The Tiger Optimization Software - A Pseudospectral Optimal Control Package

by

Sam Sowell

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
December 10, 2022

Keywords: Pseudospectral, Optimal Control, Trajectory Optimization

Approved by

Ehsan Taheri, Chair, Assistant Professor of Aerospace Engineering
Davide Guzzetti, Assistant Professor of Aerospace Engineering
Nan Li, Assistant Professor of Aerospace Engineering

Abstract

The fields of trajectory optimization and optimal control are closely connected. Practical trajectory optimization techniques are built on the *indirect* and *direct* methods for solving optimal control problems. Indirect methods seek to satisfy the first-order necessary conditions of optimality, while direct methods discretize the problem and transcribe it into a large-scale nonlinear programming (NLP) problem. This study provides a basic overview of the direct and indirect methods. The indirect necessary conditions and Pontryagin's Minimum Principle are reviewed, and the approaches taken by direct methods are presented. The focus of this study then shifts to pseudospectral direct methods, which belong to the class of collocation methods. The theoretical groundwork of pseudospectral methods is laid, leading to a pseudospectral formulation and software for solving general optimal control problems. Several types of pseudospectral methods are presented, including the Legendre-Gauss and Chebyshev-Gauss methods. Special attention is given to the Legendre-Gauss-Radau (LGR) method, which is the primary transcription employed by the MATLAB-based Tiger Optimization Software (TOPS). TOPS is a general-purpose pseudospectral optimal control software developed by the author as part of their research in the Aero-Astro Computational and Experimental (ACE) Lab. A multi-interval pseudospectral method is presented, and the discrete form of the optimality conditions are derived. The study shifts focus from theory to application, and the practical aspects of pseudospectral optimal control methods are discussed. Another objective of this research is to compile the author's knowledge with regard to implementing pseudospectral techniques, thus enabling the reader to easily implement their own pseudospectral method. Several mesh refinement methods are compared and their merits are compared. In addition, several techniques that improve the efficiency of an NLP solver are presented, including efficient and exact derivative formulas and several scaling techniques. Finally, several example optimal control problems are solved using TOPS. The solutions obtained from TOPS are compared to the solutions obtained using indirect techniques to verify their accuracy and demonstrate the capabilities of TOPS.

Acknowledgments

There are many people whose support and encouragement have given me the motivation and resolve to reach the end of this journey. I would first like to dedicate this work to my parents, Charles and Allison Sowell, for supporting me unconditionally throughout my undergraduate and graduate journey. I could not have done this without them. They will always have my love and appreciation for being the incredible parents that they are.

I would also like to thank my advisor, Dr. Ehsan Taheri, for all that he has done for me. He has always pushed me to learn and to love learning, and that is something that we both share. I remember receiving a Canvas message from Dr. Taheri when I was taking his orbital mechanics course, asking me to stop by his office. Truth be told, I was a bit worried that Dr. Taheri thought I had cheated on a test (I was, of course, innocent). When he asked if I wanted to do undergraduate research with him, I was more than surprised. I would have never considered graduate school if Dr. Taheri hadn't reached out to me, and I am very thankful he did.

I would also like to thank my ACE Lab fellows, Jack, Keziban, Nick, and Praveen, for complaining with me when things got stressful. They have been the best of companions on this journey. I wish them the best of luck as they pursue their Ph.Ds, but unfortunately our paths must part here, despite the best efforts of my advisor. Sorry Dr. Taheri, maybe one day I will come back for a Ph.D.!

As well, I would like to thank the members of my committee, Dr. Ehsan Taheri, Dr. Davide Guzzetti, and Dr. Nan Li, for reviewing and editing this thesis. Their feedback and suggestions were invaluable and improved the quality of this document.

Finally, I would like to thank the Department of Aerospace Engineering for giving me the opportunity to study and pursue a Master's degree and for supporting me financially.

War Eagle!

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AD	Automatic Differentiation
AD	Automatic/Algorithmic Differentiation
CGC	Computational Guidance and Control
CGL	Chebyshev-Gauss-Lobatto
COV	Calculus of Variations
CSC	Composite Smooth Control
DDP	Differential Dynamic Programming
ECI	Earth-Centered-Inertial
EDL	Entry, Descent, and Landing
EP	Electric Propulsion
GNC	Guidance, Navigation, and Control
HBVPs	Hamiltonian Boundary-Value Problems
HTS	Hyperbolic Tangent Smoothing
IPM	Interior Point Method
KKT	Karush-Kuhn-Tucker
LEO	Low Earth Orbit

LGL	Legendre-Gauss-Lobatto
LGR	Legendre-Gauss-Radau
NLP	Nonlinear Program
OCP	Optimal Control Problem
OO	Operator-Overloading
OOTL	Out-of-the-Loop
PJRN	Projected-Jacobian Rows Normalization
PMP	Pontryagin's Minimum Principle
PS	Pseudospectral
S2S	Source-to-Source
SCP	Sequential Convex Programming
SQP	Sequential Quadratic Program
SQP	Sequential Quadratic Programming
TPBVP	Two-Point Boundary Value Problem
UTM	Unified Trigonometrization Method

Chapter 1

Introduction

To touch the stars has been the dream of mankind since before recorded history. Myths and legends that are ingrained in our consciousness sprung from the earliest stories parents told their children about the stars. Their secrets call us to watch them, name them, and separate them from each other. But to truly reveal the secrets of the heavens, we must be able to reach them. If this is not impossible, we can, at least, try to get a better view. With the advent of modern technology, it became possible for humanity to witness the heavens firsthand, either through manned spaceflight or unmanned probes and telescopes. However, along the way, we discovered that it was incredibly difficult and costly to explore space. The technologies needed to do so, such as communication and propulsion systems, are expensive to develop and build. Life support systems are needed for humans to survive in the harsh vacuum of space. To make matters worse, Earth's gravity is so difficult to escape that propellant comprises over 80 percent of most launch vehicles! In fact, fuel expenditure is where most of the non-recoupable cost lies in space exploration. Reducing the fuel required to reach a desired science objective is one of the most fundamental problems of space exploration. However, reducing propellant expenditure is second to actually reaching a target. Motion within an inverse-square gravity field is notoriously non-intuitive and becomes chaotic when multiple bodies are present. All of these factors combine into a seemingly insurmountable challenge.

In spite of this, humanity has risen to meet it. Two new fields of mathematics were created to describe and predict the motion of the heavenly bodies - orbital mechanics, pioneered by Kepler [152] and Newton [181] and general/special relativity, pioneered by Einstein [42]. The explosion of computational technology in the last 60 years is due in no small part to the field of space exploration. Computers were designed that could make rapid calculations and generate safe trajectories within these complex gravity fields. Science objectives were finally achievable. As manufacturing techniques improved, we created powerful computers small enough to carry on-board spacecraft or in a backpack. Since practical trajectory optimization is primarily a computational practice for real-world problems, these powerful computers provided tools for solving difficult trajectory design problems [15]. Today, feasible and safe trajectories can be easily generated, so the focus has shifted to improving (i.e., optimizing) these trajectories.

Optimization-based methods are a subset of computational guidance and control (CGC), which is itself a subset of GNC discipline. During space mission design, optimization methods are typically applied to launchers, planetary landers, satellites, and manned spacecraft with the goal of improving their operational efficiency by some measure. In a general sense, the goal is to solve the optimization problem defined in Eq. (1.1)

$$\min_{\boldsymbol{x}(\cdot)\in\mathcal{X}} J\left[\boldsymbol{x}(\cdot)\right], \tag{1.1}$$

where $J : \mathbb{R}^n \to \mathbb{R}$ is a cost *functional*, $\mathcal{X} \subseteq \mathbb{R}^n$ denotes the state admissible sets, where n is the number of states variables [111]. A cost functional, J, is a "function of functions" that takes an entire function and outputs a scalar. We distinguish the points $\boldsymbol{x}$ from the function, $\boldsymbol{x}(\cdot)$. The study of functionals is rooted in calculus of variations, and optimal control theory is a branch of calculus of variations [145]. From an engineering perspective, the objective of optimal control theory is to determine control time-history that will cause a process to satisfy physical constraints while simultaneously minimizing (or maximizing) some performance criterion [91].

Inspecting Eq. (1.1) we can see that optimal control theory is so general that it can be applied to any controllable physical process. However, due to author's research focus on space mission design, the discussion of optimal control will be restricted to the field of space vehicle trajectory optimization. To understand why optimization is important to this field, consider the recent and future operations that space vehicles will perform. In recent years, in an effort to drive down the cost of spaceflight, reusable launchers and planetary landers have become popular. These vehicles require accurate Entry, Descent, and Landing (EDL) algorithms to reduce fuel consumption or access scientifically interesting regions autonomously [20]. These scientifically interesting regions are oftentimes hazardous, and future landers are required to navigate challenging environments in real-time with no human input [97, 142] due to communication delays or the risk and cost of a manned mission. In the future, space missions will require autonomous docking of spacecraft [192], which was recently demonstrated by SpaceX's Dragon capsule that automatically docked with the ISS [118]. Autonomous control of space vehicles is one of the most promising and actively researched applications of optimal control theory.

