\tau) = \sum_{i=0}^{d_s^p} \mathbf{x}_i \dot{\phi}_i(\tau) \quad \& \quad \dot{\mathbf{u}}(\tau) = \sum_{i=0}^{d_s^p} \mathbf{u}_i \dot{\phi}_i(\tau) \tag{33}$$

The derivative of state vectors in a given segment s at predefined nodes $(\tau_0, \tau_1 \dots \tau_{d_s^p})$ in matrix form is given by Equation 34.

$$\dot{\mathbf{x}}^s(\tau) = \begin{bmatrix} \dot{\phi}_0(\tau_0) & \dots & \dot{\phi}_{d_s^p}(\tau_0) \\ \vdots & \ddots & \vdots \\ \dot{\phi}_0(\tau_{d_s^p}) & \dots & \dot{\phi}_{d_s^p}(\tau_{d_s^p}) \end{bmatrix} \begin{bmatrix} \mathbf{x}_0 \\ \vdots \\ \mathbf{x}_{d_s^p} \end{bmatrix} = \mathbf{D}_s \mathbf{x}^s \tag{34}$$

Popular schemes for the selection of collocation points using Legendre polynomials include Legendre-Gauss (LG), Legendre-Gauss-Radau (LGR) [14] and Legendre-Gauss-Lobatto (LGL)[17] roots. Similar schemes exist for Chebyshev orthogonal polynomials (CG, CGR, CGL) [18].

The primary difference in a collocation roots scheme that concerns the composite differentiation matrix is that, if the roots include the terminal points of the interval [-1, 1] or not. For example, LG or CG schemes do not include both the terminal points whereas LGR or CGR schemes include one of the terminal nodes [19]. Unlike Gauss (LG) or Radau (LGR) schemes, Lobatto (LGL) scheme includes both the endpoints. In this package, we consider only schemes that include right endpoints in collocation roots.

Then, the composite differentiation matrix for a given phase p, with N_s^p number of segments is constructed as follows.

Consider the schemes where the right terminal node (+1) is included in the roots of the collocation scheme (i.e LGR, LGL, CGR, CGL etc.). Note that, in this case, the terminal node of a given segment is also the starting node of the next segment. Hence, the NLP formulation uses the same variable for both nodes. For this reason, the structure of the composite derivative matrix $(\mathbf{D}^p)$ resembles Equation 35 where the last column of each segment interpolation matrix is aligned with the first column of next segment interpolation matrix.

Structure of composite interpolation derivative matrix :

Lets consider a collocation approximation of a given phase p, where states and controls are approximated with polynomials of orders {4, 2, ..., 3, 2} respectively in each segment. Then the composite differentiation matrix $(\mathbf{D}^p)$ is constructed from differentiation matrices of each segment $(\mathbf{D}_s; s = 1, \dots N_s^p)$ aligned along the diagonal as shown in Equation 35. Each square block in Equation 35 represents a derivative interpolation matrix $(\mathbf{D}_s)$ for respective segment. Note that the terminal collocation node of each segment is aligned with the starting node of the next segment.

$$\mathbf{D}^p = \begin{bmatrix}
\bullet & \bullet & \bullet & \bullet & \bullet & & & & & & & & & \\
\bullet & \bullet & \bullet & \bullet & \bullet & & & & & & & & & \\
\bullet & \bullet & \mathbf{D}_1 & \bullet & \bullet & & & & & & & & & \\
\bullet & \bullet & \bullet & \bullet & \bullet & & & & & & & & & \\
\bullet & \bullet & \bullet & \bullet & \bullet & & & & & & & & & \\
 & & & & \bullet & \bullet & \bullet & & & & & & & \\
 & & & & \bullet & \mathbf{D}_2 & \bullet & & & & & & & \\
 & & & & \bullet & \bullet & \bullet & & & & & & & \\
 & & & & & & & \ddots & & & & & & \\
 & & & & & & & & \bullet & \bullet & \bullet & \bullet & & \\
 & & & & & & & & \bullet & \mathbf{D}_{N_s^p-1} & & \bullet & & \\
 & & & & & & & & \bullet & \bullet & \bullet & \bullet & & \\
 & & & & & & & & \bullet & \bullet & \bullet & \bullet & & \\
 & & & & & & & & & & & \bullet & \bullet & \bullet \\
 & & & & & & & & & & & \bullet & \mathbf{D}_{N_s^p} & \bullet \\
 & & & & & & & & & & & \bullet & \bullet & \bullet
\end{bmatrix}_{(N_c^p+N_s^p)\times(N_c^p+1)} \tag{35}$$

3.3 Interpolation of states and controls at non-collocation nodes

The OCP transcription defines states and controls at the collocation nodes. However, it is often required to compute the value of states and controls at nodes different from collocation nodes. Given state and control variables ($\mathbf{X}^p$, $\mathbf{U}^p$) at collocation nodes in phase p given by Equation 5 and Equation 6 respectively, the states and control variables at non-collocation nodes represented by $\mathbf{X}_I^p$, $\mathbf{U}_I^p$ respectively, can be computed via a projection operation defined in Equation 36.

$$\mathbf{X}_I^p = \mathbf{C}_I^p \mathbf{X}^p; \qquad \mathbf{U}_I^p = \mathbf{C}_I^p \mathbf{U}^p; \tag{36}$$

Similarly, the derivative at the interpolation points I can be computed via another projection operation using composite differentiation interpolation matrices ($\mathbf{D}_I^p$) as follows,

$$\dot{\mathbf{X}}_I^p = \mathbf{D}_I^p \mathbf{X}^p; \qquad \dot{\mathbf{U}}_I^p = \mathbf{D}_I^p \mathbf{U}^p; \tag{37}$$

Where the composite interpolation matrices ($\mathbf{C}_I^p; p = 1, \ldots, N_p$) and composite derivative interpolation matrices ($\mathbf{D}_I^p; p = 1, \ldots, N_p$) depend on the choice of non-collocation nodes and the polynomial basis. In multi-segment collocation, the non-collocation nodes can be chosen either relative to the segment grid or relative to the phase (i.e fixed grid).

Adaptive mesh refinement strategies which are discussed later in section 6, utilize the state and control variables at non-collocation points while refining the grid. In addition, residual in system dynamics evaluated at the non-collocation nodes is a good measure of the quality of the solution.

The construction of composite interpolation matrices and composite derivative interpolation matrices on a relative grid as well as the absolute grid is presented in this section.

Interpolation at predefined points relative to segments

Let $\mathcal{T}^s = \{\tau_i \mid \tau_i \in [-1, 1]; i = 0, \ldots, d_s^p\}$ be the set of collocation points in the segment s and phase p. Let $\mathcal{S}^\phi = \{\phi_i^s; i = 0, \ldots, d_s^p\}$ represent the set of basis polynomials corresponding to collocation nodes in $\mathcal{T}^s$. Then, by construction, the interpolated state and control vectors can be computed in terms of collocation variables ($\mathbf{x}_i, \mathbf{u}_i$) using Equation 38.

$$\mathbf{x}^s(\tau) = \sum_{i=0}^{d_s^p} \mathbf{x}_i \phi_i^s(\tau) \quad \& \quad \mathbf{u}(\tau) = \sum_{i=0}^{d_s^p} \mathbf{u}_i \phi_i^s(\tau) \tag{38}$$

Further, let $\mathcal{T}_I^s = \{\tau_i^I; i = 1, \ldots, I_s^p;\ \tau_i^I \in [-1, 1]\}$ be I_s^p number of predefined points that define the locations where the value of the states and controls are to be computed in the segment s and phase p. Then, the interpolated states in segment s can be computed using Equation 39. Note that the basis polynomials are evaluated at the interpolation points ($\mathcal{T}_I^s$).

$$\begin{bmatrix} \mathbf{x}_1^I \\ \vdots \\ \mathbf{x}_{I_s^p}^I \end{bmatrix} = \begin{bmatrix} \phi_0(\tau_1^I) & \ldots & \phi_{d_s^p}(\tau_1^I) \\ \vdots & \ddots & \vdots \\ \phi_0(\tau_{I_s^p}^I) & \ldots & \phi_{d_s^p}(\tau_{I_s^p}) \end{bmatrix}_{I_s^p \times (d_s^p+1)} \begin{bmatrix} \mathbf{x}_0 \\ \vdots \\ \mathbf{x}_{d_s^p} \end{bmatrix} = \mathbf{C}_s \mathbf{x}^s \tag{39}$$

Finally, the composite interpolation matrix ($\mathbf{C}_I^p$) for the given phase p can now be constructed by arranging the segment interpolation matrices $\mathbf{C}_s; s = 1, \ldots N_s$ (Equation 39) along the diagonal such that the last column of each matrix is aligned with the first column of interpolation matrix from the next segment as shown in Equation 35.

Similarly, the derivative interpolation matrix($\mathbf{D}_{I_s}$) at interpolation points ($\mathcal{T}_I^s$) is computed using Equation 40.

$$\begin{bmatrix} \dot{\mathbf{x}}_1^I \\ \vdots \\ \dot{\mathbf{x}}_{I_s^p}^I \end{bmatrix} = \begin{bmatrix} \dot{\phi}_0(\tau_1^I) & \dots & \dot{\phi}_{d_s^p}(\tau_1^I) \\ \vdots & \ddots & \vdots \\ \dot{\phi}_0(\tau_{I_s^p}^I) & \dots & \dot{\phi}_{d_s^p}(\tau_{I_s^p}) \end{bmatrix}_{I_s^p \times (d_s^p+1)} \begin{bmatrix} \mathbf{x}_0 \\ \vdots \\ \mathbf{x}_{d_s^p} \end{bmatrix} = \mathbf{D}_{Is}\mathbf{x}^s \tag{40}$$

The composite derivative interpolation matrix($\mathbf{D}_I^p$) corresponding to interpolation points in phase p can be constructed by arranging the individual derivative interpolation matrices of each segment (Equation 40) along the diagonal such that the last column of each segment is aligned with the first column of next segment as shown in Equation 35.

Interpolation onto a fixed grid relative to each phase

Interpolation using collocation points defined relative to each segment often results in a non-uniform grid when the segment widths are not equal. It is often useful to interpolate the states and controls onto a globally fixed grid instead of the local grid relative to each segment. Construction of the global interpolation matrix and derivative interpolation matrix with respect to a fixed grid is described in this section.

Let the time domain of a given phase p be mapped to a scaled domain $\mathcal{T} = [0,1]$. Further, define the fixed interpolation grid in time domain in the form of scaled interpolation time variables τ_i as $\mathcal{T}_I = \{\tau_i \mid \tau_i \in [0,1]; i > j \implies \tau_i > \tau_j; \forall i = 1, \dots, N_I;\}$. Then, the global interpolation points defined in $\mathcal{T}_I$ can be mapped onto the local grid relative to each segment as follows.

Let $\mathcal{H} = \{h_s^p; s = 1, \dots, N_s^p\}$ be a set of segment width fractions in phase p defined as in Equation 9. Further, let S^p denote a set of cumulative sum of the segment width fractions in phase p defined by Equation 41.

$$S^p = \{s_i;\ i = 0, \dots, N_s^p; \quad s_0 = 0; \quad s_{i+1} = s_i + h_i^s \quad \forall\, i = 1, \dots, N_s^p\} \tag{41}$$

Then, the interpolation points ($\mathcal{T}_I^k;\ k = 1 \dots N_s^p$) relative to segment k are uniquely related to the nodes in fixed grid $\mathcal{T}_I$, given by Equation 42.

$$\mathcal{T}_I^k = \left\{ \frac{\tau_j - s_{k-1}}{h_s^p} \mid s_{i-1} < \tau_j \leq s_i;\ s_i \in S^p;\ \tau_j \in \mathcal{T}_I;\ j = 1 \dots N_I;\ i = 1 \dots N_s^p \right\} \tag{42}$$

Finally, the composite interpolation matrices and derivative interpolation matrices corresponding to fixed grid are computed at local nodes in $\mathcal{T}_I^k$ using Equation 39 and Equation 40 respectively.

Estimation of residual in dynamics at non-collocated nodes

Let $\mathcal{T} = \{\mathcal{T}_I^k;\ k = 1 \dots N_k^p\}$ with $\mathcal{T}_I^k$ computed using Equation 42, represent the set of points over phase p where the residual is to be computed. The state and control vectors corresponding to the interpolation points ($\mathcal{T}$) are evaluated using composite interpolation matrix ($\mathbf{C}_I^p$) defined in Equation 36. Similarly, the derivative of states and controls are estimated using composite derivative interpolation matrix ($\mathbf{D}_I^p$) defined in Equation 37. Then, residual in dynamics at $\tau_i \in \mathcal{T}$ is estimated using Equation 43.

$$\mathbf{R}^p = \mathbf{D}_I^p \mathbf{X}^p - \mathbf{F}_I^p \tag{43}$$

Where, the interpolated system dynamics matrix $\mathbf{F}_I^p$ in phase p is given by Equation 46, computed as follows.

Let N_I^p represent the total number of interpolation points in $\mathcal{T}$. Then,

$$\mathbf{F}_I^p = \left[\mathbf{f}_i^{pT}\right]_{N_I^p \times n_x}; \qquad \forall p = 1, \ldots, N_p \tag{44}$$

$$\mathbf{f}_i^p = \frac{\left(t_{N_c^p}^p - t_0^p\right) \times r_i^p \times \mathbf{f}^{(p)}\left(\mathbf{x}_i^p, \mathbf{u}_i^p, t_i^p, \mathbf{a}^p\right)}{2} = [f^1, \ldots, f^{n_x}]^T \qquad \forall i = 1, \ldots N_I^p \tag{45}$$

Where the states and controls at interpolation points ($x_i^p, u_i^p;\ i = 1, \ldots N_I^p$) are computed using the projection operation defined in Equation 36.

In short, the interpolated system dynamics matrix in phase p is given by Equation 46.

$$\mathbf{F}_I^p = \begin{bmatrix} f_0^1 & \cdots & f_0^{n_x} \\ f_1^1 & \cdots & f_1^{n_x} \\ \vdots & \ddots & \vdots \\ f_{N_c^p}^1 & \cdots & f_{N_c^p}^{n_x} \end{bmatrix}^p \tag{46}$$

3.4 Algorithm for transcription of Multi-phase OCP

The transcription of continuous multi-phase OCP using collocation results in a Nonlinear programming problem (NLP) represented by Equation 3. The standard NLP solvers require the NLP to be represented in standard form given in Equation 47.

$$\min_{\mathbf{X}} \quad J(\mathbf{X}) \tag{47a}$$

$$\text{subject to} \quad \mathbf{LB} \leq \mathbf{F}(\mathbf{X}) \leq \mathbf{UB} \tag{47b}$$

$$\underline{\mathbf{X}} \leq \mathbf{X} \leq \overline{\mathbf{X}} \tag{47c}$$

The algorithm used to create a standard NLP (Equation 47) using collocation approximation described in subsection 3.2 is outlined in Algorithm 1.

Algorithm 1 NLP formulation of multi-phase OCP

```
1:  procedure NLP($N_p$ : No. phases, $N_s^p$ : No. seg. in each phase, $d_s^p$ : deg. poly. in each segment)
2:      for Each phase ($p$) in $[1, \dots, N_p]$ do
3:          $(\mathbf{X}^p, \mathbf{U}^p, t_0^p, t_{N_c^p}^p, \mathbf{a}^p) \leftarrow$ Create optimization variables    ▷ Variables (Equation 5)
4:          $(\mathbf{D}^p, \mathbf{W}_p) \leftarrow$ Compute collocation approximation    ▷ Collocation matrices (Equation 35)
5:          $(\mathbf{F}^p, \mathbf{LB}_f^p, \mathbf{UB}_f^p) \leftarrow$ Construct system dynamics matrix    ▷ Dynamics (Equation 18)
6:          $(\mathbf{G}^p, \mathbf{LB}_g^p, \mathbf{UB}_g^p) \leftarrow$ Create path constraints matrix    ▷ Path constraints (Equation 21)
7:          $(\mathbf{H}^p, \mathbf{LB}_h^p, \mathbf{UB}_h^p) \leftarrow$ Create terminal constraints vector    ▷ Terminal con. (Equation 24)
8:          $(\mathbf{A}^p, \mathbf{LB}_a^p, \mathbf{UB}_a^p) \leftarrow$ Additional constraints (if any)    ▷ Additional con. (subsection 3.6)
9:          $\mathbf{Q}^p \leftarrow$ Compute running costs matrix    ▷ Lagrangian (Equation 15)
10:         $\mathbf{F} \leftarrow (\mathbf{D}^p\mathbf{X}^p - \mathbf{F}^p, \mathbf{G}^p, \mathbf{H}^p, \mathbf{A}^p)$    ▷ Update NLP constraint vector
11:         $\mathbf{LB} \leftarrow (\mathbf{LB}_f^p = 0, \mathbf{LB}_g^p = -\infty, \mathbf{LB}_h^p = 0, \mathbf{LB}_a^p)$    ▷ Update NLP cons. lower bound
12:         $\mathbf{UB} \leftarrow (\mathbf{UB}_f^p = 0, \mathbf{UB}_g^p = 0, \mathbf{UB}_h^p = 0, \mathbf{UB}_a^p)$    ▷ Update NLP cons. upper bound
13:         $J \leftarrow (M^p + \mathbf{W}_p^T\mathbf{Q}^p)$    ▷ Update cost variable (Equation 3a)
14:         $\mathbf{X} \leftarrow (\mathbf{X}^p, \mathbf{U}^p, t_0^p, t_{N_c^p}^p, \mathbf{a}^p)$    ▷ Update NLP variables vector
15:         $\underline{\mathbf{X}} \leftarrow$ LowerBound($\mathbf{X}^p, \mathbf{U}^p, t_0^p, t_{N_c^p}^p, \mathbf{a}^p$)    ▷ Update NLP variables lower bound vector
16:         $\overline{\mathbf{X}} \leftarrow$ UpperBound($\mathbf{X}^p, \mathbf{U}^p, t_0^p, t_{N_c^p}^p, \mathbf{a}^p$)    ▷ Update NLP variables upper bound vector
17:     $(\mathbf{E}, \mathbf{LB}_e, \mathbf{UB}_e) \leftarrow$ Construct event constraints matrix    ▷ Event constraints (Equation 27)
18:     $\mathbf{F} \leftarrow \mathbf{E}$    ▷ Update NLP constraint vector
19:     $\mathbf{LB} \leftarrow \mathbf{LB}_e$    ▷ Update NLP constraint lower bound vector
20:     $\mathbf{UB} \leftarrow \mathbf{UB}_e$    ▷ Update NLP constraint upper bound vector
21:     return $(\mathbf{F}, \mathbf{LB}, \mathbf{UB}, J, \mathbf{X}, \underline{\mathbf{X}}, \overline{\mathbf{X}})$
```

3.5 NLP Sparsity pattern

Collocation methods are known to produce sparse NLP. Further, a multi-segment multi-phase OCP transcription often results in large NLPs. The sparsity pattern of such complex NLPs plays a significant role in the performance of NLP solvers. The NLP matrices created using Algorithm 1 ensure that the NLP is sparse by placing the collocation approximation matrices along the diagonal. The proposed algorithm results in a sparse NLP that compares very well with the sparsity pattern of GPOPS-II [9].

3.6 Additional constraints in NLP

In addition to the constraints in continuous time OCP in standard Bolza form, it may be required to impose additional constraints on state and control variables for various reasons. For instance, the derivative of a control signal may be constrained to enforce the actuator limitations in physical systems. This constraint is not part of the standard OCP formulation described in the previous section.

Another set of useful constraints may be defined as follows. While collocation approximation of the continuous time multi-phase OCP enforces system dynamics exactly at the collocation points, it is possible that the actual constraints are violated at non-collocation nodes. Generally, this problem is resolved by increasing the number of collocation points. However, it might be useful to apply additional constraints at non-collocation points using projection operations defined in Equation 36 to the original NLP.

In general, additional constraints on states and control vectors may be grouped into two categories.

1. Constraints on states and control vectors at collocation points. These may include following types.

   - Box constraints on slope of control or state variables

$$\mathbf{LB}_{dx} \le \mathbf{D}^p\mathbf{X}^p \le \mathbf{UB}_{dx}; \qquad \mathbf{LB}_{du} \le \mathbf{D}^p\mathbf{U}^p \le \mathbf{UB}_{du} \tag{48}$$

   - Box constraints on higher order derivatives

$$\mathbf{LB}_{d^nx} \le \mathbf{D}^{\mathbf{n}p}\mathbf{X}^p \le \mathbf{UB}_{d^nx}; \qquad \mathbf{LB}_{d^nu} \le \mathbf{D}^{\mathbf{n}p}\mathbf{U}^{p^n} \le \mathbf{UB}_{d^nu} \tag{49}$$

   - Continuity of control slope across the segments in Multi-segment collocation

$$\mathbf{D}^s\mathbf{x}^s_{d^p_s} = \mathbf{D}^{s+1}\mathbf{x}^{s+1}_{d^p_{s+1}} \qquad s = 1, \ldots, N^p_s - 1 \tag{50}$$

2. Constraints on states and control vectors at non-collocation points. These may include following types.

   - Box constraints on state and control vectors a non-collocation nodes (Equation 39)

$$\mathbf{LB}_x \le \mathbf{C}^p_I\mathbf{X}^p \le \mathbf{UB}_x; \qquad \mathbf{LB}_u \le \mathbf{C}^p_I\mathbf{U}^p \le \mathbf{UB}_u \tag{51}$$

   - Constraints on residual in system dynamics at non-collocation nodes (Equation 43)

$$\mathbf{LB}_r \le \mathbf{D}^p_I\mathbf{X}^p - \mathbf{F}^p_I(\mathbf{X}^p_I, \mathbf{U}^p_I, t^p, \mathbf{a}^p) \le \mathbf{UB}_r \tag{52}$$

   - Path constraints enforced at non-collocation nodes

$$\mathbf{LB}_g \le \mathbf{G}^p(\mathbf{X}^p_I, \mathbf{U}^p_I, t^p, \mathbf{a}^p) \le \mathbf{UB}_g \tag{53}$$

In collocation, aforementioned constraints are easily implemented. Specifically, features to enable box constraints on control slope, box constraints on control variables at non-collocated nodes often results in better approximation of solution. Further, multi-segment pseudo-spectral collocation often results in optimal solution that has discontinuity in slope of control across segments which may not be preferred for various reasons. The continuity of control slope across multi-segment collocation is easily implemented with appropriate construction of interpolation matrices using Equation 50.

4 Implementation of the multi-phase OCP solver

The solver package implements various modules to solve multi-phase optimal control problems in Python programming language. At the outset, the solver programmed in Python takes full advantage of automatic differentiation feature provided by CasADi [20]. In addition, various the NLP solver plugins available with CasADi which include popular solvers IPOPT, SNOPT, BlockSQP etc. are used to solve the NLP formulation. The modular design of the software allows independent implementation of four objectives in the solver namely OCP problem definition, Collocation approximation, NLP definition followed by call to appropriate NLP solver and post processing. While the NLP transcription algorithm is presented in previous section, the details of implementation of remaining modules and various design choices are discussed in this section.

4.1 Collocation approximation

Collocation points

Collocation roots are central to the convergence property of pseudo-spectral collocation. While the software design allows custom definition of collocation points, well known Legendre-Gauss-Radau (LGR), Legendre-Gauss-Lobatto (LGL), Chebyshev-Gauss-Lobatto (CGL) roots are presently supported. Moreover, scipy.special modules allows the computation of roots of various orthogonal polynomials popular in spectral methods including Jacobi polynomials.

The three schemes supported in the present implementation derive collocation roots using Legendre or Chebyshev polynomials. The relative location of nodes vary depending on the choice of orthogonal basis. For instance, the nodes in LGR, LGL and CGL schemes for approximating polynomials of order 4 are shown in Figure 1. Note that the LGR collocation does not include the lower bound of the default domain.

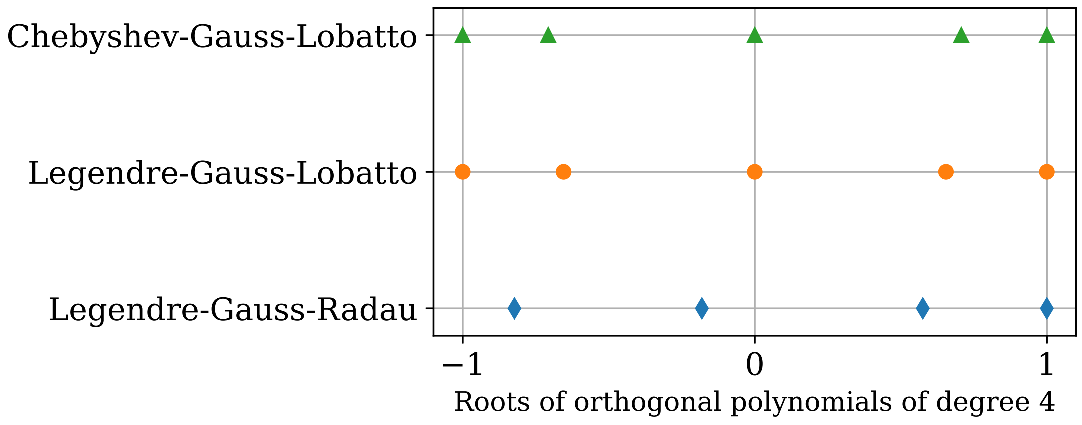


Figure 1: Location of collocation nodes for various orthogonal polynomials of order 4

Orthogonal basis and interpolation polynomials

The software implements interpolation based on Lagrange polynomials (Equation 31). While Lagrange polynomials are easily defined for any collocation basis, orthogonal collocation roots are most preferred due to their spectral property. Further, differentiation approximation and quadrature rules are computed numerically using the `numpy polynomials` package. One of the major drawback of numerical approximation is that the noise in derivative approximation is amplified with the degree of the approximating polynomials. However, from experience it is found that the numerical approximations work very well for polynomials of order up to 20.

While the software design allows for the exact definition of derivative operator and Gaussian quadrature for Legrange and Chebyshev orthogonal basis it is still a work in progress.

4.2 Initial solution estimate

NLP solver often requires a good initial solution to solve the NLP accurately. Further, a good initial solution estimate not only makes it easy to solve but also makes NLP converge faster to an optimal solution. From experience of solving various OCPs it is observed that there are two popular ways to succeed in solving complex OCPs. While the first method is to simplify the OCP formulation, second method is to have an good initial guess. Further, it is observed that a good initial estimate is often necessary to solve highly non-linear OCPs.

Initialization method in the present package is based on linear interpolation of initial guess for states and controls at start time and final times of each phase defined by the user while defining the OCP. If $x0_0^p, xf_0^p$ represent the initial guess for the state vector in phase p at the expected start time and final times of phase p $(t0_0^p, tf_0^p)$ respectively, then the state, control vectors at intermediate collocation nodes (t) are computed based on linear interpolation given in Equation 54, Equation 55 respectively.

$$x_0^p(t) = x0_0^p + \frac{(xf_0^p - x0_0^p)}{(tf_0^p - t0_0^p)}(t - t0_0^p) \tag{54}$$

$$u_0^p(t) = u0_0^p + \frac{(uf_0^p - u0_0^p)}{(tf_0^p - t0_0^p)}(t - t0_0^p) \tag{55}$$

4.3 OCP definition

The solver requires a well-defined optimal control problem to solve the problem. While the structure of the OCP in Bolza form makes it simple to define the OCP, the implementation of a well-defined OCP is not easy for multiple reasons. First of all, not all OCPs in practice have all the elements of standard Bolza form. For example, some OCPs many not have path constraints or box constraints on states and controls. Secondly, some parameters of the OCP have default values for most of the OCPs, hence it might be useful to initialize them accordingly. For example, it might useful to create a phase link pairs assuming the multi-phase OCP has phases linked in series which is the most often the case. At present, all elements of a multi-phase OCP in standard Bolza form are initialized with default values given in Table 1 at the initialization of OCP class. Subsequently, the user updates definition of appropriate variables.

For completeness, the OCP definition along with the default values of initialized parameters and syntax for the definition of all elements of standard OCP is given in Table 1.

4.4 Additional options in OCP definition

Additional constraints defined in subsection 3.6 are often required in practice. Some of the additional constraints defined in subsection 3.6 are implemented in the solver, software design allows for the implementation of custom additional constraints if required. The predefined additional constraints already implemented include box constraints on slope of the control signal at collocation nodes and box constraints on control signal itself at non-collocation nodes. Further, constraints on terminal control vectors of each segment to ensure continuity of control slope across segments in multi-segment collocation is also implemented.

Table 1: OCP definition

Sl.no	**Name**	**Default**	**Initialization**	**Example**
		Primary objects of OCP		
1	Dynamics	**0**	`dynamics[phase] = function(x, u, t, a)`	`dynamics[0] = lambda x, u, t, a: [x[1], u[0]]`
2	Lagrange term	**0**	`running_costs[phase] = function(x, u, t, a)`	`running_costs[0] = lambda x, u, t, a: u[0]*u[0]`
3	Mayer term	**0**	`terminal_costs[phase] = function(xf,tf,x0,t0,a)`	`terminal_costs[0] = lambda xf, tf, x0, t0, a: -tf`
4	Path constraints	None	`path_constraints[phase] = function(x, u, t, a)`	`path_constraints[0] = lambda x,u,t,a: [x[0]+x[1]-1, x[0]*u[0]-1]`
5	Terminal constraints	None	`terminal_constraints[phase] = function(xf,tf,x0,t0,a)`	`terminal_constraints[0] = lambda xf, tf, x0, t0, a: [xf[0]-1, xf[1] - 2]`
6	Initial state	**0**	x00[phase]	x00[0] = [1, 1]
		Optional Box constraints		
1	States	$[-\infty, \infty]$	lbx[phase], ubx[phase]	lbx[0] = -2; ubx[0] = +2
2	Controls	$[-\infty, \infty]$	lbu[phase], ubu[phase]	lbu[0] = -1; ubu[0] = +1
3	Start time	$[0, 0]$	lbt0[phase], ubt0[phase]	lbt0[0] = 0; ubt0[0] = 0
4	Final time	$[\infty, \infty]$	lbtf[phase], ubtf[phase]	lbtf[0] = 1; ubtf[0] = 1
5	Parameters	$[-\infty, \infty]$	lba[phase], uba[phase]	lba[0] = -5; uba[0] = 5
		Optional parameters : Scaling		
1	Scale states	**1**	scale_x[phase]	scale_x[0] = [1/5.0, 1/10.0]
2	Scale controls	**1**	scale_u[phase]	scale_u[0] = 1/10.0
3	Scale time	**1**	scale_t[phase]	scale_t[0] = 1/5.0
4	Scale parameters	**1**	scale_a[phase]	scale_a[0] = 1/100.0
		Optional parameters to initialize the solution		
1	Guess for xf	**0**	xf0[phase]	xf0[0] = [1, 2]
2	Control at t0	**0**	u00[phase]	u00[0] = 1
3	Control at tf	**0**	uf0[phase]	uf0[0] = 0.5
4	Initial guess for t0	**0**	t00[phase]	t00[0] = 0
5	Initial guess for tf	∞	tf0[phase]	tf0[0] = 1
		Options related to Multiple phase OCP		
1	Phase links	Serial	phase_links	phase_links = [(0, 1), (1, 3), (0, 2)]
2	Event cons. bounds[1]	**0**	lbe[link]; ube[link]	lbe[link] = [0, 0]; ube[link] = [0,0]
		Additional constraints		
1	Constraint on $\dot{u}$	False	diff_u[phase]	diff_u[0] = 1 # (Enabled)
2	Box Constraint on $\dot{u}$	[-15, 15]	lbdu[phase];ubdu[phase]	lbdu[0] = -3; ubdu[0] = 3
3	Box Cons. on u^{mid}	True	midu[phase]	midu[0] = 0
4	$\dot{u}$ cont. across seg.	False	du_continuity[phase]	du_continuity[0] = 1

4.5 Algorithm to solve the multi-phase OCP

The continuous time OCP is implemented as a class object in python is initialized and the elements of OCP are defined using the syntax in Table 1. The steps in solving the OCP are outlined as follows,

Algorithm 2 MPOPT algorithm

```
1: procedure MPOPT(OCP, $N_s^p$, $d_s^p$, collocation_scheme)
2:     $(\mathbf{F}, \mathbf{LB}, \mathbf{UB}, J, \mathbf{Z}, \underline{\mathbf{Z}}, \overline{\mathbf{Z}}) \leftarrow$ Initialize NLP matrices        ▷ Create NLP Algorithm (1)
3:     $Z_0 \leftarrow$ Create initial solution estimate        ▷ Initial solution
4:     $h_s^p \leftarrow \frac{1}{N_s^p}$        ▷ Initialize segment width fractions
5:     $\mathcal{Z} \leftarrow$ Solve NLP($(\mathbf{F}, \mathbf{LB}, \mathbf{UB}, J, \mathbf{Z}, \underline{\mathbf{Z}}, \overline{\mathbf{Z}})$, $Z_0$, $h_s^p$)        ▷ Solve NLP
6:     $\{\mathbf{X}^i, \mathbf{U}^i, t_0^i, t_f^i, a^i;\ i = 1, \ldots, p\} \leftarrow$ Retrieve OCP solution from the NLP solution $\mathcal{Z}$
7:     return $\{\mathbf{X}^i, \mathbf{U}^i, t_0^i, t_f^i, a^i\}$        ▷ Return OCP solution
```

NLP solvers

The NLP formulation in the package can be solved with various solver plugins available in CasADi. One of the popular solver based on barrier methods is IPOPT. While the present package is tested on various examples using IPOPT, other available solvers such as 'SNOPT', 'sqpmethod' can be used as well for solving the NLP.