Advances in the field of space propulsion have also led to the development and use of efficient electric propulsion systems [96]. Low-thrust electric propulsion systems are very attractive due to high specific impulse values that reduce the amount of propellant required for a desired Δv, which is a limiting factor for space missions. However, the resulting low-thrust trajectory optimization problems become numerically difficult to solve due to long transfer times and many-revolution nature of the resulting trajectories [166, 70]. The challenges presented by low-thrust trajectory design have been studied for many years. Early work in the field by Edelbaum [40] used the calculus of variations to derive optimal steering laws that model thrust as orbital element perturbations. Haissig et al. [75] used averaging methods to obtain low-thrust minimum-fuel transfers between coplanar elliptical orbits. These analytic averaging techniques as well as Lyapunov guidance laws [135] have been used extensively to design low-thrust multiple-revolution trajectories [95, 57, 79, 77, 134, 82].

These continuous-thrust trajectory optimization problems have served as one of the motivating problems for optimal control theory since its inception [150]. The application of optimal control theory allows for these problems to be solved efficiently and quickly using a variety of optimization methods [8]. This study will primarily focus on one of the most widely used optimization methods, namely, direct methods [15]. However, indirect methods will also be presented, although with less depth.

1.1 Notation

Before proceeding, it is worthwhile to briefly discuss the notation that will be used throughout this study. This study will use notation closely matching that of Ross [145] with changes appropriately highlighted.

In this study, dimensions of vectors and spaces are denoted by $N_{(\cdot)}$, where $(\cdot)$ is replaced by a subscript that appropriately links the dimension to the vector of interest. For example, N_x denotes the dimension of a vector $\boldsymbol{x}$, and $\boldsymbol{x} \in \mathbb{R}^{N_x}$. Additionally, $\boldsymbol{x} \in \mathbb{R}^{1\times N_x}$ denotes a row vector, while $\boldsymbol{x} \in \mathbb{R}^{N_x\times 1}$ denotes a column vector. An $n \times m$ matrix has n rows and m columns. However, this notation will sometimes be dropped for the sake of generality. When considering a matrix $\boldsymbol{P} \in \mathbb{R}^{n\times m}$, the subscript $\boldsymbol{P}_i$ will denote the i-th *row* of the matrix. The j-th column will be denoted by $\boldsymbol{P}_{(:,j)}$. Extending this convention, $\boldsymbol{P}_{i:j}$ will denote rows i through j and $\boldsymbol{P}_{(:,i:j)}$ will denote columns i through j. We will let $\mathbf{0}_{n\times m}$ denote an all-zeros matrix of size $n \times m$ and $\mathbf{1}_{n\times m}$ to denote a matrix of all ones of size $n \times m$. In addition, we define the "unrolled" form of $\boldsymbol{P}$ as,

$$\boldsymbol{P}_{(:)} = \begin{bmatrix} \boldsymbol{P}_{(:,1)} \\ \boldsymbol{P}_{(:,2)} \\ \vdots \\ \boldsymbol{P}_{(:,m)} \end{bmatrix}, \tag{1.2}$$

where $\boldsymbol{P}_{(:)} \in \mathbb{R}^{nm}$ is a column vector resulting from each column of $\boldsymbol{P}$ being placed under the previous column while traversing $\boldsymbol{P}$ from left to right. The unrolled form of $\boldsymbol{P}$ is primarily useful for computational indexing purposes. The operation $\mathrm{diag}(\boldsymbol{p})$, where $\boldsymbol{p} \in \mathbb{R}^n$, denotes the following operation:

$$\mathrm{diag}(\boldsymbol{p}) = \begin{bmatrix} p_1 & 0 & \cdots & 0 \\ 0 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p_n \end{bmatrix}, \tag{1.3}$$

where p_i is the i-th element of $\boldsymbol{p}$. In addition, $\langle \boldsymbol{A}, \boldsymbol{B} \rangle$ denotes the standard matrix innter product defined by,

$$\langle \boldsymbol{A}, \boldsymbol{B} \rangle = \mathrm{trace}(\boldsymbol{A}^\top \boldsymbol{B}). \tag{1.4}$$

Since many functions, functionals, and their values are represented through letters with superscripts or subscripts, we distinguish between them. The symbol $\boldsymbol{x}$ denotes a point in the space $\mathbb{R}^{N_x}$. The symbol $\boldsymbol{x}(\cdot)$ denotes an entire function in the space, $\mathcal{X}$. The symbol $\boldsymbol{x}(t)$ denotes the function $\boldsymbol{x}(\cdot)$ evaluated at a point t. It follows that $\boldsymbol{x}(t_0)$ means the function $\boldsymbol{x}(\cdot)$ evaluated at the point $\boldsymbol{x}(t_0)$. For the sake of convenience, we oftentimes abbreviate this function evaluation as $\boldsymbol{x}_0$. The symbol $\boldsymbol{x}^0$ will typically be used to denote the numerical value of $\boldsymbol{x}_0$.

Next, we discuss more matrix notation that is relevant for programming and compact representation of a discrete OCP. Consider $\boldsymbol{f}(\boldsymbol{x}) : \mathbb{R}^{N_x} \to \mathbb{R}^{N_f}$, which is a function that maps row vectors $\in N_x$ to row vectors $\in N_f$. When implementing such a function on a computer, we may wish to evaluate $\boldsymbol{f}$ at the points $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$. This operation is denoted as follows.

$$\boldsymbol{X} \in \mathbb{R}^{N \times N_x} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix}, \qquad \boldsymbol{f}(\boldsymbol{X}) \in \mathbb{R}^{N \times N_f} = \begin{bmatrix} \boldsymbol{f}(\boldsymbol{x}_1) \\ \boldsymbol{f}(\boldsymbol{x}_2) \\ \vdots \\ \boldsymbol{f}(\boldsymbol{x}_N) \end{bmatrix}. \tag{1.5}$$

Thus, we treat $\boldsymbol{f}(\cdot)$ or any such function as an overloaded operator (see pp.8-9 of [145]) that may produce a vector or matrix output, depending on the dimensions of the input. Occasionally, we may denote $\boldsymbol{f}(\boldsymbol{X})$ as $[\boldsymbol{f}(\boldsymbol{x}_i)]_{i=1}^{N}$ to distinguish between differently sized $\boldsymbol{f}(\boldsymbol{X})$ if we are considering multiple distinct $\boldsymbol{X}$.

It is also worthwhile to define the notation for derivatives of vectors and vector-valued functions. Consider $f(\boldsymbol{x}) : \mathbb{R}^{N_x} \to \mathbb{R}$ to be a function that maps a row or column vector to scalars. The first derivative of f with respect to $\boldsymbol{x}$ is referred to as the *gradient*, and is given by,

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \in \mathbb{R}^{1 \times N_x} = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_{N_x}}\right]. \tag{1.6}$$

Next, consider $\boldsymbol{f}(\boldsymbol{x}) : \mathbb{R}^{N_x} \to \mathbb{R}^{N_f}$. Here, $\boldsymbol{x}$ may be a row or column vector and the operation $\boldsymbol{f}(\boldsymbol{x})$ has the same orientation as $\boldsymbol{x}$. The first derivative of this function with respect to $\boldsymbol{x}$ is referred to as the *Jacobian*, and is given by,

$$\nabla_{\boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^{N_f \times N_x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_{N_x}} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_{N_x}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{N_f}}{\partial x_1} & \frac{\partial f_{N_f}}{\partial x_2} & \cdots & \frac{\partial f_{N_f}}{\partial x_{N_x}} \end{bmatrix}. \tag{1.7}$$

Finally, consider the second derivative of a scalar-valued function $f(\boldsymbol{x}, \boldsymbol{y}) : \mathbb{R}^{N_x} \times \mathbb{R}^{N_y} \rightarrow \mathbb{R}$. The mixed second derivative with respect to $\boldsymbol{x}$ and $\boldsymbol{y}$ is given by,

$$\nabla_{\boldsymbol{xy}} f(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^{N_x \times N_y} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial y_1} & \frac{\partial^2 f}{\partial x_1 \partial y_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial y_{N_y}} \\ \frac{\partial^2 f}{\partial x_2 \partial y_1} & \frac{\partial^2 f}{\partial x_2 \partial y_2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial y_{N_y}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_{N_x} \partial y_1} & \frac{\partial^2 f}{\partial x_{N_x} \partial y_2} & \cdots & \frac{\partial^2 f}{\partial x_{N_x} \partial y_{N_y}} \end{bmatrix}. \tag{1.8}$$

If $f(\boldsymbol{x})$ is a function of a single vector, then this mixed partial derivative matrix is referred to as the *Hessian*, and is given by,

$$\nabla_{\boldsymbol{xx}} f(\boldsymbol{x}) \in \mathbb{R}^{N_x \times N_x} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_{N_x}} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_{N_x}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_{N_x} \partial x_1} & \frac{\partial^2 f}{\partial x_{N_x} \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_{N_x}^2} \end{bmatrix}. \tag{1.9}$$

Note that the second derivative matrices are symmetric about the diagonal. Thus,

$$\nabla_{\boldsymbol{xy}} f(\boldsymbol{x}, \boldsymbol{y}) = [\nabla_{\boldsymbol{yx}} f(\boldsymbol{x}, \boldsymbol{y})]^\top \quad \text{and} \quad \nabla_{\boldsymbol{xx}} f(\boldsymbol{x}) = [\nabla_{\boldsymbol{xx}} f(\boldsymbol{x})]^\top.$$

The reader is informed that discussion of the topics in this study may necessitate abuse or change of these notations. Any such abuse or change will be highlighted and the difference in meaning will be explained.