Profiling the algorithm

The algorithm 2 has three distinct steps namely creating NLP matrices, initializing the solution and solving the NLP. A good knowledge of the computational time not only enables one to optimize the solver implementation but also enables creating of efficient grid refinement techniques.

A test case OCP given in Equation 59 is solved using the package by varying the number of collocation nodes. The time taken by the algorithm in various steps of computation is recorded and plotted in Figure 2. For the simple moon lander OCP solved using the package, it is found that the NLP creation process is computationally expensive compared to NLP solve time. However, this trend can not be generalized as the time required for solving OCPs with complex dynamics would be much higher than the NLP creation time as it is problem dependant.

4.6 Post processing

The package comes with a post processing module which not only processes the optimizer solution but also provides various methods for interactive visualization. It is written in order to provide easy access to the optimizer solution to user without having to deal with the interpolation polynomials used in pseudo-spectral collocation. For example, the values of states and controls can be obtained using single command as follows.

```
(x, u, t) = post.get_data()
```

Similarly the data can be visualized in a python plot as follows.

```
fig, axs = post.plot_phases()
fig, axs = post.plot_phase(0)
fig, axs = post.plot_x()
fig, axs = post.plot_u()
```

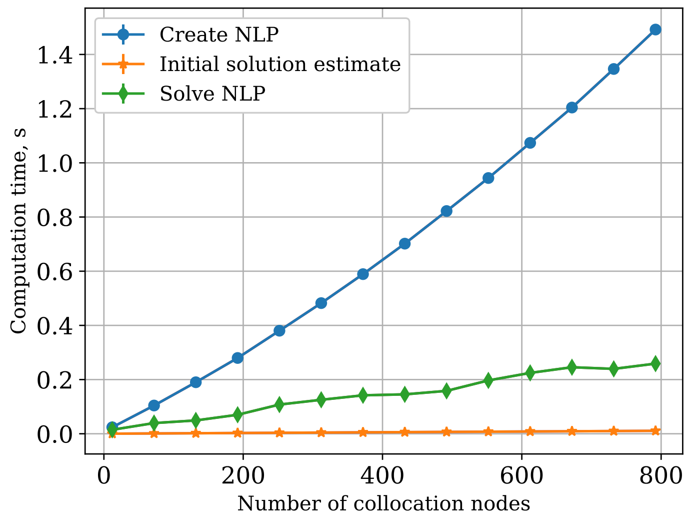


Figure 2: Profiling OCP solver algorithm for moon lander OCP: Comparison of NLP creation time and NLP solve time by varying number of collocation nodes

The post processing methods come with various options such as re-scaling before accessing the solution and interpolating the solution onto uniform grid. Further, it is possible to access or plot the data corresponding to specific phases.

The package comes with multiple built-in sophisticated methods to process data of multiple phases and plot the variables in various plots as required by the user. The post processing package is built as an independent class that can be customized or extended depending on user requirements.

4.7 Package hosting and source code details

The MPOPT package is officially hosted on The Python Package Index (PyPI) [2], making it easily accessible for all to experiment with a single command from the terminal.

```
pip install mpopt
```

The source code is hosted in GitHub[3] featuring continuous integration and test coverage. While the source code is extensively documented with doc strings, a Jupyter notebook [4] demonstrates various features and limitations of the software. A set of examples provided in the form of standalone scripts show the extended usage of the software along with various hyper parameters in the proposed adaptive grid refinement methods.

[2]https://pypi.org/project/mpopt/

[3]https://github.com/mpopt

[4]https://github.com/mpopt/mpopt/blob/master/notebooks/getting_started.ipynb

5 Examples

The multi-phase OCP solver packaged implemented in Python is easy to use and solves many standard optimal control problems efficiently. This section gives simple examples of software usage and demonstrates various features of the package.

Though there exist multitude of OCP solver test cases in literature, only three OCPs are solved in this section. The first case is a single phase OCP formulation namely Van der Pol oscillator given in Equation 56. Second example is a test case for the multi-phase optimal control problem solver named Two-phase Schwartz problem, given in Equation 57. While the first two test cases are chosen to be simple to demonstrate various features of the package, a more complex multi-phase problem dealing with the ascent optimization of multi-stage launch vehicle trajectory is selected as the final example.

5.1 Single phase

Continuous time Van der Pol oscillator OCP formulation with two states and one control variable is given in Equation 56.

Van der Pol Oscillator

$$\min_{x,u} \quad J = 0 + \int_{t_0}^{t_f} \left(x_0^2 + x_1^2 + u^2\right) dt \tag{56a}$$

$$\text{subject to} \quad \dot{x_0} = (1 - x_1^2) \times x_0 - x_1 + u \tag{56b}$$

$$\dot{x_1} = x_0 \tag{56c}$$

$$x_1 \geq -0.25 \tag{56d}$$

$$-1 \leq u \leq 1 \tag{56e}$$

$$x_0(t_0) = 0;\ x_1(t_0) = 1; \tag{56f}$$

$$t_0 = 0.0;\ t_f = 10 \tag{56g}$$

The Van der Pol OCP is initialized, solved and results are post processed in under 10 lines of effective python code as follows. The syntax follows from the OCP initialization examples given in Table 1.

```python
#Van der Pol oscillator OCP from https://web.casadi.org/docs/
from mpopt import mp

# Define OCP
ocp = mp.OCP(n_states=2, n_controls=1)
ocp.dynamics[0] = lambda x, u, t: [(1 - x[1] * x[1]) * x[0] - x[1] + u[0], x[0]]
ocp.running_costs[0] = lambda x, u, t: x[0] * x[0] + x[1] * x[1] + u[0] * u[0]
ocp.lbx[0][1] = -0.25
ocp.lbu[0], ocp.ubu[0] = -1.0, 1.0
ocp.x00[0] = [0, 1]
ocp.lbtf[0], ocp.ubtf[0] = 10.0, 10.0

# Create optimizer, solve and post process
mpo = mp.mpopt(ocp, n_segments=1, poly_orders=20, scheme="LGR")
solution = mpo.solve()
post = mpo.process_results(solution, plot=True)
```

Listing 1: Van der Pol Oscillator OCP solution using MPOPT

The package implements different collocation roots using Legendre and Chebyshev polynomials. The Van der Pol OCP is easily solved using LGR, LGL and CGL roots by changing the collocation scheme while initializing the optimizer. The optimal value of the cost reported by the solver is (2.87373 2.87329, 2.87397) for 15 roots of LGR, LGL and CGL schemes confirms the compatibility of various orthogonal polynomial roots in the package. Further, the states and control trajectories shown in Figure 3a indicate the solution of global collocation approximation where a single polynomial of order 15 is used, tics indicating the collocation node locations. Lastly, the OCP is solved using multi-segment collocation with 5 number of segments and polynomials of order 15. The states and control trajectories for multiple-segment collocation shown in Figure 3b confirm that the single phase problems can be solved with multi-segment pseudo-spectral collocation using the package.

5.2 Multi phase

Two-phase Schwartz problem written in the standard Bolza form is given in Equation 57. Note that the constraints in phase-1 differ from phase-2 while dynamics remain same in both phases.

Two-phase Schwartz problem

$$\min_{x,u} \quad J = 5(x_0(t_f)^2 + x_1(t_f)^2) + \int_{t_0}^{t_f} 0 dt \tag{57a}$$

$$\text{subject to} \quad \dot{x_0} = x_1 \tag{57b}$$

$$\dot{x_1} = u - 0.1(1 + 2x_0^2)x_1 \tag{57c}$$

$$\textit{Phase 1:} \quad 1 - 9(x_0 - 1)^2 - \left(\frac{x_1 - 0.4}{0.3}\right)^2 \leq 0 \tag{57d}$$

$$x_1 \geq -0.8 \tag{57e}$$

$$-1 \leq u \leq 1 \tag{57f}$$

$$x_0(t_0) = 1;\ x_1(t_0) = 1; \tag{57g}$$

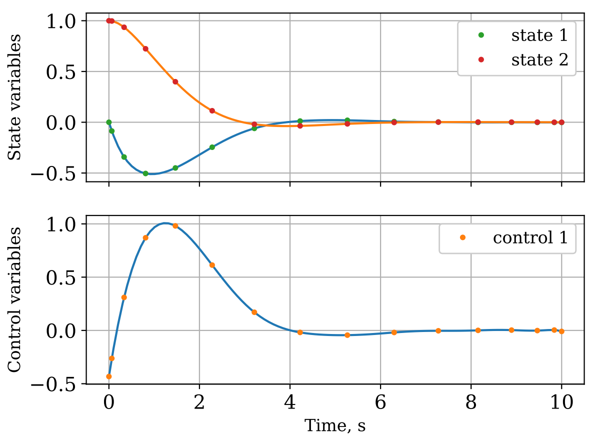
$$t_0 = 0;\ t_f = 1 \tag{57h}$$

$$\textit{Phase 2:} \quad t_0 = 1;\ t_f = 2.9 \tag{57i}$$

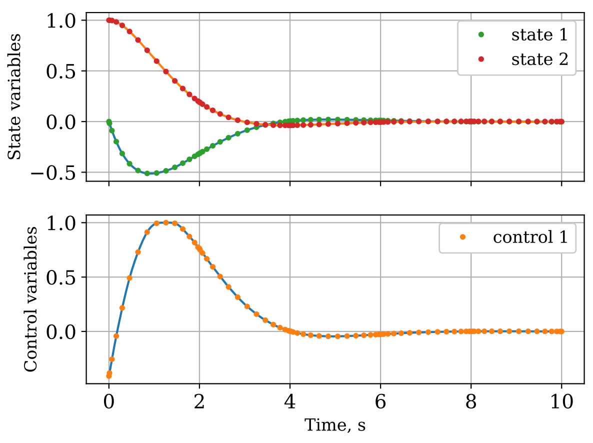
$$x \in \mathbb{R}^2;\ u \in \mathbb{R} \tag{57j}$$

The program listing given below, shows that the Two-phase Schwartz problem is initialized, solved and results are processed in under 15 lines of effective python code.

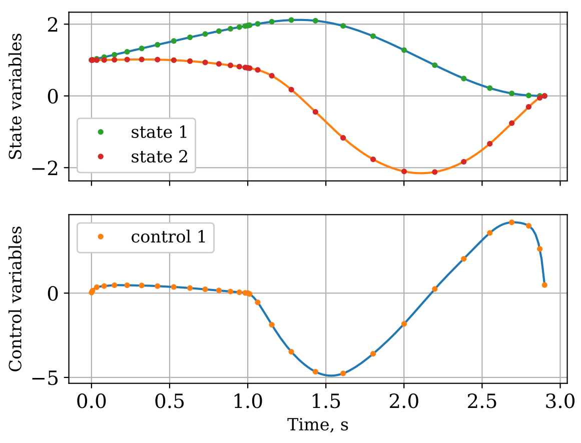
```
from mpopt import mp

# Initialize OCP
ocp = mp.OCP(n_states=2, n_controls=1, n_phases=2)
dynamics = lambda x, u, t:[x[1], u[0] - 0.1 * (1.0 + 2.0 * x[0] * x[0]) * x[1]]
ocp.dynamics = [dynamics, dynamics]
ocp.path_constraints[0] = lambda x, u, t: [1.0 - 9.0 * (x[0] - 1) * (x[0] - 1) -
    (x[1] - 0.4) * (x[1] - 0.4) / (0.3 * 0.3)]
ocp.terminal_costs[1] = lambda xf, tf, x0, t0: 5 * (xf[0]**2 + xf[1]**2)

ocp.x00[0] = ocp.x00[1] = [1, 1]
ocp.xf0[0], ocp.xf0[1] = [1, 1], [0, 0]
ocp.lbx[0][1] = -0.8
ocp.lbu[0], ocp.ubu[0] = -1, 1
```



(a) State and control trajectories for Van der Pol Oscillator OCP using Legendre-Gauss-Radau roots (Single segment collocation)



(b) State and control trajectories for Van der Pol Oscillator OCP using Legendre-Gauss-Radau roots (Multi segment collocation - 5 segments with polynomials of degree 15)



(c) State and control trajectories for Two-phase Schwartz OCP using LGR collocation nodes (Direct collocation with polynomial of order 15)

Figure 3: Simple single phase and multi-phase Optimal control problems validating the package using direct collocation and multi-segment collocation approximations

```
ocp.lbt0[0], ocp.ubt0[0] = 0, 0
ocp.lbtf[0], ocp.ubtf[0] = 1, 1
ocp.lbtf[1], ocp.ubtf[1] = 2.9, 2.9

mpo, post = mp.solve(ocp, n_segments=1, poly_orders=15, "LGR", plot=True)
```

Listing 2: Two-phase Schwartz OCP solution using MPOPT

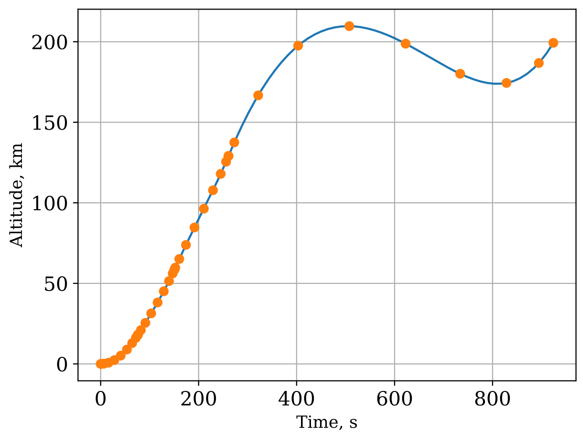
The problem is easily solved using the package and the trajectories of states and controls are shown in Figure 3c. The optimal value reported by the solver is 10^{-27} compares very well with the actual solution.

Multi-stage launch vehicle

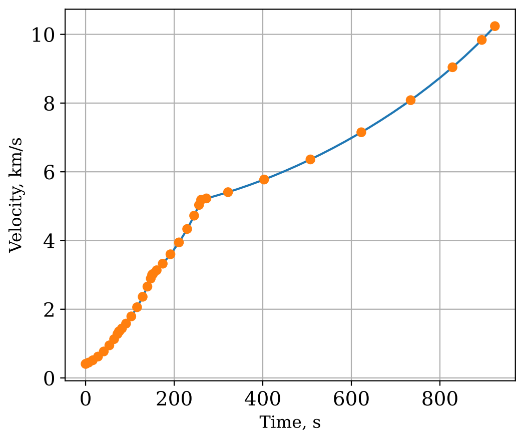
The third and last example solved using the package is a popular Aerospace application that finds the optimum ascent trajectory of a given multi-stage launch vehicle. We consider the standard formulation of delta-III launch vehicle popularly used in literature for the validation of multiphase solvers.

Although the problem is well defined in literature [9], none of the formulations are found to be complete for implementing in software. Therefore, a complete standalone formulation of the ascent trajectory optimization problem is given in Equation 58 in standard Bolza form.

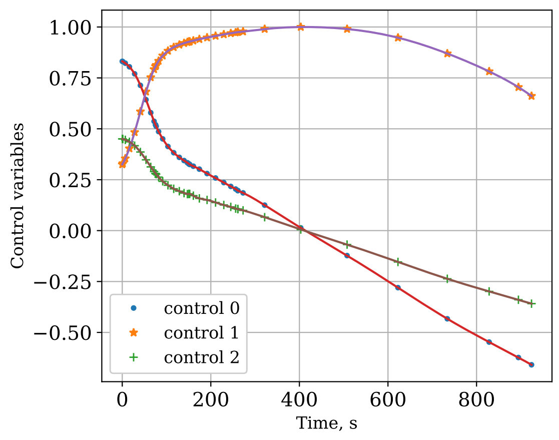
The source code of the OCP formulation using the present package is given in Appendix Listing 10. Due to the highly non-linear nature of drag term in the system dynamics, the OCP is solved in two steps, first by disabling the drag term and then with drag enabled. When initialized with initial guess from the first step, the solver converges in milliseconds against few seconds of time reported in the literature [21]. The optimal value obtained by the MPOPT is 7529.7129kg comparable with GPOPS-II, PSOPT, SOCS values (7529.7123, 7529.661, 7529.7125) [22]. The terminal time reported by the solver is 924.1393s is in good agreement with the solution from PSOPT 924.1413s. The difference in the optimal solution is primarily due to the underlying approximation scheme and the number of collocation nodes used. For instance, GPOPS-II solution uses an adaptive grid refinement scheme to obtain the best possible estimate. However, other schemes employ fixed collocation nodes in each segment. The altitude and velocity profile are shown in Figure 4a, Figure 4b respectively. The optimum thrust control vector is shown in Figure 4c. Finally, the evolution of the mass of the launch vehicle is plotted in Figure 4d.



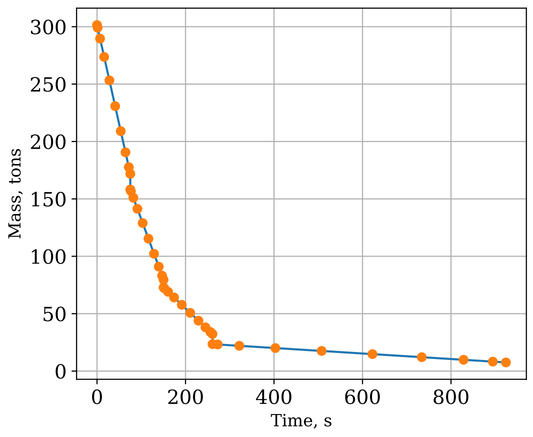
(a) Optimal altitude profile of the Multistage launch vehicle



(b) Optimal velocity profile of the Multistage launch vehicle



(c) Optimal thrust vector control profile of the Multistage launch vehicle



(d) Mass evolution of Multistage launch vehicle

Figure 4: Multiphase optimization of Delta-III launch vehicle trajectory (1 segment and 9 collocation nodes in each phase)

$$\min_{x,u} \qquad J = -x_6(t_f) + \int_{t_0}^{t_f} 0 dt$$

subject to

Define

$$\mathbf{r} = [x_0, x_1, x_2];\ \mathbf{v} = [x_3, x_4, x_5];\ m = x_6;\ \mathbf{u} = [u_0, u_1, u_2];$$
$$\mu = 3.986012e14;\ \omega_e = [0, 0, 7.29211585e-5]^T;\ R_e = 6378145;$$
$$\rho_0 = 1.225;\ H_0 = 7200;\ C_d = 0.5;\ A^{\text{ref}} = 4\pi;$$
$$T^0 = 4854100;\ T^1 = 2968600;\ T^2 = 1083100; T^3 = 110094;$$
$$\dot{m}^0 = 1723.273;\ \dot{m}^1 = 1044.682;\ \dot{m}^2 = 366.092;\ \dot{m}^3 = 24.029;$$

Dynamics:

$$\dot{\mathbf{r}} = \mathbf{v}$$
$$\dot{\mathbf{v}} = -\frac{\mu}{\|\mathbf{r}\|^3}\mathbf{r} + \frac{T^p}{m}\mathbf{u} - \frac{C_d A^{\text{ref}} \rho_0 \big\|\mathbf{v} - (\omega_e \times \mathbf{r})\big\| \left(\mathbf{v} - (\omega_e \times \mathbf{r})\right) e^{\frac{\|\mathbf{r}\| - R_e}{H_0}}}{2m}$$
$$\dot{m} = -\dot{m}^p \qquad\qquad p = 0, 1, 2, 3$$

Path constraints:

$$\|\mathbf{u}\| = 1$$
$$\|\mathbf{r}\| \geq R_e$$

Terminal constraints:

$$t_0^0 = 0;\ t_0^1 = 75.2;\ t_0^2 = 150.4;\ t_0^3 = 261$$
$$t_f^0 = 75.2;\ t_f^1 = 150.4;\ t_f^2 = 261$$

Let $\mathbf{r_f} = [x_0(t_f^3), x_1(t_f^3), x_2(t_f^3)];\ \mathbf{v_f} = [x_3(t_f^3), x_4(t_f^3), x_5(t_f^3)];$

$$a_f(\mathbf{r_f}, \mathbf{v_f}) = 24361140; e_f(\mathbf{r_f}, \mathbf{v_f}) = 0.7308;$$
$$i_f(\mathbf{r_f}, \mathbf{v_f}) = \frac{28.5\pi}{180}; \Omega_f(\mathbf{r_f}, \mathbf{v_f}) = \frac{269.8\pi}{180};$$
$$\omega_f(\mathbf{r_f}, \mathbf{v_f}) = \frac{130.5\pi}{180};$$

Where $a_f, e_f, i_f, \Omega_f, \omega_f$ are computed as follows

$$\mathbf{h} = \mathbf{r_f} \times \mathbf{v_f};\ \mathbf{n} = [0, 0, 1]^T \times \mathbf{v_f};$$
$$\mathbf{e} = \frac{1}{\mu}\mathbf{v_f} \times \mathbf{h} - \frac{\mathbf{r_f}}{\|\mathbf{r_f}\|};$$
$$e_f = \|\mathbf{e}\|;\ a_f = -\frac{\mu}{\|\mathbf{v_f}\|^2 - \frac{2\mu}{\|\mathbf{r_f}\|}};\ i_f = \cos^{-1}\left(\frac{\mathbf{h}.[0, 0, 1]^T}{\|\mathbf{h}\|}\right);$$

if $\mathbf{n}[1]$[5]>0: $\Omega_f = \cos^{-1}\left(\frac{\mathbf{n}.[0, 0, 1]^T}{\|\mathbf{n}\|}\right)$ else: $\Omega_f = (2\pi - \Omega_f);$

if e[2]>0: $\omega_f = \cos^{-1}\left(\frac{\mathbf{n}.\mathbf{e}}{\|\mathbf{n}\|\, e_f}\right)$ else: $\omega_f = (2\pi - \omega_f);$

Event constraints:

$$\mathbf{r}(t_f^p) - \mathbf{r}(t_0^{p+1}) = 0;\ \mathbf{v}(t_f^p) - \mathbf{v}(t_0^{p+1}) = 0; \quad p = 0, 1, 2$$
$$m(t_f^p) - m(t_0^{p+1}) = [13680, 6840, 8830] \quad p = 0, 1, 2$$

Initial conditions:

$$\mathbf{r}(t_0^0) = [5605222.973, 0, 3043387.761];\ \mathbf{v}(t_0^0) = [0, 408.74, 0];$$
$$m(t_0^0) = 301454$$

6 Adaptive mesh refinement schemes

Global collocation works efficiently when the solution of OCP is smooth. However, global collocation often fails when the optimal solution involves discontinuities in the state, control, or their derivatives [23]. While direct collocation using multiple segments often results in a better approximation, the selection of the number of segments, the width of each segment, and the approximating polynomials is a complex exercise. Adaptive mesh refinement schemes that vary the number of segments and approximating polynomials are found to produce robust solutions [12].

Various adaptive mesh refinement schemes are proposed in literature [12] [9] [24] [25]. Traditional OCP solver packages implement customized adaptive mesh refinement schemes. For example, GPOPS-II uses the estimate of the residuals to refine the grid in every iteration until a pre-specified tolerance on the residual is achieved. Similarly, PSOPT uses a linear search algorithm to find the best estimates of the number of segments and polynomial order [21]. While the iterative methods are proven to produce robust solutions, they are computationally intensive due to the reformulation of NLP with continuously changing collocation approximations in each iteration. The adaptive algorithms used in various software, change the number of segments and the approximating polynomials in each iteration of the mesh refinement. Changing the number of segments or approximating polynomials not only requires re-computation of collocation approximation but also reformulation of the NLP which is often a computationally expensive exercise compared to NLP solve time.

In contrast, three different mesh refinement techniques are proposed in this section that produce the best possible solution to a given collocation approximation (Fixed number of segments along with predefined approximating polynomial degree) by refining the grid using the segment widths.

6.1 Motivation : Moon-lander OCP

Continuous time OCP formulation of a simple moon lander example that is often used in literature for validating adaptive mesh refinement schemes is given in Equation 59.

$$\begin{aligned}
\min_{x,u} \quad & J = 0 + \int_{t_0}^{t_f} u \, dt && \text{(59a)}\\
\text{subject to} \quad & \dot{x_0} = x_1 && \text{(59b)}\\
& \dot{x_1} = u - 1.5 && \text{(59c)}\\
& x_0 \geq 0 && \text{(59d)}\\
& 0 \leq u \leq 3 && \text{(59e)}\\
& x_0(t_0) = 10;\ x_1(t_0) = -2; t_0 = 0.0; && \text{(59f)}\\
& x_0(t_f) = 0;\ x_1(t_f) = 0; t_f = \text{free variable} && \text{(59g)}
\end{aligned}$$

The optimal solution to OCP given in Equation 59 has a classical bang-bang thrust profile [26]. The problem is easily solved using the solver package with less than 10 lines of effective python code given below.

Comparison of the solution obtained using direct collocation using the polynomials of order 20 and the exact solution given in Figure 5a shows that direct collocation fails to approximate the optimal solution. Further, the problem is solved using multi-segment collocation with 5 segments each having approximating polynomial of order 3. A comparison of the multi-segment solution with the exact solution shown in Figure 5b indicates that an equal width solution requires a large number of collocation nodes to accurately solve the problem. Lastly, the problem is solved using 200 segments each approximated with 3rd order polynomials, the solution is plotted in Figure 5c. A close look at the solution near the discontinuity given in Figure 5d in the thrust profile shows that the non-adaptive collocation approximations fail to capture the optimal thrust profile even with a large number of collocation nodes in addition to being computationally expensive. For instance, collocation using 200 segments is found to take approximately 25 times more time compared to a 5 segment equivalent NLP.

```python
# Moon lander OCP direct collocation
from mpopt import mp

# Define OCP
ocp = mp.OCP(n_states=2, n_controls=1)
ocp.dynamics[0] = lambda x, u, t: [x[1], u[0] - 1.5]
ocp.running_costs[0] = lambda x, u, t: u[0]
ocp.terminal_constraints[0] = lambda xf, tf, x0, t0: [xf[0], xf[1]]
ocp.x00[0] = [10.0, -2.0]
ocp.lbu[0], ocp.ubu[0] = 0, 3

# Create optimizer(mpo), solve and post process(post) the solution
mpo, post = mp.solve(ocp, n_segments=1, poly_orders=20, scheme="LGR", plot=True)
```

Listing 3: Solving moon lander OCP example (Equation 59) using MPOPT

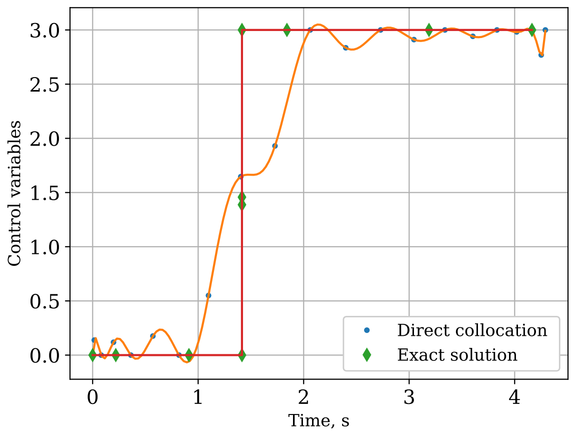
This example shows that the default-choice of segment widths (Equal width segments) and approximating polynomials chosen by the user, affect the quality of the solution. In order to get robust solutions, adaptive grid refinement techniques are generally employed.

In the present package, three different adaptive mesh refinement schemes are implemented. The collocation approximation remains the same as long the number of segments and approximating polynomials are unchanged. Further, the absolute grid can be made either finer or sparser by changing the segment widths without affecting the approximation. In all the three schemes presented in this section, only the segment widths are varied to get the best possible solution for a given approximation. Such a mesh refinement philosophy is often computationally attractive in addition to being deterministic in computational time. Since the NLP does not change the dimension in each subsequent iteration, the computational time is further accelerated by employing initial guess from the previous iterations while solving the NLP.

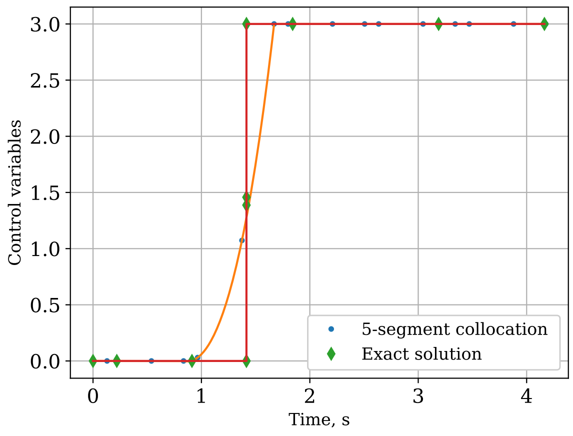
Of the three proposed schemes, the first two formulations are iterative in nature while the third formulation solves for the optimal segment width fractions along with the original NLP variables. While all three schemes are found to be successful against standard test cases, each fit best for a different class of problems as discussed in this section.

6.2 Algorithm for Iterative Adaptive mesh refinement schemes

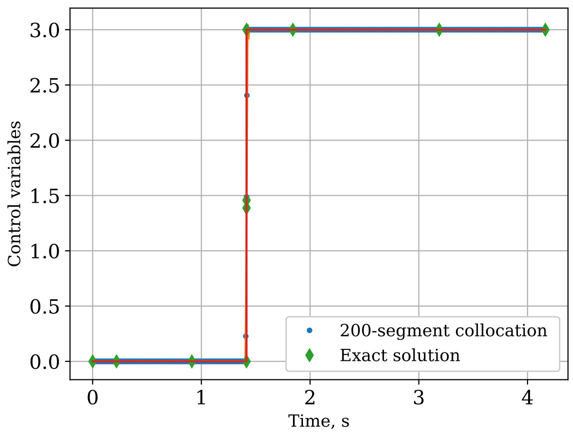
Consider a collocation approximation of given multi-phase continuous time OCP with N_s^p number of segments in each phase p, with each segment having approximating polynomials of order d_s^p. Further, let $\mathcal{Z}^0 = \{\mathbf{X}^i, \mathbf{U}^i, t_0^i, t_f^i, a^i;\ i = 1, \dots, p\}$ represent the solution to the OCP where all segments have equal width. Now, an outline of the iterative adaptive mesh refinement schemes proposed in this package is given in Algorithm 3.



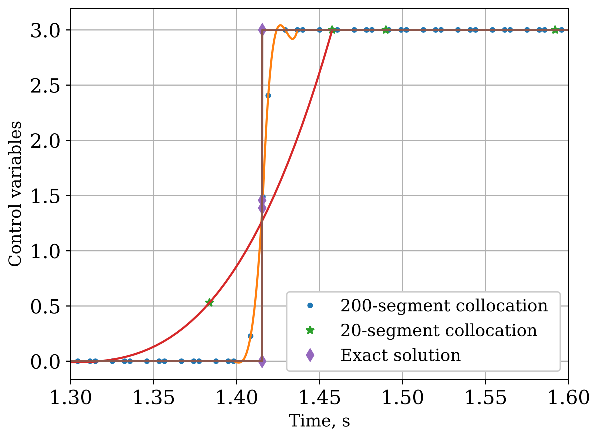
(a) Comparison of **global** collocation solution with exact thrust profile for moon lander example



(b) Comparison of **multiple segment** collocation solution with exact thrust profile for moon lander example (5 segments, polynomial degree 3)



(c) Comparison of **multiple segment** collocation solution with exact thrust profile for moon lander example(200 segments, polynomial degree 3)



(d) Comparison of **multiple segment** collocation solution with exact thrust profile for moon lander example

Figure 5: Comparison of moon lander thrust profile using global collocation and multi-segment collocation (non adaptive) with exact solution

Algorithm 3 Iterative segment width refinement: Adaptive solver

```
1:  procedure H-ADAPTIVE($N_p, N_s^p, d_s^p$)
2:      NLP($N_p, N_s^p, d_s^p$) $\leftarrow$ Initialize NLP                                  $\triangleright$ Algorithm (1)
3:      $h_s^p \leftarrow \dfrac{1}{N_s^p}$                  $\triangleright$ Segment width fraction set to equal width segments
4:      $\mathcal{Z}^0 = \{\mathbf{X}^i, \mathbf{U}^i, t_0^i, t_f^i, a^i;\ i = 1, \ldots, p\} \leftarrow$ Solve NLP with $h_s^p$        $\triangleright$ Reference NLP solution
5:      for Each iteration $k$ in $[1, \ldots, Iter_{max}]$ do
6:          $\mathbf{R} \leftarrow$ Estimate of max. residual in $\mathcal{Z}^{(k-1)}$        $\triangleright$ At non-collocation nodes (Equation 43)
7:          if $(\mathbf{R} < \epsilon^r)$ or $(\dfrac{dh_s^p}{dk} \leq \epsilon^h)$ then
8:              break;
9:          else
10:             $h_s^p \leftarrow$ Refine segment widths using solution $\mathcal{Z}^{(k-1)}$ $\triangleright$ Adaptive schemes (6.2.1, 6.2.2)
11:             $\mathcal{Z}^k = \{\mathbf{X}^i, \mathbf{U}^i, t_0^i, t_f^i, a^i;\ i = 1, \ldots, p\} \leftarrow$ Solve NLP with $h_s^p$      $\triangleright$ Refined NLP solution
12:     return $\mathcal{Z}$                                                  $\triangleright$ Return updated solution
```

6.2.1 Adaptive scheme-I: Heuristic

In this heuristic scheme, we consider the class of problems where a measure of required collocation nodes density at any given time can be defined in terms of the optimization variables themselves. For example, consider a system where a rapidly changing control signal requires the refined grid to capture the solution accurately. In such cases, the slope of the control signal is a good measure of the collocation nodes density. Moreover, it can be seen that increasing the density of collocation nodes close to the thrust discontinuity in moon lander example given in subsection 6.1 results in refined solution. Various possible criteria are listed in Table 2. While different measures can be used to represent the collocation nodes density, slope of control variables or state variables is often found to be a good measure. In collocation, the slope of control, or state variables is easily computed using the projection operation equation given in Equation 37 using pre-computed interpolation matrices. The algorithm for the mesh refinement using the proposed heuristic scheme is presented in this section.