1.2 The Bolza Optimal Control Problem

The formulation of an optimal control problem requires:

1. A mathematical model of the system or process that is being controlled.

2. A statement of the physical constraints.

3. A performance criterion (cost functional).

It is advantageous to obtain the simplest possible mathematical description that will accurately predict the response of such a system or process to known inputs. Most systems (but not all) studied in optimal control theory are described using time-varying ordinary differential equations (ODEs). These represent the system dynamics and are typically represented in a state-space form. Consider some state variables (states), $x_1(t), x_2(t), \ldots, x_n(t)$, and some control variables (controls), $u_1(t), u_2(t), \ldots, u_m(t)$. We may write the time rate of change of the states as,

$$\begin{aligned}
\dot{x}_1(t) &= f_1(x_1(t), \ldots x_n(t), u_1(t), \ldots, u_m(t), t), \\
\dot{x}_2(t) &= f_2(x_1(t), \ldots x_n(t), u_1(t), \ldots, u_m(t), t), \\
&\vdots \\
\dot{x}_n(t) &= f_n(x_1(t), \ldots x_n(t), u_1(t), \ldots, u_m(t), t),
\end{aligned}$$

where f_i is an algebraic expression. Thus, we can define the state and control vectors of the system as follows

$$\boldsymbol{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix} \in \mathbb{R}^{n\times 1}, \qquad \boldsymbol{u}(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_m(t) \end{bmatrix} \in \mathbb{R}^{m\times 1}. \tag{1.10}$$

This allows us to write the system dynamics in a compact state-space form as,

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t). \tag{1.11}$$

Many sources use n and m to denote the dimension of the state and control, respectively. However, for the sake of clarity, we adopt the notations N_x and N_u to denote the dimension of the states and controls. Thus, $\boldsymbol{x}(t) \in \mathbb{R}^{N_x}$, $\boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \in \mathbb{R}^{N_x}$, and $\boldsymbol{u}(t) \in \mathbb{R}^{N_u}$. In Eq. (1.11), the time $t \in [t_0, t_f]$. It is important to note that the left- and right-hand sides of Eq. (1.11) should not be conflated. Although they are equal, the right-hand side, $\boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t)$ is a "rule" that is independent of $\dot{\boldsymbol{x}}(t)$ (see p.7 in [145]). In fact, Eq. (1.11) is often treated as a constraint. However, it is such an important constraint that it should be treated differently than other constraints. Any other constraint can be represented as,

$$\boldsymbol{e}^L \leq \boldsymbol{e}(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f) \leq \boldsymbol{e}^U, \tag{1.12}$$

$$\boldsymbol{h}^L \leq \boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \leq \boldsymbol{h}^U, \tag{1.13}$$

where $\boldsymbol{e}(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f) \in \mathbb{R}^{N_e}$ are called the *event* constraints, whereas $\boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \in \mathbb{R}^{N_h}$ are called *path* constraints. The vectors $\boldsymbol{e}^L$ and $\boldsymbol{e}^U$ are the N_e-dimensional lower and upper bounds on the event constraints, respectively. The vectors $\boldsymbol{h}^L$ and $\boldsymbol{h}^U$ are the N_h-dimensional lower and upper bounds on the path constraints, respectively. Event constraints are functions only of initial and final states and times. For this reason, they are often referred to as *boundary* constraints. The path constraints are functions of the state, control, and time at any point along the trajectory. It is common in the literature to restate Eq. (1.12) and Eq. (1.13) as,

$$\boldsymbol{e}(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f) \leq \boldsymbol{0}, \tag{1.14}$$

$$\boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \leq \boldsymbol{0}, \tag{1.15}$$

by simply introducing additional inequality constraints. Although Eq. (1.14) and Eq. (1.15) are admittedly more opaque than their previous forms, they are used more frequently. Additionally, Eq. (1.14) and Eq. (1.15) are more compact and in many cases more friendly to computational implementation. As such, we will use the forms given in Eq. (1.14) and Eq. (1.15). Note that Eq. (1.14) is sometimes altered and takes the form,

$$\boldsymbol{e}(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f) = \mathbf{0}. \tag{1.16}$$

This form arises primarily from convention, as in practice event or boundary constraints are most often enforced as an equality constraint. It is less common to set a range of values as a final condition rather than an algebraic constraint or numerical value. However, Eq. (1.14) is more general, so it is adopted.

The performance criterion or *cost functional* may be represented as,

$$J[\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot), t_0, t_f] = E(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f) + \int_{t_0}^{t_f} F(\boldsymbol{x}(t), \boldsymbol{u}(t), t)dt. \tag{1.17}$$

In Eq. (1.17), $E(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f) \in \mathbb{R}$ is called the endpoint/terminal cost and is a function of initial and final states and times. This term is also called the Mayer cost in a traditional setting. The term $F(\boldsymbol{x}(t), \boldsymbol{u}(t) \in \mathbb{R}$ is called the running cost, and it is a function of the *instantaneous* state, control, and time at every point along the trajectory. It is traditionally called the Lagrange cost. In Eq. (1.17), the arguments of J are the decision variables. They are also *functions*, while the arguments of E and F are those functions evaluated along the trajectory or at its boundaries. While it may not be obvious at first, the state is indeed a decision variable. Given a control time history, many state time histories may be dynamically feasible (i.e., satisfy the dynamic constraints). However, only some $\boldsymbol{x}(\cdot)$ and $\boldsymbol{u}(\cdot)$ will satisfy the dynamics and other constraints while minimizing the cost.

Now that we have formulated a model, constraints, and performance criterion, we may formulate a general framework for OCPs. Consider the continuous-time OCP defined on the interval $t \in [t_0, t_f]$.

$$
\boldsymbol{x} \in \mathbb{R}^{N_x}, \;\; \boldsymbol{u} \in \mathbb{R}^{N_u}, \;\; t_0, \;\; t_f,
$$

$$
(\mathrm{B}) \begin{cases} \text{Minimize} \quad J\left[\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot), t_0, t_f\right] = E(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f) + \displaystyle\int_{t_0}^{t_f} F(\boldsymbol{x}(t), \boldsymbol{u}(t), t) dt, \\ \text{Subject to :} \qquad \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t), \\ \qquad \boldsymbol{e}(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f) \leq \mathbf{0}, \\ \qquad \boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \leq \mathbf{0}. \end{cases}
$$

Problem B is known as the Bolza OCP, and is a general form of a constrained, time-varying OCP with equality and inequality along-the-path state constraints.

1.3 Direct and Indirect Methods

Trajectory optimization problems are typically solved through the use of *direct* or *indirect* methods [180, 15] or genetic/evolutionary algorithms [46, 160]. As stated previously, this study will focus on direct and indirect methods. On a high level, direct methods "discretize then optimize" by transcribing the original OCP into a nonlinear program (NLP) problem through some parameterization scheme of the states and controls. The discrete problem can then be solved using a variety of robust NLP solvers. Direct methods can further be divided into *simultaneous* methods that parameterize the state and control and *sequential* methods that parameterize the control only [49, 88]. Indirect methods, on the other hand, "optimize then discretize" by analytically deriving the set of first-order necessary (and in some cases, second-order [85, 25] sufficinet) conditions of optimality for the problem and applying Pontryagin's Minimum Principle (PMP) to obtain an extremal control. This process results in HBVPs and in simpler cases in two-point boundary-value problems (TPBVPs), typically solved using single- or multiple-shooting methods that produce high-resolution solutions.

The primary advantage of indirect methods, when applied to low-thrust trajectory design, is that high-resolution solutions can be obtained that are guaranteed to be local extrema [31].

Additionally, indirect methods are more numerically tractable for long-duration low-thrust trajectory optimization problems. A major difficulty that is inherent to indirect methods is the sensitivity of the solution to the unknown initial costate values, which must be guessed to solve the TPBVPs [166, 168]. This often limits practical applications of the indirect method. To alleviate this sensitivity, numerical methods such as continuation and homotopy are used [166, 173]. These techniques begin by solving a simplified form of the desired problem. The solution is used to solve progressively more difficult forms of the problem until the original problem solution is recovered within some tolerance.

On the other hand, the appeal of direct methods is their robustness and large radius of convergence [31]. In addition, path constraints (which indirect methods traditionally have difficulty incorporating) are trivial to include. Dickmanns et al. [36] and Hargraves and Paris [76] popularized direct methods within the aerospace community by solving an OCP using Hermite polynomial function approximations. Direct methods are now extremely mature and used extensively in the aerospace field. The distinctions between different direct approaches primarily lie in the method in which the integration rules are constructed. The most common approaches use the trapezoid rule [16] or Hermite-Simpson Runge-Kutta methods [76]. One that has become very popular in the last two decades are pseudospectral (PS) methods [12, 58, 60]. These methods use a basis of global orthogonal Legendre or Chebychev polynomials to approximate the states, while the control variables are arbitrarily approximated [47, 49]. In addition to highly accurate solutions for a relatively sparse NLP, PS methods also allow for a mapping between the Karush-Kuhn-Tucker (KKT) multipliers used internally by many NLP solvers and the continuous time costates [49, 12, 58, 60]. This mapping produces a costate estimate that may be used to initialize indirect shooting schemes.

1.4 Contributions and TOPS

This study primarily focuses on PS direct methods and the development of an in-house MATLAB solver to be used as a general-purpose optimal control software package. However, indirect methods will be presented for comparison with direct methods. Fig. 1.1 shows a broad overview of the different types of optimal control methods. This thesis focuses on the topics enclosed in the red box in Fig. 1.1. Additionally, this document will present multiple flavors of PS methods as well as techniques to improve their capability and performance in solving challenging OCPs. This document also seeks to dispel the air of vagueness that often surrounds the literature when it comes to implementing PS methods. Finally, the elements of PS theory that are implemented in the Tiger Optimization Software (TOPS), a MATLAB-based PS optimal control software package, are described. TOPS is then used to solve several demonstrative problems, for which the indirect solutions are used for validation. Another goal of this effort is to pave the way for researchers who are interested in learning more about PS methods. More specifically, the software is written in a manner to make it as modular and documented as possible. This is done in the hope that future researchers will be able to easily understand the implementation of the PS features of TOPS and contribute to the development of the software by investigating the items listed in the "future work" section. In addition, TOPS is compatible with several popular NLP solvers to improve the capability of the software. It is the opinion and experience of the author that the best strategy to learn PS methods is through coded examples.

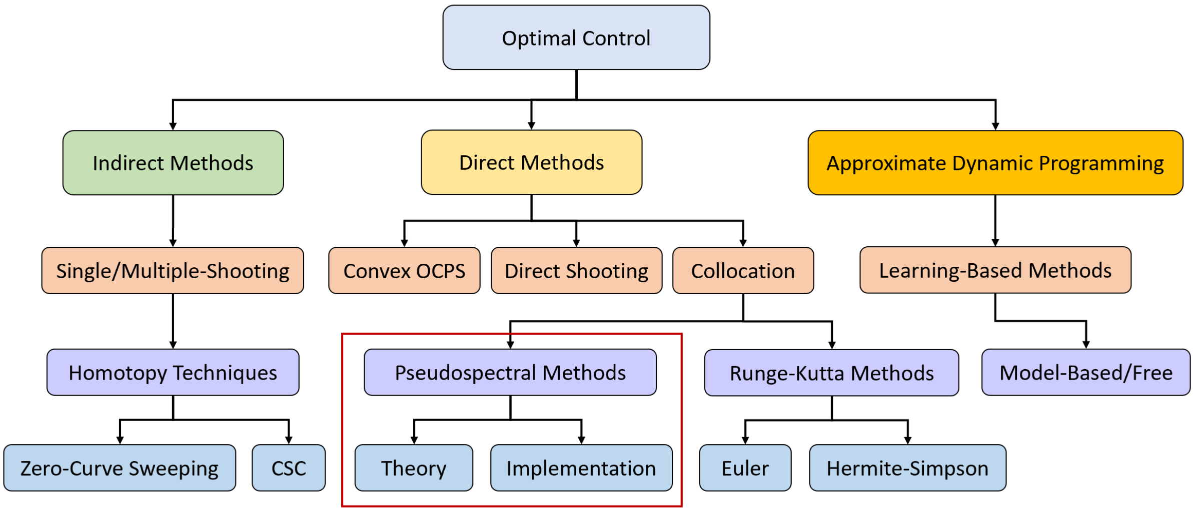


Figure 1.1: An overview of the solution methods for solving OCPs

Chapter 2

Overview of Indirect Methods

The field of indirect methods originated in the earliest attempts to solve OCPs. Queen Dido of Carthage was the first person to attempt a problem that can be solved using the calculus of variations, the cornerstone of the indirect method. Queen Dido was given a bull's hide and promised all the land she could enclose within it. Having cleverly cut it into strips and tied the ends together, her problem was to determine a suitable shape to enclose the maximum area. The calculus of variations can be used to prove that she should have chosen a circle [91]. Other prominent pioneers of the field include Sir Isaac Newton, Johann and Jacob Bernoulli (who first solved the Brachistochrone problem), L'Hospital, and of course, Lev Pontryagin, whose minimum principle revolutionized the study of the field [164].

2.1 The Calculus of Variations and the Euler-Lagrange Equations

The indirect method was originally built on functional analysis, the foundation of which is the calculus of variations (COV) . While traditional calculus allows us to find a point that minimizes or maximizes a function, the calculus of variations allows us to find a function that minimizes or maximizes a functional, which can be viewed as a "function of functions" that maps the results into a scalar value. The classic example of a functional is the integral operation,

$$J[\boldsymbol{x}(\cdot)] = \int_{t_0}^{t_f} L(\boldsymbol{x}(t))dt, \tag{2.1}$$

where $L = L(\boldsymbol{x}(t)) : \mathbb{R}^{N_x} \to \mathbb{R}$ is a nonlinear function that depends on $\boldsymbol{x}(t)$, the instantaneous value of $\boldsymbol{x}(\cdot)$ evaluated at t. The COV employs the concept of the *increment* of a functional, J, denoted as,

$$\Delta J(\boldsymbol{x}(t), \delta\boldsymbol{x}(t)) \hat{=} J(\boldsymbol{x}(t) + \delta\boldsymbol{x}(t)) - J(\boldsymbol{x}(t)) = \delta J(\boldsymbol{x}(t), \delta\boldsymbol{x}(t)) + g(\boldsymbol{x}(t), \delta\boldsymbol{x}(t)) \cdot \|\delta\boldsymbol{x}(t)\|, \tag{2.2}$$

where $\delta\boldsymbol{x}(t)$ denotes the *variation* of the function $\boldsymbol{x}(t)$. The fundamental theorem of the calculus of variations is,

Theorem 2.1 *If $\boldsymbol{x}^*(t)$ is an extremal, the variation of J, denoted as δJ, must vanish on $\boldsymbol{x}^*(t)$. That is,*

$$\delta J(\boldsymbol{x}^*(t), \delta\boldsymbol{x}(t)) = 0, \tag{2.3}$$

for all admissible $\delta\boldsymbol{x}(t)$.

Proofs of Theorem 2.1 can be found in [62, 198, 91]. As the focus of this study is direct methods, the derivations of the Euler-Lagrange equations will not be shown. However, rigorous application of this principle allows the first-order necessary conditions of optimality for constrained, multivariate functionals to be obtained. These are referred to as the Euler-Lagrange equations and are given as follows. Consider the functions below.

$$H(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{\lambda}(t), t) = F(\boldsymbol{x}(t), \boldsymbol{u}(t), t) + \boldsymbol{\lambda}^\top(t)\boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t), \tag{2.4}$$

$$\Phi(\boldsymbol{x}_f, t_f, \boldsymbol{\nu}) = E(\boldsymbol{x}_f, t_f) + \boldsymbol{\nu}^\top \boldsymbol{e}(\boldsymbol{x}_f, t_f). \tag{2.5}$$

Here, H is referred to as the *control Hamiltonian* and Φ (sometimes denoted as $\bar{E}$) is referred to as the *endpoint Lagrangian* [91] or the *augmented terminal cost* [171]. The terms $\boldsymbol{\lambda}(t)$ and $\boldsymbol{\nu}$ are Lagrange multipliers that are used to enforce the dynamics and boundary conditions, respectively. The *costates* are $\boldsymbol{\lambda}$ and the *endpoint covector* is called $\boldsymbol{\nu}$. The first-order necessary conditions of optimality (Euler-Lagrange equations) are given as,

$$\dot{\boldsymbol{x}}^*(t) = \frac{\partial H}{\partial \boldsymbol{\lambda}(t)} \left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{\lambda}^*(t), t\right), \tag{2.6}$$

$$\dot{\boldsymbol{\lambda}}^*(t) = -\frac{\partial H}{\partial \boldsymbol{x}(t)} \left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{\lambda}^*(t), t\right), \tag{2.7}$$

$$\boldsymbol{0} = \frac{\partial H}{\partial \boldsymbol{u}(t)} \left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{\lambda}^*(t), t\right), \tag{2.8}$$

and

$$\left[\frac{\partial \Phi}{\partial \boldsymbol{x}(t)} \left(\boldsymbol{x}_f^*, t_f\right) - \boldsymbol{\lambda}_f^*\right]^\top \delta \boldsymbol{x}_f + \left[H(\boldsymbol{x}_f^*, \boldsymbol{u}_f^*, \boldsymbol{\lambda}_f^*, t_f) + \frac{\partial \Phi}{\partial t} \left(\boldsymbol{x}_f^*, t_f\right)\right] \delta t_f = 0. \tag{2.9}$$

Eq. (2.7) is referred to as the *adjoint equation* or *costate dynamics*. Eq. (2.8) is referred to as the *strong form of optimality*. The entirety of Eq. (2.9) is referred to as the *transversality condition*. The bracketed terms in Eq. (2.9) only apply if their variational multiplier is *non-zero*. For example, if some boundary conditions are free (i.e., $\delta \boldsymbol{x}_f \neq \boldsymbol{0}$), then $\left[\frac{\partial \Phi}{\partial \boldsymbol{x}(t)} \left(\boldsymbol{x}_f, t_f\right) - \boldsymbol{\lambda}_f\right] = \boldsymbol{0}$. However, if boundary conditions are fixed, then the bracketed term should not be used. The same logic applies should $\delta t_f \neq 0$.