Let $\mathcal{Z}^k = \{\mathbf{X}^i, \mathbf{U}^i, t_0^i, t_f^i, a^i;\ i = 1, \ldots, p\}$ represent the solution to the OCP at iteration k. Let $\mathcal{T}^{\text{grid}} = \left\{t_i^p \in (t_0^p, t_f^p); i = 1, \ldots, N_g^p\right\}$ represent the set of predefined points between start and final times of each phase excluding the terminal points. Let, $\mathcal{C}$ represent a criteria for detecting the required collocation node density. Further, let $\mathcal{C}(t_i)$ represent the measure of required collocation nodes density at time t_i. Now, we estimate a set of points using the defined criteria ($\mathcal{C}$), and store them in a pivot set represented by $\mathcal{T}^{\text{pivot}}$, defined as follows.

Define $\mathcal{T}^{\text{pivot}} = \left\{t_i^p \in \mathcal{T}^{\text{grid}} \mid \mathcal{C}(t_i^p) > \epsilon;\ \mathcal{C}(t_i^p) > \mathcal{C}(t_j^p)\ \forall\, i > j\right\}$ as the set of ordered points according to a given criteria ($\mathcal{C}$) and threshold (ϵ^c).

Let $\mathcal{T}_k^{\text{pivot}}$ represent the set of pivot locations estimated in iteration k using solution $\mathcal{Z}^k$. Then, two different scenarios exist. The first scenario is that the number of pivot points in ($\mathcal{T}_k^{\text{pivot}}$) are more than the number of points required to uniquely define segment start and end points (i.e $N_s^p - 2$). In this case, we define the first $N_s^p - 1$ points from $\mathcal{T}^{\text{pivot}}$ in addition to phase starting and ending times (t_0^p, t_f^p) as the refined segment start or end points. These $N_s^p + 1$ points uniquely define new segments. Therefore, segment width fractions for N_s^p segments are refined and the problem can be solved with refined segment widths. The procedure is repeated iteratively until a pre-specified convergence criterion is reached. Now, the algorithm for the proposed scheme can be outlined as in Algorithm 4.

Sl. no	Criteria ($\mathcal{C}$)	Remarks
1	$\|\|\dot{\mathbf{u}}\|\|$	Norm of slope of control variables (1-norm, 2-norm, ∞-norm)
2	$\dot{u}_i;\ i = 1, \ldots, n_u$	Slope of any control variable of choice (i)
3	$\|\|\ddot{\mathbf{u}}\|\|$	Norm of second derivative of control variables
4	$\|\|\dot{\mathbf{x}}\|\|$	Norm of slope of state variables
5	$\dot{x}_i;\ i = 1, \ldots, n_x$	Slope of any state variable of choice (i)
6	$\|\|\ddot{\mathbf{x}}\|\|$	Norm of second derivative of state variables
7	$f(\mathbf{x}, \mathbf{u}) \in \mathbb{R}$	Custom criteria

Table 2: Criteria for deciding pivot locations

Algorithm 4 Iterative scheme for segment width refinement: Adaptive scheme-I

```
1:  procedure H-ADAPTIVE-I($\mathcal{Z}^k, h_s^p, \mathcal{T}^{\text{grid}}$)
2:      $\mathcal{C} \leftarrow$ Select criteria for measure of discontinuity    ▷ From Table 2
3:      $\mathcal{T}^{\text{pivot}} = \{t_i^p \in \mathcal{T}^{\text{grid}};\ \mathcal{C}(t_i^p) > \epsilon;\ \mathcal{C}(t_i^p) > \mathcal{C}(t_j^p)\ \forall\ i > j\}$    ▷ Compute pivot locations in $\mathcal{Z}^k$
4:      if No. of pivots $> N_s^p - 2$ then    ▷ Enough pivots to define new segments
5:          $\mathcal{T}^{\text{pivot}} \leftarrow$ Select first $N_s^p - 1$ pivots from $\mathcal{T}^{\text{pivot}}$
6:          $\mathcal{T}^{\text{pivot}} \leftarrow [t_0^p, t_f^p]$    ▷ Add start and final times to pivots
7:          $\mathcal{T}^{\text{pivot}} \leftarrow$ Sort pivots in ascending order
8:      else    ▷ Not enough pivots to define new segments
9:          $\mathcal{T}^{\text{pivot}} \leftarrow [t_0^p, t_f^p]$    ▷ Add start and final times to pivots
10:         $\mathcal{T}^{\text{pivot}} \leftarrow$ Sort pivots in ascending order
11:         $N \leftarrow$ Number of pivots in $\mathcal{T}^{\text{pivot}}$
12:         $\mathcal{T}^{\text{pivot}} \leftarrow$Split first segment in $\mathcal{T}^{\text{pivot}}$ into $\left\lfloor \frac{N_s^p - N}{2} \right\rfloor$ of equal segments
13:         $\mathcal{T}^{\text{pivot}} \leftarrow$ Split last segment in $\mathcal{T}^{\text{pivot}}$ into remaining number of equal segments
14:     $h_s^p \leftarrow$ Estimate segment width fractions from $\mathcal{T}^{\text{pivot}}$    ▷ Refine segment width fractions
15:     return $h_s^p$    ▷ Return refined segment width fractions
```

While the heuristic adaptive mesh refinement scheme presented in this section refines the segment widths, selection of number of segments themselves is often a difficult choice. Although, selecting more number of segments is always preferred with the proposed scheme, it is often computationally costly. However, the proposed schemes is found to be robust with respect to the choice of segments and polynomials when initialized with more than minimum required segments to approximate the solution profile. That is to say, depending on the nature of solution and number of discontinuities expected in the solution profile, it is often possible to know the minimum number of segments required for a given OCP. For example, moon-lander OCP has bang-bang thrust profile with one discontinuity. Hence, at least 3 segments are required to approximate the control exactly.

Descriptive examples

In literature, two specific OCPs are widely used for validating the adaptive mesh refinement schemes. First one being the moon-lander OCP given in Equation 59 representing discontinuity in control signal, second one is the hyper-sensitive OCP given in Equation 62.

Example-1: Moon lander OCP

The moon lander OCP with 2 states and 1 control variable is given in Equation 59. The states and control trajectories obtained using the proposed adaptive scheme are plotted in Figure 6b. The grid refinement history corresponding to the solution in Figure 6b is plotted in Figure 6c. The grid is iteratively refined until a user defined tolerance (10^{-6}) in residual is achieved. The refined grid places more collocation nodes close to the discontinuity resulting in an accurate solution. Comparison of the non-adaptive solution and adaptive solution using the proposed adaptive scheme and the exact solution is given in Figure 6d for the same collocation approximation. The free final times obtained using the non-adaptive scheme and the adaptive scheme are (4.1762, 4.1641) respectively. The value obtained using proposed matches exactly with the analytical solution (4.1641). The proposed scheme not only produces a better solution compared to the non-adaptive solution, it is also computationally attractive. The adaptive scheme produces a very close approximation to the exact solution with as low as 15 collocation nodes compared to 600 nodes in the non-adaptive solution. The adaptive scheme converges within 5 iterations to the user-defined tolerance (10^{-6}).

Further, the robustness of the method is verified by varying the number of segments and approximating polynomials in each segment. The residual in dynamics estimated at non-collocation points serves as a good measure of the accuracy of the solution. Further, the residual is estimated on a fixed grid over the time horizon in order to be able to compare various solutions. The evolution of maximum residual in dynamics at non-collocation nodes with respect to number of mesh iterations is shown in Figure 7 for polynomials of order 3. The residual converges to a specified tolerance of 10^{-6} within less than 10 iterations in all the cases. Further, an acceptable tolerance of 10^{-3} is generally achieved with in 3 iterations. Evolution of the maximum residual with respect to the approximating polynomials of degree 4, 5, 6 is shown in Figure 7.

The state and control trajectories to hypersensitive OCP and Van der Pol oscillator OCP using the adaptive scheme are plotted in Figure 8.

Remarks

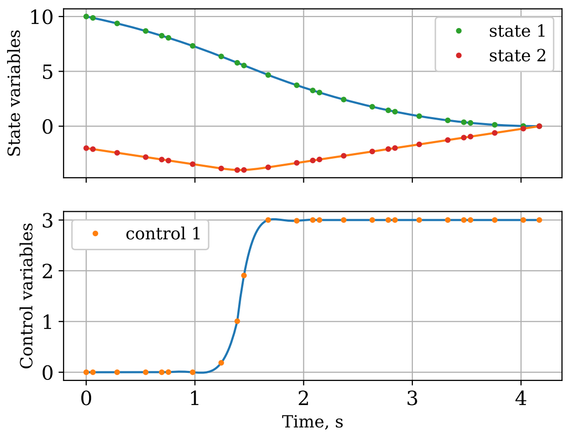
While the proposed adaptive scheme succeeds in obtaining the robust solutions in a select class of problems where the measure of collocation nodes density can be defined the convergence of the methods is not guaranteed. Another drawback of the proposed method is that the need to select criteria for a measure of grid density. While slope of the control signal is found to be a good measure for moon-lander OCP (Equation 59), Van der Pol OCP (Equation 56) and hypersensitive OCP (Equation 62), it is not guaranteed to work for other OCPs.

6.2.2 Adaptive scheme-II: Residuals based

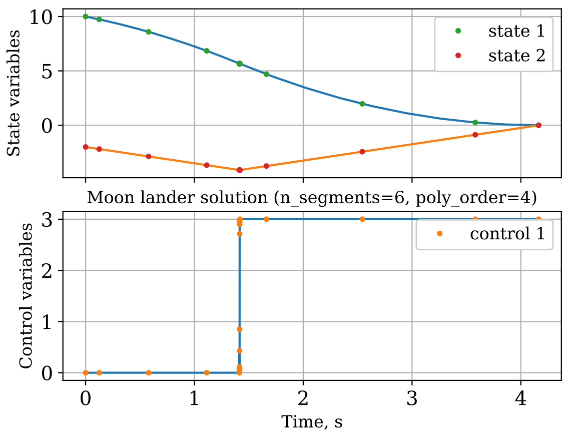
The second iterative adaptive mesh refinement proposed in the package is based on the residuals in system dynamics. Since residuals at non-collocation nodes is a good measure of the accuracy of the solution, this scheme detects the location of maximum residual in a given solution and refines the grid such that more collocation points are concentrated where residual is higher.

Let $\mathcal{Z}^0 = \{\mathbf{X}^i, \mathbf{U}^i, t_0^i, t_f^i, a^i;\ i = 1, \ldots, p\}$ represent the non-adaptive solution of given multi-phase OCP. Then, the residual in the dynamics corresponding to given solution ($\mathcal{Z}^0$) is easily estimated using the composite derivative interpolation matrix defined in Equation 40. Let $\mathbf{R}_s^p$ represent the residual in system dynamics in segment s of phase p computed using Equation 37.

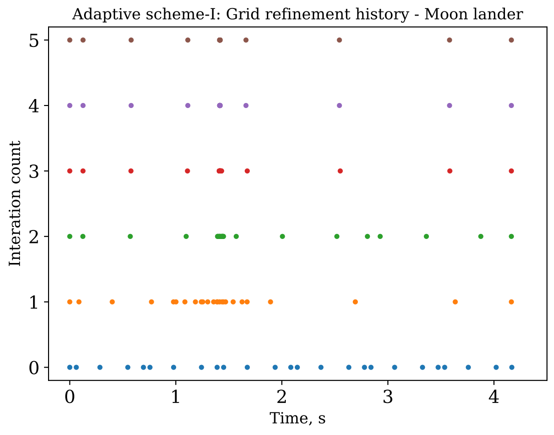
The package implements two different grid refinement methods based on the estimated residual ($\mathbf{R}_s^p$). While the first method ensures equal integrated residual in each segment, the second method



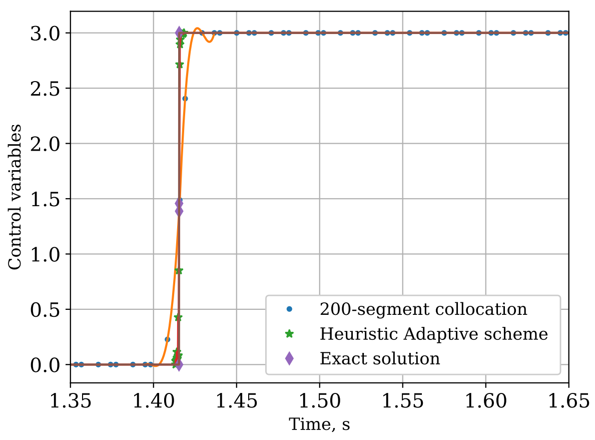
(a) States and control trajectories using equal width segments: Moon lander



(b) States and control trajectories using Adaptive mesh refinement scheme-I: Moon lander



(c) Grid refinement history for the adaptive mesh refinement scheme I based on control slope criteria. Note that the number of collocation nodes remain same throughout the iterations



(d) Comparison of optimal thrust profile using non-adaptive, Heuristic adaptive mesh refinement scheme and exact solution close to the discontinuity

Figure 6: Heuristic adaptive grid refinement strategy applied to moon lander OCP

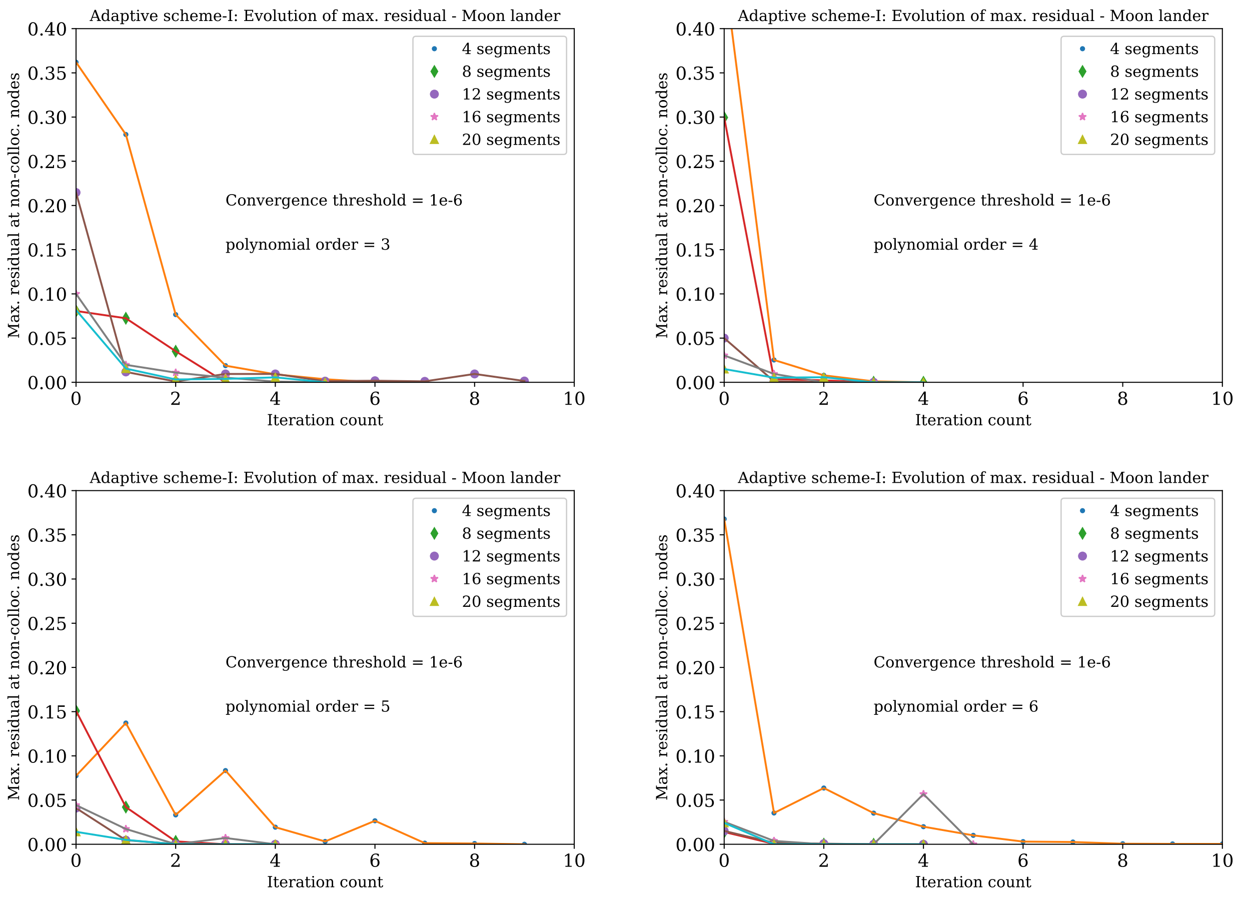
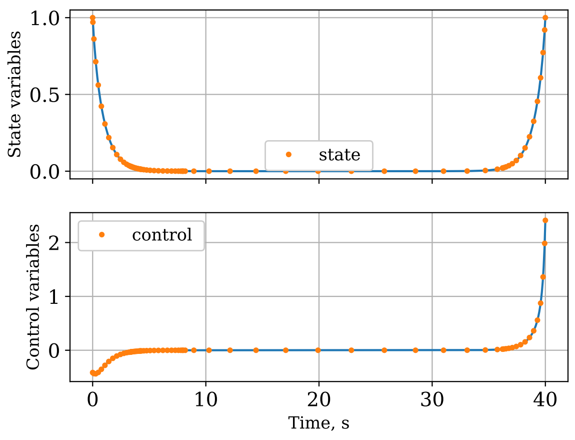
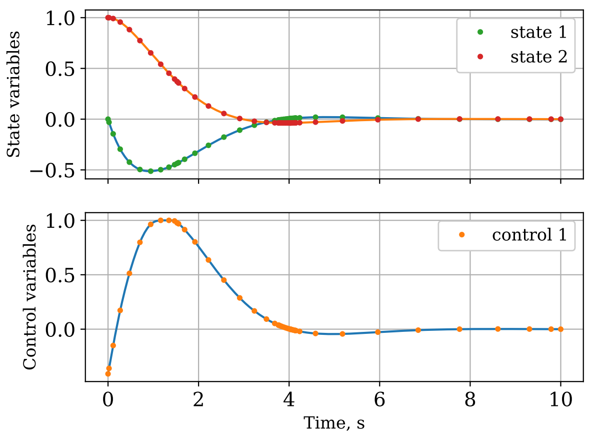


Figure 7: Evolution of maximum residual for approximating polynomials of orders 3, 4, 5, 6



(a) State and control trajectories of hypersensitive ocp (Equation 62) solved using Heuristic adaptive scheme



(b) State and control trajectories of hypersensitive ocp (Equation 56) solved using Heuristic adaptive scheme

Figure 8: Optimal solutions for Heuristic adaptive scheme-I test cases

refines the grid by merging the segments where residual is low and splitting the segments where residual is high.