The difficulty of this problem is that it requires searching over infinite dimensional state spaces for $\boldsymbol{x}(t)$, $\boldsymbol{u}(t)$, and $\boldsymbol{\lambda}(t)$, which means that we are looking for the *entire* optimal time history of the state, costate, and control simultaneously. This problem is very difficult. Another difficulty that astute readers may have noticed lies in Eq. (2.8), which states that the optimal control law is obtained by setting the derivative of the Hamiltonian with respect to $\boldsymbol{u}$ equal to zero and solving for $\boldsymbol{u}$. If the Hamiltonian is linear with respect to control, it is impossible to determine the optimal control law, as in the derivative, the control vanishes. The solution to this problem is presented in the next section.

2.2 Pontryagin's Maximum (Minimum) Principle

Pontryagin's Minimum Principle (PMP) was developed by Lev Pontryagin and his students in the 1960s and published in 1987 [140], where it was referred to as the Maximum Principle due to a difference in sign of the Hamiltonian. At the heart of PMP is the Hamiltonian Mimization

Condition, shown below.

$$\boldsymbol{u}^*(t) \in \underset{\boldsymbol{u}(t)\in\mathbb{U}}{\arg\min}\, H(\boldsymbol{x}^*(t), \boldsymbol{u}(t), \boldsymbol{\lambda}^*(t), t), \tag{2.10}$$

where $\mathbb{U}$ denotes the admissible control set. Thus, the Minimum Principle states that minimizing the Hamiltonian only with respect to $\boldsymbol{u}$ while holding other terms static produces a candidate extremal control solution [140, 145, 111]. This converts the infinite-dimensional optimization problem into a finite-dimensional optimization problem. Rather than minimize over the entire function space, $\mathbb{U}$, we need only solve a parameter optimization of $\boldsymbol{u}$ at each time instant t. In addition, it also allows the derivation of the extremal control without using Eq. (2.8) and when the Hamiltonian is linear in control (except for singular control arcs [105, 108, 106]). Thus, the first order necessary conditions become,

$$\dot{\boldsymbol{x}}^*(t) = \frac{\partial H}{\partial \boldsymbol{\lambda}(t)} \left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{\lambda}^*(t), t\right), \tag{2.6}$$

$$\dot{\boldsymbol{\lambda}}^*(t) = -\frac{\partial H}{\partial \boldsymbol{x}(t)} \left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{\lambda}^*(t), t\right), \tag{2.7}$$

$$H\left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{\lambda}^*(t), t\right) \leq H\left(\boldsymbol{x}^*(t), \boldsymbol{u}(t), \boldsymbol{\lambda}^*(t), t\right), \tag{2.11}$$

for all $t \in [t_0, t_f]$,

and

$$\left[\frac{\partial \Phi}{\partial \boldsymbol{x}(t)} \left(\boldsymbol{x}_f^*, t_f\right) - \boldsymbol{\lambda}_f^*\right]^\top \delta \boldsymbol{x}_f + \left[H(\boldsymbol{x}_f^*, \boldsymbol{u}_f^*, \boldsymbol{\lambda}_f^*, t_f) + \frac{\partial \Phi}{\partial t} \left(\boldsymbol{x}_f^*, t_f\right)\right] \delta t_f = 0. \tag{2.9}$$

These are the first-order necessary conditions for optimality. Together, they form a two-point boundary value problem (TPBVP) that must be solved by a "guess and check" shooting method. In a shooting method, the optimal control law is derived using Eq. (2.8) or Eq. (2.10). This control law is a function of the state and costate values at time $t \in [t_0, t_f]$. Initial conditions for the states are typically known, and costate initial values have to be guessed. The state-costate differential equations are then propagated simultaneously to the final time. This is where the phrase "optimize then discretize" originates. The optimality conditions are formulated for the problem and are then satisfied at every discrete time instant stepped through by a fixed or variable step-size numerical integration scheme.

When applying Eq. (2.8) or Eq. (2.10), for simple cases, one obtains a control law, $\boldsymbol{u}^*(t) = \boldsymbol{u}^*(\boldsymbol{x}(t), \boldsymbol{\lambda}(t), t)$ [153, 28]. For more complex problems, such a closed-form expression may not exist and one has to solve a transcendental equation to obtain control [106]. The costates $\boldsymbol{\lambda}(t)$ that govern this control law have little intuitive physical meaning. Because of this, their initial values are *unknown* and must be guessed. Once the state and costate equations are propagated for the initial conditions, the final conditions are known and can be checked to see how well they satisfy the problem-specific boundary conditions and the transversality condition. Such a solution is called a candidate extremal solution. Only a locally or globally extremal solution will satisfy all the boundary conditions and the transversality condition. Thus, the solutions of indirect method are guaranteed to be local or global extremal. Gradient-descent solvers can take the initial costate guess, improve upon it, and re-propagate until a guess that satisfies the optimality conditions is found. However, these shooting schemes are *extremely* sensitive to the quality of the initial costate guess [111]. In addition to this, more Lagrange multipliers must be introduced when incorporating path constraints other than the dynamics. These multipliers must also be guessed. In fact, this is the primary shortcoming of indirect methods: there is no guarantee of convergence for an arbitrary initial costate guess [171].

2.3 Composite Smooth Control and Homotopy

A promising new technique that has been regularly used to reduce the "curse of sensitivity" is composite smooth control (CSC) [177] combined with continuation/homotopy [6]. Consider the problem of a minimum-fuel orbit transfer. This is a prevalent problem in the field of space mission design, and its solutions are incredibly useful for practical applications. The optimal control law for such a problem takes the following form:

$$T^* = \begin{cases} T_{\max}, & S(\boldsymbol{x}, \boldsymbol{\lambda}) > 0, \\ 0, & S(\boldsymbol{x}, \boldsymbol{\lambda}) < 0, \end{cases} \tag{2.12}$$

where $S(\boldsymbol{x}, \boldsymbol{\lambda})$ is a switching function obtained using PMP and $T_{\max}$ is the maximum allowable thrust. The value of S determines whether the thrust is on or off. It is evident from Eq. (2.12) that T^* is a discontinuous or "bang-bang" control. When using time-marching integrators such as a variable step-size Runge-Kutta method, absolute discontinuities such as this cause numerical difficulties and further increase sensitivity. The idea with CSC is to approximate such a control using smoothing functions, such as a Hyperbolic Tangent Smoothing (HTS) [168] or Heaviside step function [157, 9]. HTS functions take the form,

$$\delta(t) = \frac{1}{2}\left[1 + \tanh\left(\frac{S(\boldsymbol{x}(t), \boldsymbol{\lambda}(t))}{\rho}\right)\right], \tag{2.13}$$

where $\delta \in [0, 1]\ \forall\ S$. If we want to recover T, we simply take $T = T_{\max}\delta$, where δ may be considered the throttle. Here, S may be interpreted as a "distance measure" that defines the closeness of the event encoded in its value. This general notion of distance measures can be used to construct a number of composite functions that indicate whether a constraint or control should be active [166]. Thus, the added benefit of CSC is that these trigonometric approximations can be used to enforce the state and control path constraints given by Eq. (1.15) that are otherwise quite difficult to enforce using indirect methods [107]. The parameter ρ is referred to as the *continuation parameter*. Fig. 2.1 shows Eq. (2.13) for decreasing values of ρ for a

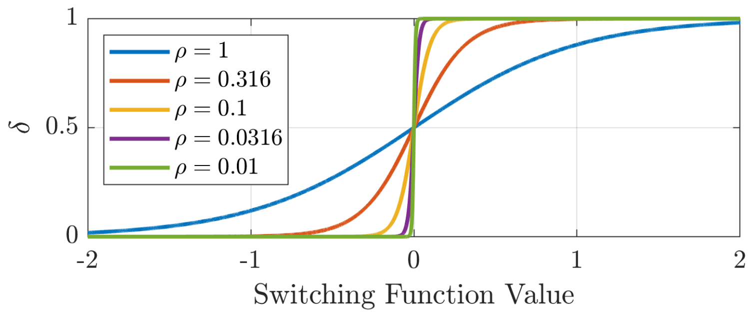


Figure 2.1: Eq. (2.13) for decreasing values of ρ

representative scenario where $S \in [-2, 2]$. It can be seen that as the continuation parameter decreases, the throttle more closely approximates the true discontinuous control *while remaining smooth and differentiable*. Solutions for low values of ρ will closely approximate the true solution to the problem. This is the key idea behind homotopy: the original problem is embedded in a family of neighboring problems using a smooth approximation of some aspect of the dynamics for which the smoothed problem is trivially solved. Since this smoothness can be varied using the continuation parameter, numerical continuation can be performed on its value. This means that once a smoothed problem converges, the solution is used to re-initialize the same problem for a reduced continuation parameter. Thus, solutions are swept across values of the continuation parameter until the original problem is recovered to the desired accuracy [175, 176]. Homotopy of this type can be performed on the entire dynamics model such that as the homotopy parameter is swept out, highly nonlinear terms are slowly introduced to a simplified dynamics model through numerical continuation. However, the existence of such a continuous *zero curve* such that the solutions of continuation steps are near one another is not guaranteed [125].