Method-1: Equal residual in each segment

Let $\mathcal{T}^{\text{grid}} = \{t_i^p \in [t_0^p, t_f^p]; i = 1, \dots, N_g^p\}$ represent the set of predefined points between start and final times of each phase where the residual is to be estimated. Let R_i^p be the 2-norm of residual at time $t_i^p \in \mathcal{T}^{\text{grid}}$. Further, let $R(t)$ represent a residual curve over the domain $[t_0^p, t_f^p]$ obtained using interpolation of estimated residuals R_i^p. Dividing the residual curve into N_s^p number of equal area segments results in improved collocation nodes density where there is high residual.

Let $t_0^s, t_f^s;\ s = 1, \dots, N_s^p$ represent the start and end points of refined mesh segment s. Then, the refined segment start and end points are uniquely defined by Equation 60 and Equation 61.

$$\int_{t_0^s}^{t_f^s} R(t)\, dt = \int_{t_0^{s+1}}^{t_f^{s+1}} R(t)\, dt; \qquad s \in [1, \dots, N_s^p - 1] \tag{60}$$

$$t_f^s = t_0^{s+1} \tag{61}$$

Descriptive example

hypersensitive problem given in Equation 62 is often considered a standard test case to validate adaptive mesh refinement schemes.

$$\min_{x,u} \qquad J = 0 + \frac{1}{2}\int_{t_0}^{t_f} (x^2 + u^2)\, dt \tag{62a}$$

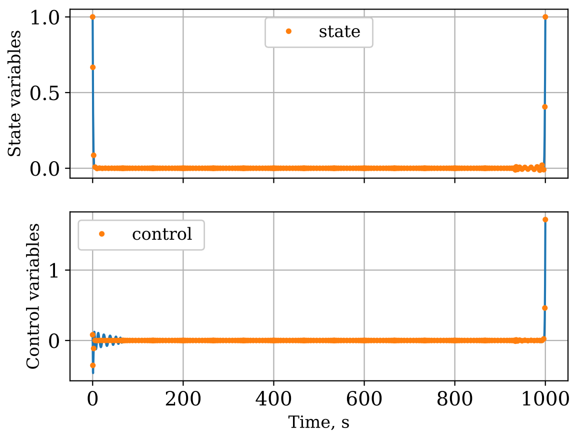
$$\text{subject to} \qquad \dot{x} = -x^3 + u \tag{62b}$$

$$x_0(t_0) = 1; t_0 = 0; \tag{62c}$$

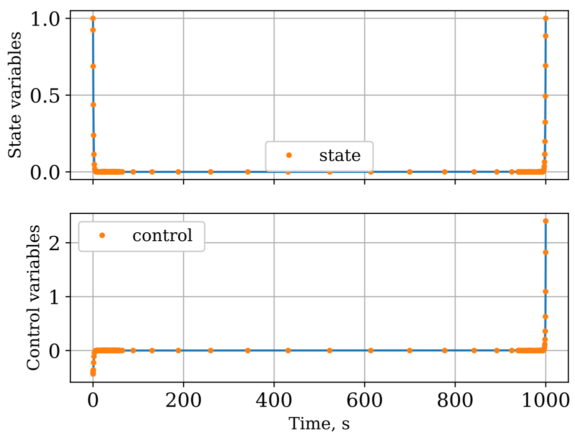
$$x_1(t_f) = 0; t_f = 1000 \tag{62d}$$

The hypersensitive problem is easily programmed in the package within less than 10 lines of effective python code given below. The problem is first solved using non-adaptive scheme and the state and control trajectories for non-adaptive collocation approximation are plotted in Figure 9a. The evolution of residual with respect to the adaptive refinement iterations is plotted in Figure 11b. The exact solution requires a higher number of collocation nodes near the start as well as terminal times. The evolution of the grid with respect to each iteration of the proposed equal residual adaptive grid refinement scheme is given in Figure 9c. Note the change in collocation nodes distribution from non-adaptive solution (Iteration 0) to the first iteration. The method detects higher residual towards the terminal times in iteration 0 and places more segments towards the terminal times in the first iteration, refining the solution. The process is repeated until a predefined convergence tolerance is achieved, the evolution of the grid is shown in Figure 9c. The states and control trajectories from the converged solution are plotted in Figure 9b. Equal residual segments proposed in this section refine the grid within one iteration and place the collocation nodes towards the corners as shown in Figure 9c. The comparison of the non-adaptive solution with the adaptive solution given in Figure 9d shows the higher density of nodes close to the starting time makes the solution converge to the exact solution.

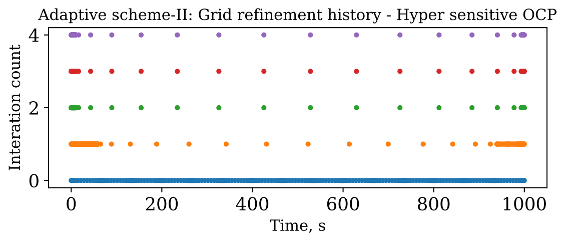
The complexity of the problem is increased when the terminal time (t_f) is increased. The states and control trajectories obtained for various terminal times are plotted in indicating the robustness of the method.



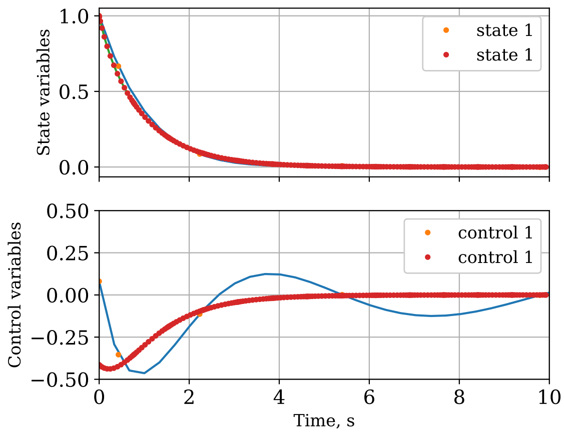
(a) Multiple segment solution of hypersensitive problem with equal width segments



(b) Multiple segment solution of hypersensitive problem with adaptive mesh refinement : Each segment has equal integrated residual

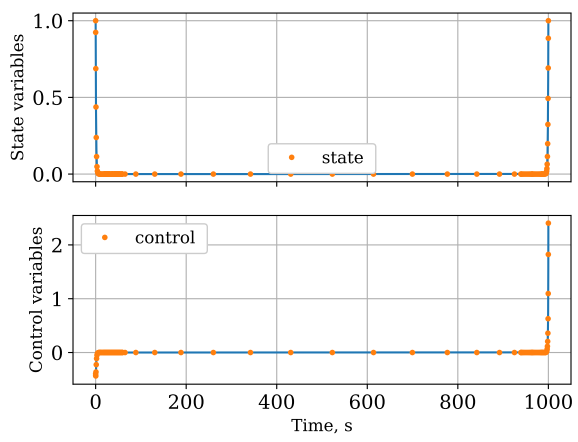


(c) Multiple segment solution of hypersensitive problem with adaptive mesh refinement : Grid refinement history

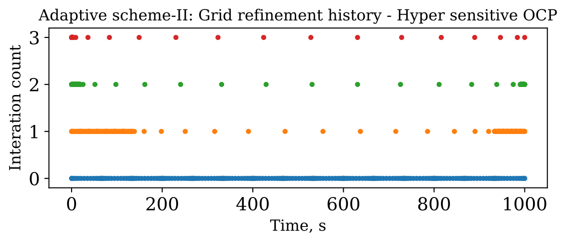


(d) Comparison of adaptive solution based on equal residual segments with non-adaptive solution

Figure 9: hypersensitive problem solved using iterative adaptive grid refinement scheme based on equal residual segments philosophy (15 segments, polynomial degree 15)



(a) State and control trajectories for hypersensitive problem obtained with adaptive mesh refinement scheme using merge split philosophy



(b) Evolution of grid refinement for hypersensitive problem obtained with adaptive mesh refinement scheme using merge split philosophy

Figure 10: hypersensitive problem solved using iterative adaptive grid refinement scheme based on residuals and merge/split philosophy (15 segments, polynomial degree 15)

Method-2: Merge or split segments

Let maximum residual in segment s be defined by $R_s^p := \|\mathbf{R}_s^p\|_\infty$. Consider two consecutive segments that have an acceptable maximum residual. In addition, if merging these two segments also keeps the maximum residual in the acceptable level, the free segment that resulted from the merge operation can be used to refine the grid elsewhere. Note that the number of segments is kept the same throughout the grid refinement.

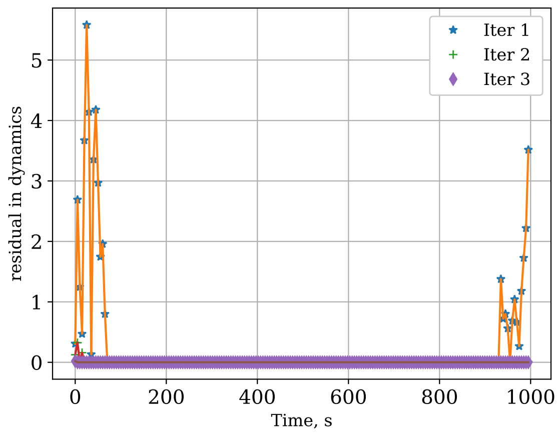
Let $\mathcal{S}^p = 1, \ldots, N_s^p$ represent the set of segment indices in phase p.

Let $\mathcal{R}^{\text{err}} := \{i \mid R_i^p > \epsilon^r;\ i \in \mathcal{S}^p\}$ be a set of segments which have maximum residual above given threshold (ϵ^r).

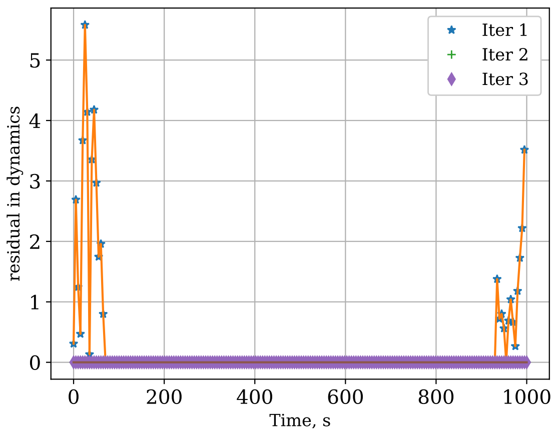
Let $\mathcal{R}^{\text{groups}} := \left\{i, i+1 \mid (R_i^p, R_{i+1}^p) < \epsilon^r; i \in \mathcal{S}^p\right\} \cup \mathcal{R}^{\text{err}}$ represent disjoint sets of consecutive segments grouped by residual (ϵ^r). Further, let the disjoint sets in $\mathcal{R}^{\text{groups}}$ be ordered in ascending order of their indices. Let N^e, N^g represent the number of sets in $\mathcal{R}^{\text{err}}, \mathcal{R}^{\text{groups}}$ respectively. Define $\mathcal{H}^p := \left\{\sum_{i \in S} h_i^p; \text{For Each } S \in \mathcal{R}^{\text{groups}}\right\}$. Number of free segments(N^f) are defined as $N^g - N^e$. If the number of free segments are 0, the grid is not refined further. Otherwise, the segment width fractions are refined by merging each group in $\mathcal{R}^{\text{groups}}$ if the group has acceptable tolerance and splitting the groups that have residual higher than the tolerance while maintaining total number of segments same.

Descriptive examples

The hypersensitive example given in Equation 62 is solved using the merge/split adaptive strategy proposed in this section. The evolution of grid refinement with respect to iterations using *merge split* philosophy is plotted in Figure 10b. The evolution of residual on a fixed grid with respect to the iterations are plotted in Figure 11a. The segments close to the start and end times are split due to their higher maximum residual and the segments in the middle are merged in each iteration. The adaptive solution using merge split philosophy is given in Figure 10a. The problem is solved to an acceptable level (10^{-3}) using the proposed adaptive scheme within 3 iterations.



(a) Adaptive mesh refinement scheme using merge split philosophy



(b) Adaptive mesh refinement scheme using equal residual segments

Figure 11: hypersensitive problem solved using iterative adaptive grid refinement schemes: Evolution of residual with respect to iterations (15 segments, polynomial degree 15)

Remarks

While the proposed adaptive mesh refinement schemes based on residual work well on the test cases, the convergence of the method is not guaranteed. One known drawback of the method is that the algorithm doesn't converge when initialized with fewer segments than the minimum required number.

6.3 Adaptive scheme-III: Direct optimization

The third adaptive mesh refinement scheme implemented in the package is a unique one. While all the adaptive refinement schemes encountered in literature are iterative in nature, this method attempts to optimize the segment width fractions along with the original NLP.

The adaptive direct optimization scheme proposed in this section modifies given NLP by including segment width fractions (h_s^p) defined in Equation 9 as additional optimization variables along with additional constraints on residual in system dynamics (Equation 63). Let R_s^p be the residual matrix in segment s. Then, the constraints given in Equation 63 are added to the original NLP in the proposed adaptive scheme.

$$\sum_{i=1}^{N_s^p} h_i^p = 1 \qquad \forall p = 1, \ldots, N_p \tag{63a}$$

$$-\epsilon^{tol} \leq h_s^p \times R_s^p \leq \epsilon^{tol} \qquad \epsilon^{tol} > 0; \qquad \forall s = 1, \ldots, N_s^p;\ p = 1, \ldots, N_p \tag{63b}$$

$$\epsilon^h < h_s^p \leq 1; \qquad \epsilon^h > 0; \qquad \forall s = 1, \ldots, N_s^p;\ p = 1, \ldots, N_p \tag{63c}$$

Intuitively, the constraint in Equation 63b makes the segment width fraction lower where ever the residual in dynamics is higher. Therefore increasing the density of the collocation nodes where ever the residual is higher which in turn refines the solution.