Recent works have made optimal control problems with discontinuous control profiles much more amenable to numerical treatments such as numerical homotopy and improved control smoothing/regularization techniques [14, 63, 139, 74, 200, 199, 30, 124, 202, 102, 169, 126, 170, 98, 127, 162, 128]. Methods for alleviating the difficulty associated with generating an initial costate guess have also been developed [174, 29, 84]. Such improved techniques have also inspired the development of various indirect methods that make practical low-thrust trajectory optimization problems easier to solve without the need for detailed derivation of oftentimes complicated necessary conditions. These methods have been applied to problems such as low-thrust electric propulsion (EP) transfers with the inclusion of operational constraints such as eclipsing [167, 161] and even the modeling of high-fidelity multi-mode EP systems [165]. Petukhov and Wook [136] presents a joint-optimization framework that uses the indirect method to implicitly optimize spacecraft hardware parameters (initial mass, thrust, and power) by elevating the thrust and power parameters to states in the system and deriving of a set of first-order necessary conditions that minimize propellant mass. Arya et al. [8, 7] present an indirect method for concurrent trajectory and high-fidelity thruster design by using the recently developed CSC [177] method to perform an optimization on the discrete operating modes of the electric thruster such that the optimal thrust and specific impulse time history may be retrieved at discrete values, which is consistent with the actual operation of EP systems.

To summarize, the primary advantage of indirect methods is the guarantee that a solution is a local extremal and high-resolution time history of states and controls are obtained. Additionally, for low-thrust trajectory optimization problems, indirect methods are often among the popular solution methods, as the computationally intensive parameter optimization methods used by direct transcription can prohibit their use [111]. The primary disadvantages of indirect methods include their extreme sensitivity to the unknown initial costate guess and the difficulty of incorporating path constraints. Additionally, indirect methods require the necessary conditions of optimality to be re-derived for each new problem. There are, however, recent progress in overcoming some of these issues. For instance, Mall and Taheri have developed an advanced trigonometric regularization method – unified trigonometrization method (UTM) – for solving problems with multiple state and control path constraints [104, 105, 103, 109, 106].

Chapter 3

Overview of Direct Methods

Direct methods are compelling alternatives to indirect methods, primarily due to their ease of use, the engineering accuracy of the solutions obtained using them, their large radius of convergence, and the ability for a class of direct methods (i.e., convex optimization) to be used for on-board automated guidance systems [111]. Direct methods first *discretize* Problem (B) into a parameter optimization problem using specialized techniques. The resulting NLP problem is then *optimized* using NLP solvers. The method used to discretize the problem is typically called a *transcription* technique. There are a multitude of different direct transcription schemes. In the general case, the resulting NLP problems are solved via a primal-dual interior point method (IPM) or Sequential Quadratic Programs (SQPs) [111]. Today, an extensive selection of software exists in nearly every programming language thanks to a large community that has been actively developing, improving, and researching numerical methods for the past 40 years [195, 55]. Due to this software ecosystem, the ease of use of direct methods, and their ability to "effortlessly" handle path constraints like Eq. (1.15), direct methods [39] can be considered to be among the most popular methods for solving OCPs [194].

In general, Problem (B) must be reduced into a finite-dimensional problem to be solved. This is achieved through the process of *discretization*, in which the continuous constraints of the problem are converted into finite-dimensional algebraic constraints. The time-horizon of the problem is divided into a finite number of segments, the endpoints of which are referred to as *mesh points* or *nodes*. A polynomial basis is then applied to approximate the state or control or both at the discrete nodes. This method was termed "direct transcription" by Canon et al. [26]. Methods that discretize only the state or control are called *sequential* methods, while those that discretize both are called *simultaneous* methods [88]. When the state is approximated using some polynomial, the differential algebraic constraints can be enforced using some derivative approximation or integration scheme specific to the polynomial basis. This is referred to as a *collocation* method, which is among the most popular direct methods [90].

3.1 Collocation Methods

To further clarify the meaning of "collocation," consider the differential (left) and integral forms (right) of the dynamics equations,

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{t}), \qquad \boldsymbol{x}(t) = \boldsymbol{x}(t_0) + \int_{t_0}^{t} \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t)dt. \tag{3.1}$$

In a collocation method, a polynomial basis is chosen to interpolate the state. Since polynomials may be differentiated or integrated, integration or differentiation rules may be constructed from these polynomials. Using these integration rules, the derivative or integral elements can be represented as,

$$\dot{\boldsymbol{x}}(t) \approx \sum_{i=1}^{N} \boldsymbol{d}_i \boldsymbol{x}_i(t), \qquad \int_{t_0}^{t} \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t)dt \approx \sum_{i=1}^{N} w_i \boldsymbol{f}(x_i(t), \boldsymbol{u}_i(t), t), \tag{3.2}$$

where $\boldsymbol{d}_i$ is some differentiation operator (a matrix value) and w_i is some quadrature weight (a scalar value). Further discussion will clarify how these can be obtained. However, for now, these elements *replace* their corresponding elements in Eq. (3.1) and we obtain,

$$\sum_{i=1}^{N} \boldsymbol{d}_i \boldsymbol{x}_i(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{t}) \qquad \boldsymbol{x}(t) = \boldsymbol{x}(t_0) + \sum_{i=1}^{N} w_i \boldsymbol{f}(x_i(t), \boldsymbol{u}_i(t), t). \tag{3.3}$$

The relations in Eq. (3.2) and Eq. (3.3) are then evaluated at the N support points $t_i \in [t_0, t_f]$, which are called *collocation points*. This is the process of collocation. Since both state and control are discrete values at the support points, collocation is a type of simultaneous direct method.

Collocation methods are now extremely mature and used extensively in the aerospace field. The distinctions between different direct approaches primarily lie in the method in which the integration or differentiation rules are constructed. The most common approaches use the trapezoid rule [16] or Hermite-Simpson Runge-Kutta methods [76]. The defect equation (used for enforcing the dynamics) for the trapezoidal method is,

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \frac{h_k}{2}\left[\boldsymbol{f}_k + \boldsymbol{f}_{k+1}\right], \quad \text{(Trapezoidal).} \tag{3.4}$$

The defect equation for the compressed Hermite-Simpson method is,

$$\begin{aligned}
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \tfrac{h_k}{6}\left[\boldsymbol{f}_k + 4\boldsymbol{f}_{k+\frac{1}{2}} + \boldsymbol{f}_{k+1}\right], \\
\boldsymbol{f}_{k+\frac{1}{2}} &= \boldsymbol{f}\left[\boldsymbol{x}_{k+\frac{1}{2}}, \boldsymbol{u}_{k+\frac{1}{2}}, t_k + \tfrac{h_k}{2}\right], \\
\boldsymbol{x}_{k+\frac{1}{2}} &= \tfrac{1}{2}\left(\boldsymbol{x}_{k+1} + \boldsymbol{x}_k\right) + \tfrac{h_k}{8}\left(\boldsymbol{f}_k - \boldsymbol{f}_{k+1}\right), \\
\boldsymbol{u}_{k+\frac{1}{2}} &= \tfrac{1}{2}\left(\boldsymbol{u}_{k+1} + \boldsymbol{u}_k\right).
\end{aligned} \tag{3.5}$$

In Eq. (3.5), $(k = 0, \ldots, N)$ where N is the number of collocation points and $\boldsymbol{f}_k$ is shorthand for $\boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k, t_k)$. The time step-size is the distance between the collocation points, $h_k = t_{k+1} - t_k$ [16].

One family of transcriptions that has become very popular in the last two decades is PS methods [12, 58, 60]. These methods use a basis of global orthogonal Lagrange or Chebychev polynomials to approximate the state and control variables [47, 49]. In addition to highly accurate solutions for a relatively sparse NLP problem, PS methods also allow for a mapping between the Karush-Kuhn-Tucker (KKT) multipliers used internally by many NLP solvers and the continuous time costates [49, 12, 58, 60]. This mapping produces a costate estimate that may be used to initialize indirect shooting schemes.