The modified NLP in the proposed adaptive direct optimization scheme is given by Equation 64.

$$\min_{\mathcal{X},\mathcal{U},\mathcal{T},\mathcal{A},\mathcal{P},\mathcal{H}} \quad \sum_{p=1}^{N_p} \left[M^p + \mathbf{W}_p^T \mathbf{Q}^p \right] \tag{64a}$$

$$\text{s. t.} \quad 0 = \mathbf{D}^p\mathbf{X}^p - \mathbf{F}^p = 0 \qquad \forall p = 1,\ldots,N_p \tag{64b}$$

$$-\infty = \mathbf{G}^p \leq 0 \qquad \forall p = 1,\ldots,N_p \tag{64c}$$

$$0 = \mathbf{H}^p = 0 \qquad \forall p = 1,\ldots,N_p \tag{64d}$$

$$\underline{\mathbf{E}}^l = \mathbf{E}^l = \overline{\mathbf{E}}^l \qquad \forall\, l \in \mathcal{P} \tag{64e}$$

$$-\epsilon^{tol} = \mathbf{R}^p = \epsilon^{tol} \qquad \forall p = 1,\ldots,N_p \tag{64f}$$

$$\mathbf{X}^p \in \mathcal{X}^p; \quad \mathbf{U}^p \in \mathcal{U}^p; \qquad \forall p = 1,\ldots,N_p \tag{64g}$$

$$t_0^p \in \mathcal{T}_0^p; \quad t_{N_c^p}^p \in \mathcal{T}_f^p; \quad \mathbf{a}^p \in \mathcal{A}^p; \qquad \forall p = 1,\ldots,N_p \tag{64h}$$

Where all matrices except R^p are constructed in the same way as in non-adaptive solver describe in Section 3.2. The matrix R_p is constructed as follows,

Let $\mathcal{T}_I^s = \{\tau_i^I; i = 1,\ldots,I_s^p;\ \tau_i^I \in [-1,1]\}$ be I_s^p number of predefined points where the value of the residual is to be constrained (Equation 63b) in the segment s and phase p. Further, let composite interpolation ($\mathbf{C}_I^s$) and derivative interpolation matrices ($\mathbf{D}_I^s$) corresponding to $\mathcal{T}_I^s$ be computed using the individual segment interpolation matrices (Equation 39, Equation 40) as described in Section 3.2. Then, the residual matrix is easily computed using Equation 43 which is then multiplied by the respective segment width fractions.

Selection of nodes for residual estimation

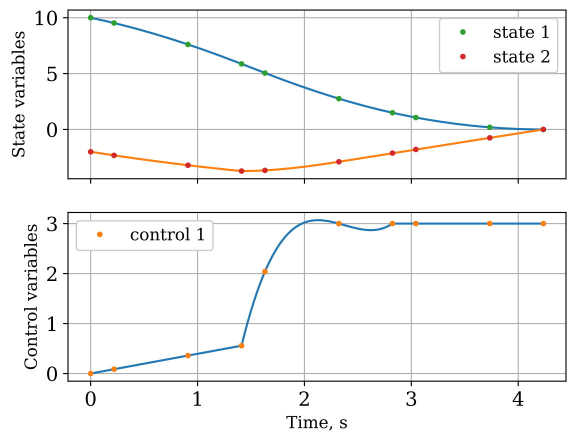
The non-collocation nodes where the residual is constrained in the adaptive direct optimization problem ($\mathcal{T}_I^s$) are chosen such that they result in higher residual where ever more collocation nodes are required. There are two popular methods to select $\mathcal{T}_I^s$. While first method uses mid points of the consecutive collocation nodes in a given segment, the second method uses JACOBI polynomials of different order from the selected approximating polynomial for the segment. i.e

$$\tau_i^I = \frac{\tau_i^s + \tau_{i+1}^s}{2}; \qquad \forall\, s = 1,\ldots,N_s^p - 1 \qquad \textbf{Method-1} \tag{65}$$

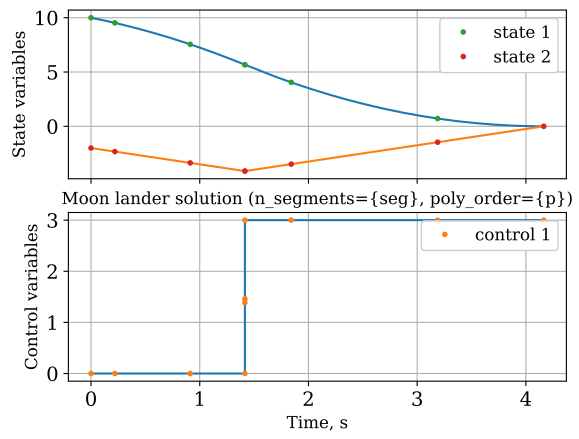
$$\tau_i^I = \text{roots}(d_i); \qquad \forall\, s = 1,\ldots,N_s^p; d_i \neq d_s^p \qquad \textbf{Method-2} \tag{66}$$

While the methods given above are popular choices, other methods of selecting the nodes exist such as uniformly spaced nodes etc.

Outline of the solution procedure in the package for adaptive direct optimization is outlined in Algorithm 5.



(a) Multiple segment solution of moon lander OCP with equal width segments



(b) Multiple segment solution of moon lander OCP with adaptive direct optimization scheme

Figure 12: Moon lander OCP solved using adaptive direct optimization scheme

Algorithm 5 MPOPT adaptive algorithm

```
1: procedure MPOPT-ADAPTIVE(OCP, $N_s^p$, $d_s^p$, collocation_scheme)
2:     $(\mathbf{F}, \mathbf{LB}, \mathbf{UB}, J, \mathbf{Z}, \underline{\mathbf{Z}}, \overline{\mathbf{Z}}) \leftarrow$ Initialize NLP matrices        ▷ Create NLP Algorithm (1)
3:     $(\mathbf{F}, \mathbf{LB}, \mathbf{UB}) \leftarrow$ Update with constraints on residuals        ▷ Equation 64f
4:     $(\mathbf{F}, \mathbf{LB}, \mathbf{UB}) \leftarrow$ Update with constraints on segment widths        ▷ Equations 63a, 63c
5:     $\mathbf{Z} \leftarrow h_s^p; = 1, \ldots, N_p$        ▷ Add segment width to the optimization variables
6:     $Z_0 \leftarrow$ Create initial solution estimate        ▷ Initial solution
7:     $\mathcal{Z} \leftarrow$ Solve NLP($(\mathbf{F}, \mathbf{LB}, \mathbf{UB}, J, \mathbf{Z}, \underline{\mathbf{Z}}, \overline{\mathbf{Z}})$, $Z_0$)        ▷ Solve NLP
8:     $\{\mathbf{X}^i, \mathbf{U}^i, t_0^i, t_f^i, a^i, h_s^i;\ i = 1, \ldots, N_p\} \leftarrow$ Retrieve OCP solution from the NLP solution $\mathcal{Z}$
9:     return $\{\mathbf{X}^i, \mathbf{U}^i, t_0^i, t_f^i, a^i, h_s^i\}$        ▷ Return OCP solution
```

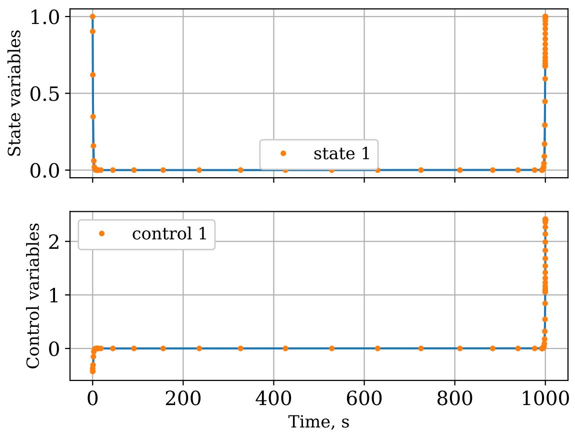
Examples

The proposed adaptive direct optimization method produces an exact solution to the OCP with the minimal number of collocation nodes. For example, the bang-bang thrust profile in the moon-lander example requires exactly 3 segments to approximate the control variable. The solution using equal with segments is plotted in Figure 6a. The states and control trajectories obtained using the direct adaptive scheme are plotted in Figure 12b. The collocation approximation uses 3 segments with approximation polynomials of order 3. The terminal times obtained using the scheme and the exact values are (4.1641, 4.1641) respectively. Further, the optimal switch over time found using this scheme is 1.41535 matches with the exact value from the analytical solution (1.4154).

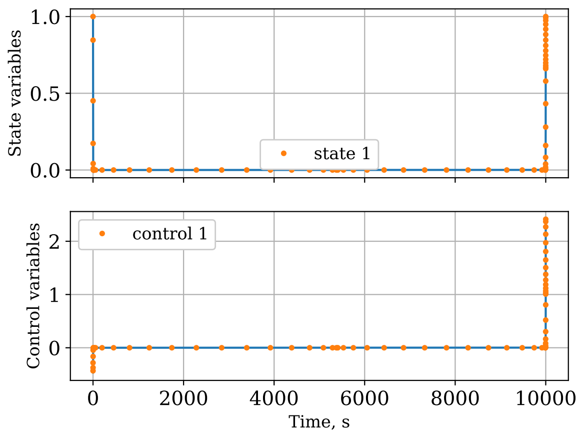
The method is found to be robust and successful in solving various popular adaptive test cases from the literature. For example, the hypersensitive problem is found to fail with the non-adaptive solvers as the terminal time in the OCP formulation is increased (Equation 62). However, the direct optimization adaptive scheme proposed in this section produces accurate solutions with a minimum number of segments as shown in Figure 13.

Remarks

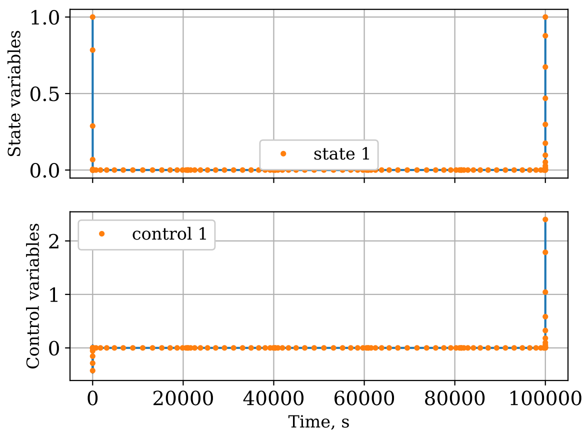
The direct optimization scheme is found to be successful in solving for optimal segment widths given a collocation approximation. One major drawback of the method is found to be computational



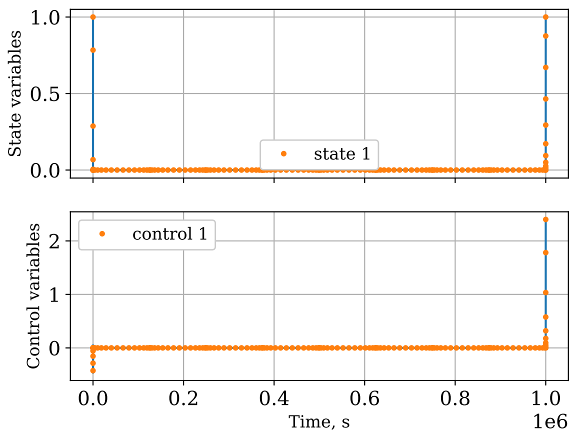
(a) Multiple segment solution of hypersensitive problem (Tf = 1000) using adaptive direct optimization scheme (5 segments)



(b) Multiple segment solution of hypersensitive problem (Tf = **10000**) using adaptive direct optimization scheme(5 segments)



(c) Multiple segment solution of hypersensitive problem (Tf = **100000**) using adaptive direct optimization scheme(7 segments)



(d) Multiple segment solution of hypersensitive problem (Tf = **1000000**) using adaptive direct optimization scheme(10 segments)

Figure 13: hypersensitive problem solved using adaptive direct optimization scheme

time. For example, solve time for the hypersensitive OCP takes 3 orders of magnitude higher time compared to the non-adaptive schemes. The computations are expected to be expensive in the proposed method due to the estimation of residual at the non-collocation nodes. Moreover, the additional constraints posed on the residual and segment widths increase the complexity of the original NLP.

7 Limitations of the package

The package while being successful in solving test cases has limitations like similar software. First of all, the solver requires the dynamics, constraints, and the objectives to be continuous and differentiable explicit functions. For example, the multi-stage launch vehicle ascent trajectory optimization OCP has a terminal constraint on the argument of right ascension that depends on the quadrant of *line of nodes*. Since a conditional on the terminal state variables is not supported by the solver, one must know apriori, the quadrant in which the *line of nodes* lies. Another limitation is the need to initialize the solver with a collocation approximation. While three different grid adaptation techniques are inbuilt in the package, the total number of collocation nodes is still an input by the user.

8 Further improvements

While the core elements of the package are implemented, future work can improve the following aspects of the solver.

Problem definition

The OCP definition can be made more user friendly by allowing definition using a custom signature for the dynamics and constraints. The package in its present form uses a predefined signature.

Initial guess

The initial guess estimation can be improved from the present implementation. While the present implementation uses a linear interpolation of initial estimates for states and controls defined in the OCP, advanced initialization strategies can be implemented in the future to improve convergence. Likewise, the initial solution for the adaptive grid refinement technique can be improved by interpolating the latest available solution onto the refined grid before solving the NLP.

Collocation approximation

The package uses NumPy numerical modules for the derivative and quadrature approximation. The numerical approximation is known to produce noise in derivative estimation for polynomials of order more than 20. The derivative approximation can be improved by implementing the exact analytical formulations for Legendre or Chebyshev polynomials. The software design allows for the custom definition of derivative and integration matrices.

The package at present only supports collocation points that include either one or both terminal points of the standard domain ([-1, 1]). The compatibility for Lagrange-Gauss roots that do not include both ends of the domain can be implemented in the future.

Additional constraints

Definition of constraints in terms of first and higher order derivatives of states and control can be implemented in the future to enable additional OCP formulations.

NLP solver plugins

The NLP is solved using 'ipopt' NLP solver plugin in the present package. The package can be extended with support for other NLP solver plugins available with CasADi.

Support for DAEs

The package can be extended with support for differential-algebraic constraints.

Adaptive schemes

The package can be extended with various grid refinement available in the literature.

9 Conclusion

An open source, extensible, versatile multi-phase nonlinear optimal control problem solver is developed as part of the thesis. A detailed formulation of the transcription process and the solution algorithms are derived and implemented in the solver. The package supports various collocation approximations available in literature including global collocation, pseudo-spectral collocation using various orthogonal polynomials. The package, validated with various test cases from literature is found to be successful in solving various single-phase and multi-phase OCPs. Further, three different adaptive mesh refinement schemes developed and implemented in the package help in producing robust solutions to the OCPs. While the first two proposed grid refinement techniques are iterative, the third method is a unique grid adaptation technique that optimizes the segment widths along with the original optimization problem itself. The proposed iterative adaptive schemes are computationally efficient as they can be initialized with solutions from the previous iterations. The proposed algorithms have been tested on standard test cases for grid adaption techniques namely moon lander OCP and hypersensitive problem. Although all of the proposed methods are found to be successful in solving the test cases, the choice of the total number of collocation nodes is still a parameter chosen by the user.

References

[1] P. Listov and C. Jones, “PolyMPC: An efficient and extensible tool for real-time nonlinear model predictive tracking and path following for fast mechatronic systems,” *Optimal Control Applications and Methods*, vol. 41, no. 2, pp. 709–727, 2020.

[2] B. A. Steinfeldt, M. J. Grant, D. A. Matz, R. D. Braun, and G. H. Barton, “Guidance, Navigation, and Control System Performance Trades for Mars Pinpoint Landing,” *Journal of Spacecraft and Rockets*, vol. 47, pp. 188–198, Jan. 2010.

[3] N. A. o. Engineering, *Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2016 Symposium.* Jan. 2017.

[4] A. Wächter and L. T. Biegler, “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming,” *Mathematical Programming*, vol. 106, pp. 25–57, Mar. 2006.

[5] P. E. Gill, W. Murray, and M. A. Saunders, “SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization,” *SIAM Review*, vol. 47, pp. 99–131, Jan. 2005.

[6] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, “CasADi: a software framework for nonlinear optimization and optimal control,” *Mathematical Programming Computation*, vol. 11, pp. 1–36, Mar. 2019.

[7] J. T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Advances in Design and Control, Society for Industrial and Applied Mathematics, Jan. 2010.

[8] V. M. Becerra, “Solving complex optimal control problems at no cost with PSOPT,” in *2010 IEEE International Symposium on Computer-Aided Control System Design*, pp. 1391–1396, Sept. 2010. ISSN: 2165-302X.

[9] M. A. Patterson and A. V. Rao, “GPOPS-II: A MATLAB Software for Solving Multiple-Phase Optimal Control Problems Using hp-Adaptive Gaussian Quadrature Collocation Methods and Sparse Nonlinear Programming,” *ACM Transactions on Mathematical Software*, vol. 41, pp. 1:1–1:37, Oct. 2014.

[10] I. M. Ross, “Enhancements to the DIDO Optimal Control Toolbox,” *arXiv:2004.13112 [cs, math]*, June 2020. arXiv: 2004.13112.

[11] J. K. Moore and A. D. v. Bogert, “opty: Software for trajectory optimization and parameter identification using direct collocation,” *Journal of Open Source Software*, vol. 3, p. 300, Jan. 2018.

[12] C. L. Darby, W. W. Hager, and A. V. Rao, “An hp-adaptive pseudospectral method for solving optimal control problems,” *Optimal Control Applications and Methods*, vol. 32, no. 4, pp. 476–502, 2011.

[13] J. T. Betts, “Survey of Numerical Methods for Trajectory Optimization,” *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[14] D. A. Benson, G. T. Huntington, T. P. Thorvaldsen, and A. V. Rao, “Direct Trajectory Optimization and Costate Estimation via an Orthogonal Collocation Method,” *Journal of Guidance, Control, and Dynamics*, vol. 29, no. 6, pp. 1435–1440, 2006.