3.2 Other Direct Methods

There are many other less widely used but promising types of direct methods. A few will be briefly discussed here before moving on to PS methods.

One of the most promising new types of direct methods is convex optimization. Convex optimization methods have exploded in popularity in the optimization community due to the following fact: if a function is convex, global statements can be made from local function evaluations [111]. The strongest of these include a guarantee of finding a *global* optimum [143] and polynomial bounds on the maximum iteration count of an algorithm solving a convex problem [133]. As these types of methods are not the focus of this study, it is sufficient to define a set, $\mathcal{C} \subseteq \mathbb{R}^n$, to be convex if it contains every point in the line segment connecting any two of its points (i.e., a convex combination of the two pints). Similarly, a function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is convex if its domain is convex and it lies below the line segment connecting any two of its points [22, 195]. A convex discrete problem takes the form,

$$\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N), \;\; \boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{N-1}), \;\; t_f,$$

$$(\mathrm{P} : \mathrm{Convex}) \begin{cases} \text{Minimize} \quad J\left[\boldsymbol{X}, \boldsymbol{U}, t_f\right] = J(\boldsymbol{x}_N, t_f), \\ \text{Subject to :} \qquad \boldsymbol{x}_{k+1} = A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{d}_k, \;\; \forall \;\; (k = 1, \ldots, N-1), \\ \qquad \boldsymbol{e}(\boldsymbol{x}_1, \boldsymbol{x}_N) = \boldsymbol{0}, \\ \qquad \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, k) \leq \boldsymbol{0} \;\; \forall \;\; (k = 1, \ldots, N-1). \end{cases}$$

In $(\mathrm{P} : \mathrm{Convex})$, $J \in \mathbb{R}$ is the convex cost function, $\boldsymbol{e} \in \mathbb{R}^{N_e}$ is an affine function defining the boundary conditions, and $\boldsymbol{h} \in \mathbb{R}^{N_h}$ defines a convex set of path constraints. Problem $(\mathrm{P} : \mathrm{Convex})$ can be solved using efficient algorithms designed to solve convex problems [21]. It turns out that in many aerospace applications, such as spacecraft rendezvous and rocket landing [2], the dynamics can be expressed in the form shown in $(\mathrm{P} : \mathrm{Convex})$. The stipulation with convex optimization is that *the problem must be convex.* Many realistic problems are not convex. However, they can sometimes be convexified through a process of *lossless convexification* in which the original problem is reformulated in such a way that it remains the same, but is now convex [111]. However, this is not always possible, leading to the field of sequential convex programming (SCP) . SCP is based on the idea of solving a convex approximation of Problem

(B) through linearization. Each time this problem is solved, the approximated solution is updated. This continues as new solutions are obtained. This convex approximation is referred to as the subproblem [37]. Application of convex optimization for spacecraft low-thrust trajectory design are demonstrated in [189, 188, 80, 121]. In recent years, convex optimization and pseudospectral methods have been combined to solve powered descent and landing problems [155].

Another family of direct methods is differential dynamic programming (DDP) . DDP is also built on the idea of linearization, solving a discrete-time OCP through the use of an additive cost function. One attractive feature of the DDP algorithm is that it generates a decision rule (i.e., a control law) for the *entire state space* [196]. This means that a DDP algorithm need only be run *once* in order to obtain a closed-loop optimal control. Although it will not be discussed in detail, it has been used with great success for low-thrust long-duration trajectory optimization [191]. NASA Mystic software uses a variant of DDP for low-thrust trajectory optimization [93].

Chapter 4

The Theory of Pseudospectral Methods

Over the last two decades, PS optimal control has gained popularity. PS methods were originally derived from the *spectral* methods used to solve fluid dynamics problems [27]. Since their adaptation to optimal control, numerous theoretical results have been published [47, 49, 147, 51, 148, 12, 60, 145] for solving many different types of constrained optimization problems. This has led to numerous practical applications of PS methods, including flight maneuvers for NASA missions [11, 89], long-duration low-thrust orbit transfer maneuvers [81, 73, 70], and atmospheric flight optimization [116, 117] to name a few.

In a PS method, the state and control are both discretized while the state alone is parameterized using global orthogonal polynomials [58]. Differentiation and quadrature rules derived from these polynomials and their support points are used to discretize Problem (B) and solve it using an NLP solver. While PS methods do have an indirect form [48], it is not used in practice.

This chapter will focus exclusively on the theory and implementation of direct PS methods. The details of several PS schemes will be presented and their strengths and weaknesses will be considered. Mesh refinement schemes will be presented and discussed, as well as practical methods to calculate derivatives of the NLP resulting from a PS transcription. Scaling methods and their usefulness will be discussed. The elements of this section that are implemented in TOPS will also be highlighted.

4.1 Orthogonal Polynomials and Legendre-Gauss Pseudospectral Methods

Although the title to this section makes reference to a popular flavor of PS methods, a PS method can be derived making no assumptions as to the type of discretization being used.

There are four elements to a PS method: domain transformation, interpolation, differentiation, and integration [149].

4.1.1 Domain Transformation

The first step to any PS method is to map the physical time domain, $t \in [t_0, t_f]$ to the scaled computational domain, $\tau \in [-1, 1]$. This is done so that specific types of interpolating polynomials can be used. The transformation can be generated as follows.

$$t_0 \le t \le t_f \;\rightarrow\; 0 \le t - t_0 \le t_f - t_0 \;\rightarrow\; 0 \le \frac{t - t_0}{t_f - t_0} \le 1,$$
$$\rightarrow\; 0 \le 2\frac{t - t_0}{t_f - t_0} \le 2 \;\rightarrow\; -1 \le 2\frac{t - t_0}{t_f - t_0} - 1 \le 1,$$

Here, we take

$$\tau(t, t_0, t_f) = 2\frac{t - t_0}{t_f - t_0} - 1, \tag{4.1}$$

which can be inverted to produce

$$t(\tau, t_0, t_f) = \frac{t_f - t_0}{2}\tau + \frac{t_f + t_0}{2}. \tag{4.2}$$

This transformation will be used in the following derivations to scale the generalized problem to the interval $[-1, 1]$. It is also helpful to define the following derivative.

$$\frac{dt}{d\tau} = \frac{t_f - t_0}{2}, \tag{4.3}$$

Eq. (4.3) can be used to scale derivatives to the interval $\tau \in [-1, 1]$. Using this information, we may re-express Problem (B) as,

$$
\boldsymbol{x}(t) \in \mathbb{R}^{N_x}, \;\; \boldsymbol{u}(t) \in \mathbb{R}^{N_u}, \;\; t_0 \in \mathbb{R}, \;\; t_f \in \mathbb{R},
$$

$$
(\mathrm{B}_1) \begin{cases} \text{Minimize} & J\left[\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot), t_0, t_f\right] = E(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f) + \displaystyle\int_{t_0}^{t_f} F\left[\boldsymbol{x}(t), \boldsymbol{u}(t), t\right] dt, \\ \text{Subject to :} & \dfrac{d\boldsymbol{x}(\tau)}{d\tau} = \dfrac{t_f - t_0}{2} \boldsymbol{f}(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \tau), \\ & \boldsymbol{e}(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f) \leq \mathbf{0}, \\ & \boldsymbol{h}(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \tau) \leq \mathbf{0}. \end{cases}
$$

Note in Problem (B_1), the running cost F is not yet transformed to the interval $[-1, 1]$. This will be done shortly. Also note that this transformation assumes a finite time horizon. For an infinite time horizon, the transformation takes the form [58],

$$
t(\tau) = \frac{1 + \tau}{1 - \tau}. \tag{4.4}
$$

This study will assume a finite time horizon formulation of the OCPs.

4.1.2 Interpolation

At the heart of PS methods are the Lagrange interpolating polynomials. First, let $\pi^N := \{\tau_1, \tau_2, \ldots, \tau_N\}$ be a set of distinct points on the interval $[-1, 1]$. Let $\{\ell_1(\tau), \ell_2(\tau), \ldots, \ell_N(\tau)\}$ be a set of Lagrange *basis* polynomials given by,

$$
\ell_j(\tau) = \prod_{\substack{i=1 \\ i \neq j}}^{N} \frac{\tau - \tau_i}{\tau_j - \tau_i}, \quad (j = 1, \ldots, N). \tag{4.5}
$$

The Lagrange basis polynomials $\ell_j(\tau_i)$, $(i, j = 1, \ldots, N)$ satisfy the following property,

$$
\ell_j(\tau_i) = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases} \tag{4.6}
$$

Using the Kronecker delta, this relationship can be re-expressed as $\ell_j(\tau_i) = \delta_{ji}$. Next, we interpolate the k-th component of the state using a weighted interpolant of the form,

$$x_k(\tau) \approx \sum_{j=1}^{N} \frac{W(\tau)}{W(\tau_j)} x_{jk} \ell_j(\tau), \quad (k = 1, \ldots, N_x), \tag{4.7}$$

where W is a positive weight function [149]. Choosing the simplest case of $W(\tau) = 1$, we obtain,

$$x_k(\tau) \approx \sum_{j=1}^{N} x_{jk} \ell_j(\tau), \quad (k = 1, \ldots, N_x). \tag{4.8}$$

Eq. (4.8) is the classical Lagrange *interpolating* polynomial. Note that since the interpolating polynomial satisfies the Kronecker condition, the polynomial coefficients x_{jk} are *exactly* equal to the continuous function value evaluated at the corresponding support points. In other words,

$$x_{jk} = x_k(\tau_j), \tag{4.9}$$

where $(j = 1, \ldots, N)$ and $(k = 1, \ldots, N_x)$. This feature is an extremely important aspect of PS methods [48].