[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, pp. 1843–1851, Nov. 2010.

[16] G. Molnárka, "Fornberg, B.: A Practical Guide to Pseudospectral Methods. Cambridge, Cambridge University Press 1996. X. 231 pp., £37.50. ISBN 0-521-49582-2 (Cambridge Monographs on Applied and Computational Mathematics 1)," *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 77, no. 10, pp. 798–798, 1997.

[17] G. Elnagar, M. Kazemi, and M. Razzaghi, "The pseudospectral Legendre method for discretizing optimal control problems," *IEEE Transactions on Automatic Control*, vol. 40, pp. 1793–1796, Oct. 1995.

[18] F. Fahroo and I. M. Ross, "Direct Trajectory Optimization by a Chebyshev Pseudospectral Method," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 160–166, 2002.

[19] D. Garg, M. Patterson, W. Hager, A. Rao, D. R. Benson, and G. T. Huntington, "An overview of three pseudospectral methods for the numerical solution of optimal control problems." Oct. 2017.

[20] J. Andersson, J. Åkesson, and M. Diehl, "Dynamic optimization with CasADi," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 681–686, Dec. 2012. ISSN: 0743-1546.

[21] "PSOPT/psopt."

[22] A. V. Rao, D. A. Benson, C. Darby, M. A. Patterson, C. Francolin, I. Sanders, and G. T. Huntington, "Algorithm 902: GPOPS, A MATLAB software for solving multiple-phase optimal control problems using the gauss pseudospectral method," *ACM Transactions on Mathematical Software*, vol. 37, pp. 22:1–22:39, Apr. 2010.

[23] S. Kameswaran and L. T. Biegler, "Convergence rates for direct transcription of optimal control problems using collocation at Radau points," *Computational Optimization and Applications*, vol. 41, pp. 81–126, Sept. 2008.

[24] F. Liu, W. W. Hager, and A. V. Rao, "Adaptive mesh refinement method for optimal control using nonsmoothness detection and mesh size reduction," *Journal of the Franklin Institute*, vol. 352, pp. 4081–4106, Oct. 2015.

[25] T. Fujikawa and T. Tsuchiya, "Enhanced Mesh Refinement in Numerical Optimal Control Using Pseudospectral Methods," *SICE Journal of Control, Measurement, and System Integration*, vol. 7, no. 3, pp. 159–167, 2014.

[26] A. Lee, "Fuel-efficient Descent and Landing Guidance Logic for a Safe Lunar Touchdown," in *AIAA Guidance, Navigation, and Control Conference*, American Institute of Aeronautics and Astronautics.

10 Annexure

Links to the package and source code

- **Recommended virtual environment**

```
# Recommended
python3.6 -m venv my_env
source my_env/bin/activate

```

- **Getting started - Pypi :** https://pypi.org/project/mpopt/

```
pip install mpopt
wget https://raw.githubusercontent.com/mpopt/mpopt/master/getting_started.ipynb
jupyter-notebook getting_started.ipynb

```

- **Development code on GitHub :** https://github.com/mpopt

```
git clone https://github.com/mpopt/mpopt.git
make init
make test

# Solve a test case
python examples/multistage_launch_vehicle.py

```

Multistage launch vehicle trajectory ascent optimization code

```
#
# Copyright (c) 2020 LA EPFL.
#
# This file is part of MPOPT
# (see http://github.com/mpopt).
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU Lesser General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU Lesser General Public License for more details.
#
# You should have received a copy of the GNU Lesser General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
#
"""
Created: 6th May 2020
Author : Devakumar Thammisetty
Description : OCP definition for use in NLP transcription
gpops2.com/resources/gpops2UsersGuide.pdf
"""
from mpopt import mp
import numpy as np
```

```python
import casadi as ca

ocp = mp.OCP(n_states=7, n_controls=3, n_phases=4)

# Constants
Re = 6378145.0  # m
omegaE = 7.29211585e-5
rho0 = 1.225
rhoH = 7200.0
Sa = 4 * np.pi
Cd = 0.5
muE = 3.986012e14
g0 = 9.80665

# Variable initialization
lat0 = 28.5 * np.pi / 180.0
r0 = np.array([Re * np.cos(lat0), 0.0, Re * np.sin(lat0)])
v0 = omegaE * np.array([-r0[1], r0[0], 0.0])
m0 = 301454.0
mf = 4164.0
mdrySrb = 19290.0 - 17010.0
mdryFirst = 104380.0 - 95550.0
mdrySecond = 19300.0 - 16820.0
x0 = np.array([r0[0], r0[1], r0[2], v0[0], v0[1], v0[2], m0])

# Step-1 : Define dynamics
# Thrust(N) and mass flow rate(kg/s) in each stage
Thrust = [6 * 628500.0 + 1083100.0, 3 * 628500.0 + 1083100.0, 1083100.0,
    110094.0]
mdot = [
    (6 * 17010.0) / (75.2) + (95550.0) / (261.0),
    (3 * 17010.0) / (75.2) + (95550.0) / (261.0),
    (95550.0) / (261.0),
    16820.0 / 700.0,
]


def dynamics(x, u, t, param=0, T=0.0, mdot=0.0):
    r = x[:3]
    v = x[3:6]
    m = x[6]
    r_mag = ca.sqrt(r[0] * r[0] + r[1] * r[1] + r[2] * r[2])
    v_rel = ca.vertcat(v[0] + r[1] * omegaE, v[1] - r[0] * omegaE, v[2])
    v_rel_mag = ca.sqrt(v_rel[0] * v_rel[0] + v_rel[1] * v_rel[1] + v_rel[2] *
    v_rel[2])
    h = r_mag - Re
    rho = rho0 * ca.exp(-h / rhoH)
    D = -rho / (2 * m) * Sa * Cd * v_rel_mag * v_rel
    g = -muE / (r_mag * r_mag * r_mag) * r

    xdot = [
        x[3],
        x[4],
        x[5],
        T / m * u[0] + param * D[0] + g[0],
        T / m * u[1] + param * D[1] + g[1],
        T / m * u[2] + param * D[2] + g[2],
        -mdot,
    ]
    return xdot
```

```python


def get_dynamics(param):
    dynamics0 = lambda x, u, t: dynamics(
        x, u, t, param=param, T=Thrust[0], mdot=mdot[0]
    )
    dynamics1 = lambda x, u, t: dynamics(
        x, u, t, param=param, T=Thrust[1], mdot=mdot[1]
    )
    dynamics2 = lambda x, u, t: dynamics(
        x, u, t, param=param, T=Thrust[2], mdot=mdot[2]
    )
    dynamics3 = lambda x, u, t: dynamics(
        x, u, t, param=param, T=Thrust[3], mdot=mdot[3]
    )

    return [dynamics0, dynamics1, dynamics2, dynamics3]


ocp.dynamics = get_dynamics(0)


def path_constraints0(x, u, t):
    return [
        u[0] * u[0] + u[1] * u[1] + u[2] * u[2] - 1,
        -u[0] * u[0] - u[1] * u[1] - u[2] * u[2] + 1,
        -ca.sqrt(x[0] * x[0] + x[1] * x[1] + x[2] * x[2]) / Re + 1,
    ]


ocp.path_constraints = [path_constraints0] * ocp.n_phases


def terminal_cost3(xf, tf, x0, t0):
    return -xf[-1] / m0


ocp.terminal_costs[3] = terminal_cost3


def terminal_constraints3(x, t, x0, t0):
    # https://space.stackexchange.com/questions/1904/how-to-programmatically-
    calculate-orbital-elements-using-position-velocity-vecto
    # http://control.asu.edu/Classes/MAE462/462Lecture06.pdf
    h = ca.vertcat(
        x[1] * x[5] - x[4] * x[2], x[3] * x[2] - x[0] * x[5], x[0] * x[4] - x[1]
     * x[3]
    )

    n = ca.vertcat(-h[1], h[0], 0)
    r = ca.sqrt(x[0] * x[0] + x[1] * x[1] + x[2] * x[2])

    e = ca.vertcat(
        1 / muE * (x[4] * h[2] - x[5] * h[1]) - x[0] / r,
        1 / muE * (x[5] * h[0] - x[3] * h[2]) - x[1] / r,
        1 / muE * (x[3] * h[1] - x[4] * h[0]) - x[2] / r,
    )

    e_mag = ca.sqrt(e[0] * e[0] + e[1] * e[1] + e[2] * e[2])
    h_sq = h[0] * h[0] + h[1] * h[1] + h[2] * h[2]
    v_mag = ca.sqrt(x[3] * x[3] + x[4] * x[4] + x[5] * x[5])

    a = -muE / (v_mag * v_mag - 2.0 * muE / r)
    i = ca.acos(h[2] / ca.sqrt(h_sq))
    n_mag = ca.sqrt(n[0] * n[0] + n[1] * n[1])

    node_asc = ca.acos(n[0] / n_mag)
    # if n[1] < -1e-12:
    node_asc = 2 * np.pi - node_asc

    argP = ca.acos((n[0] * e[0] + n[1] * e[1]) / (n_mag * e_mag))
    # if e[2] < 0:
    #     argP = 2*np.pi - argP

    a_req = 24361140.0
    e_req = 0.7308
    i_req = 28.5 * np.pi / 180.0
    node_asc_req = 269.8 * np.pi / 180.0
    argP_req = 130.5 * np.pi / 180.0

    return [
        (a - a_req) / (Re),
        e_mag - e_req,
        i - i_req,
        node_asc - node_asc_req,
        argP - argP_req,
    ]


ocp.terminal_constraints[3] = terminal_constraints3

ocp.scale_x = [
    1 / Re,
    1 / Re,
    1 / Re,
    1 / np.sqrt(muE / Re),
    1 / np.sqrt(muE / Re),
    1 / np.sqrt(muE / Re),
    1 / m0,
]
ocp.scale_t = np.sqrt(muE / Re) / Re


# Initial guess estimation
def ae_to_rv(a, e, i, node, argP, th):
    p = a * (1.0 - e * e)
    r = p / (1.0 + e * np.cos(th))

    r_vec = np.array([r * np.cos(th), r * np.sin(th), 0.0])
    v_vec = np.sqrt(muE / p) * np.array([-np.sin(th), e + np.cos(th), 0.0])

    cn, sn = np.cos(node), np.sin(node)
    cp, sp = np.cos(argP), np.sin(argP)
    ci, si = np.cos(i), np.sin(i)

    R = np.array(
        [
            [cn * cp - sn * sp * ci, -cn * sp - sn * cp * ci, sn * si],
            [sn * cp + cn * sp * ci, -sn * sp + cn * cp * ci, -cn * si],
            [sp * si, cp * si, ci],
            ]
    )

    r_i = np.dot(R, r_vec)
    v_i = np.dot(R, v_vec)

    return r_i, v_i


# Target conditions
a_req = 24361140.0
e_req = 0.7308
i_req = 28.5 * np.pi / 180.0
node_asc_req = 269.8 * np.pi / 180.0
argP_req = 130.5 * np.pi / 180.0
th = 0.0
rf, vf = ae_to_rv(a_req, e_req, i_req, node_asc_req, argP_req, th)

# Timings
t0, t1, t2, t3, t4 = 0.0, 75.2, 150.4, 261.0, 924.0
# Interpolate to get starting values for intermediate phases
xf = np.array([rf[0], rf[1], rf[2], vf[0], vf[1], vf[2], mf + mdrySecond])
x1 = x0 + (xf - x0) / (t4 - t0) * (t1 - t0)
x2 = x0 + (xf - x0) / (t4 - t0) * (t2 - t0)
x3 = x0 + (xf - x0) / (t4 - t0) * (t3 - t0)

# Update the state discontinuity values across phases
x0f = np.copy(x1)
x0f[-1] = x0[-1] - (6 * 17010.0 + 95550.0 / t3 * t1)
x1[-1] = x0f[-1] - 6 * mdrySrb

x1f = np.copy(x2)
x1f[-1] = x1[-1] - (3 * 17010.0 + 95550.0 / t3 * (t2 - t1))
x2[-1] = x1f[-1] - 3 * mdrySrb

x2f = np.copy(x3)
x2f[-1] = x2[-1] - (95550.0 / t3 * (t3 - t2))
x3[-1] = x2f[-1] - mdryFirst

# Step-8b: Initial guess for the states, controls and phase start and final
#             times
ocp.x00 = np.array([x0, x1, x2, x3])
ocp.xf0 = np.array([x0f, x1f, x2f, xf])
ocp.u00 = np.array([[1, 0, 0], [1, 0, 0], [0, 1, 0], [0, 1, 0]])
ocp.uf0 = np.array([[0, 1, 0], [0, 1, 0], [0, 1, 0], [0, 1, 0]])
ocp.t00 = np.array([[t0], [t1], [t2], [t3]])
ocp.tf0 = np.array([[t1], [t2], [t3], [t4]])

# Step-8c: Bounds for states
rmin, rmax = -2 * Re, 2 * Re
vmin, vmax = -10000.0, 10000.0
rvmin = [rmin, rmin, rmin, vmin, vmin, vmin]
rvmax = [rmax, rmax, rmax, vmax, vmax, vmax]
lbx0 = rvmin + [x0f[-1]]
lbx1 = rvmin + [x1f[-1]]
lbx2 = rvmin + [x2f[-1]]
lbx3 = rvmin + [xf[-1]]
ubx0 = rvmax + [x0[-1]]
ubx1 = rvmax + [x1[-1]]
ubx2 = rvmax + [x2[-1]]
```

```python
ubx3 = rvmax + [x3[-1]]
ocp.lbx = np.array([lbx0, lbx1, lbx2, lbx3])
ocp.ubx = np.array([ubx0, ubx1, ubx2, ubx3])

# Bounds for control inputs
umin = [-1, -1, -1]
umax = [1, 1, 1]
ocp.lbu = np.array([umin] * ocp.n_phases)
ocp.ubu = np.array([umax] * ocp.n_phases)

# Bounds for phase start and final times
ocp.lbt0 = np.array([[t0], [t1], [t2], [t3]])
ocp.ubt0 = np.array([[t0], [t1], [t2], [t3]])
ocp.lbtf = np.array([[t1], [t2], [t3], [t4 - 100]])
ocp.ubtf = np.array([[t1], [t2], [t3], [t4 + 100]])

# Event constraint bounds on states : State continuity/disc.
lbe0 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -6 * mdrySrb]
lbe1 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -3 * mdrySrb]
lbe2 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -mdryFirst]
ocp.lbe = np.array([lbe0, lbe1, lbe2])
ocp.ube = np.array([lbe0, lbe1, lbe2])
ocp.validate()

# Solve with drag disables
ocp.dynamics = get_dynamics(0)

if __name__ == "__main__":
    mpo = mp.mpopt(ocp, 1, 11)
    sol = mpo.solve()

    # Solve with drag enabled and initial guess
    ocp.dynamics = get_dynamics(1)
    ocp.validate()

    mpo._ocp = ocp
    sol = mpo.solve(
        sol, reinitialize_nlp=True, nlp_solver_options={"ipopt.acceptable_tol":
    1e-6}
    )
    print("Final mass : ", round(-sol["f"].full()[0, 0] * m0, 4))

    mp.post_process._INTERPOLATION_NODES_PER_SEG = 200
    # Post processing
    post = mpo.process_results(sol, plot=False, scaling=False)

    # ************** Plot height and velocity ************************
    x, u, t = post.get_data(interpolate=True)
    print("Final time : ", t[-1])
    figu, axsu = post.plot_u()

    # Plot mass
    figm, axsm = post.plot_single_variable(
        x * 1e-3, t, [[-1]], axis=0, fig=None, axs=None, tics=["-"] * 15
    )

    # Compute and plot altitude, velocity
    r = 1e-3 * (np.sqrt(x[:, 0] ** 2 + x[:, 1] ** 2 + x[:, 2] ** 2) - Re)
    v = 1e-3 * np.sqrt(x[:, 3] ** 2 + x[:, 4] ** 2 + x[:, 5] ** 2)
    y = np.column_stack((r, v))
```

```
fig, axs = post.plot_single_variable(y, t, [[0], [1]], axis=0)

x, u, t = post.get_data(interpolate=False)
r = 1e-3 * (np.sqrt(x[:, 0] ** 2 + x[:, 1] ** 2 + x[:, 2] ** 2) - Re)
v = 1e-3 * np.sqrt(x[:, 3] ** 2 + x[:, 4] ** 2 + x[:, 5] ** 2)
y = np.column_stack((r, v))
fig, axs = post.plot_single_variable(
    y, t, [[0], [1]], axis=0, fig=fig, axs=axs, tics=["."] * 15
)
axs[0].set(ylabel="Altitude, km", xlabel="Time, s")
axs[1].set(ylabel="Velocity, km/s", xlabel="Time, s")

# Plot mass at the collocation nodes
figm, axsm = post.plot_single_variable(
    x * 1e-3, t, [[-1]], axis=0, fig=figm, axs=axsm, tics=["."] * 15
)

mp.plt.show()
```