4.1.3 Differentiation

Now that the state is approximated by a continuous interpolating polynomial, we can approximate its derivative. Differentiating Eq. (4.8) with respect to τ we obtain,

$$\dot{x}_k(\tau) \approx \sum_{j=1}^{N} x_{jk} \dot{\ell}_j(\tau), \quad (k = 1, \ldots, N_x), \tag{4.10}$$

since x_{jk} are constant coefficients. Note that we denote,

$$\dot{x}(\tau) = \frac{dx(\tau)}{d\tau}, \tag{4.11}$$

but when necessary, the dot notation may be interchanged with Leibniz derivative notation for clarity. Next, let,

$$D_{ij} = \dot{\ell}_j(\tau_i), \tag{4.12}$$

where D_{ij} is called the *differentiation matrix*. In matrix form, the state and differentiation matrices can be expressed as,

$$\boldsymbol{D} = \begin{bmatrix} \dot{\ell}_1(\tau_1) & \dot{\ell}_2(\tau_1) & \cdots & \dot{\ell}_N(\tau_1) \\ \dot{\ell}_1(\tau_2) & \dot{\ell}_2(\tau_2) & \cdots & \dot{\ell}_N(\tau_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{\ell}_1(\tau_N) & \dot{\ell}_2(\tau_N) & \cdots & \dot{\ell}_N(\tau_N) \end{bmatrix}, \quad \boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix}, \tag{4.13}$$

where $\boldsymbol{D} \in \mathbb{R}^{N\times N}$, $\boldsymbol{X} \in \mathbb{R}^{N\times N_x}$, and $\boldsymbol{x}_i \in \mathbb{R}^{N_x}$ are *row* vectors corresponding to the state vector at time t_i. Using these matrices, we may write,

$$\dot{x}_k(\tau_i) = [\boldsymbol{D}\boldsymbol{X}]_{ik}\,. \tag{4.14}$$

Equation (4.14) approximates the left-hand side of the dynamics, $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t)$, during the collocation process. Next, (B_1) is re-expressed as,

$$\boldsymbol{X} \in \mathbb{R}^{N\times N_x},\ \ \boldsymbol{U} \in \mathbb{R}^{N\times N_u},\ \ \boldsymbol{x}(t) \in \mathbb{R}^{N_x},\ \ \boldsymbol{u}(t) \in \mathbb{R}^{N_u},\ \ t_0 \in \mathbb{R},\ \ t_f \in \mathbb{R},$$

$$(\mathrm{B}_2)\begin{cases} \text{Minimize} \quad J[\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot), t_0, t_f] = E(\boldsymbol{x}_1, \boldsymbol{x}_N, t_0, t_f) \\ \qquad\qquad\qquad + \displaystyle\int_{t_0}^{t_f} F[\boldsymbol{x}(t), \boldsymbol{u}(t), t(\tau, t_0, t_f)]\, dt, \\ \text{Subject to :} \qquad \boldsymbol{D}\boldsymbol{X} = \dfrac{t_f - t_0}{2}\boldsymbol{f}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}(\boldsymbol{\tau}, t_0, t_f)), \\ \qquad \boldsymbol{e}(\boldsymbol{x}_1, \boldsymbol{x}_N, t_0, t_f) \leq \boldsymbol{0}, \\ \qquad \boldsymbol{h}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}(\boldsymbol{\tau}, t_0, t_f)) \leq \boldsymbol{0}, \end{cases}$$

where we have defined two matrices according to,

$$\boldsymbol{U} \in \mathbb{R}^{N\times N_u} = \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{u}_N \end{bmatrix}, \qquad \boldsymbol{t} \in \mathbb{R}^{N} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix},$$

in which $\boldsymbol{u}_i \in \mathbb{R}^{N_u}$ are *row* vectors. Additionally, t_i, $(i = 1, 2, \ldots, N)$ are the time values associated with the collocation points. A vector $\boldsymbol{\tau}$ may be constructed in an identical fashion to $\boldsymbol{t}$, and the expression $\boldsymbol{t}(\boldsymbol{\tau}, t_0, t_f)$ is given by Eq. (4.2). Note that we have not yet fully discretized the domain, since the running cost is still a function of continuous functions and is not discretized/approximated. This will be remedied shortly.

4.1.4 Integration

Since we are operating on the domain $\tau \in [-1, 1]$, we may apply weighted quadrature to approximate the integral. Consider the arbitrary function $g(t)$ defined in $[t_0, t_f]$ being integrated over the same interval. In order to interpolate this function using a basis of weighted Lagrange polynomials, we must transform this integral to the interval $[-1, 1]$. Rearranging Eq. (4.3) as,

$$dt = \frac{t_f - t_0}{2} d\tau, \tag{4.15}$$

we may write,

$$\int_{t_0}^{t_f} g(t) dt = \frac{t_f - t_0}{2} \int_{-1}^{1} g\left[t(\tau, t_0, t_f)\right] d\tau. \tag{4.16}$$

Note that in Eq. (4.16) we have changed the argument of $g(t)$ to $g\left[t(\tau, t_0, t_f)\right]$. This is done for two reasons. The first is that we are making no assumptions on the operations performed by $g(t)$. They may be dependent on the domain of $g(t)$ being $[t_0, t_f]$. Second, we do not know that $g(t)$ is defined on $[-1, 1]$, so when evaluating g, we simply replace its argument with the affine transformation given in Eq. (4.2). Next, we may write

$$\frac{t_f - t_0}{2} \int_{-1}^{1} g\left[t(\tau, t_0, t_f)\right] d\tau = \frac{t_f - t_0}{2} \sum_{j=1}^{N} \int_{-1}^{1} \frac{W(\tau)}{W(\tau_j)} g_j \ell_j(\tau) d\tau, \tag{4.17}$$

where $(j = 1, \ldots, N)$. Since g_j are constant coefficients, we may factor them out of the integral. Thus, we obtain,

$$\frac{t_f - t_0}{2} \int_{-1}^{1} g\left[t(\tau, t_0, t_f)\right] d\tau = \frac{t_f - t_0}{2} \sum_{j=1}^{N} w_j g_j, \tag{4.18}$$

where w_j are referred to as the quadrature weights and are given by,

$$w_j = \int_{-1}^{1} \frac{W(\tau)}{W(\tau_j)} \ell_j(\tau) d\tau. \tag{4.19}$$

Eq. (4.19) is a generalized expression for the quadrature weights. For specific transcription methods, explicit formulas are given to compute these weights.

4.1.5 Gaussian Quadrature Points and the Pseudospectral Bolza Problem

In the discussion above, we have considered an arbitrary grid of support points or "nodes." In practice, the placement of these nodes is incredibly important and is an important step in implementing PS methods. Davis and Rabinowitz [35] showed that the best selection of grid points for interpolation, integration, and differentiation of functions is a grid of Gaussian quadrature points. By "best," the author means that for N nodes, a Gaussian distribution of those nodes will minimize the approximation error present in the aforementioned operations. In fact, Gaussian points greatly reduce the Runge phenomenon that appears for an even distribution of points [51]. All Gaussian quadrature points lie in the domain $[-1, 1]$ and are more densely packed around the endpoints of the interval.

In the following sections, several NLPs will be formulated using Legendre-Gauss transcriptions. Additionally, a Chebyshev-Gauss discretization will be briefly shown. All Legendre-Gauss points can be obtained from the roots of the Legendre polynomials and/or linear combinations of the polynomials and their derivatives [61] while the Chebyshev-Gauss points can be obtained from the extrema of the Chebyshev polynomials [50].

Now that we have formulated all the necessary elements, we may use them to generate a PS discretization of Problem (B). This is presented below.

$$
(\mathrm{B}_{\mathrm{PS}}) \begin{cases} & \boldsymbol{X} \in \mathbb{R}^{N \times N_x}, \;\; \boldsymbol{U} \in \mathbb{R}^{N \times N_u}, \;\; t_0 \in \mathbb{R}, \;\; t_f \in \mathbb{R}, \\ \text{Minimize} & J\left[\boldsymbol{X}, \boldsymbol{U}, t_0, t_f\right] = E(\boldsymbol{x}_1, \boldsymbol{x}_N, t_0, t_f) + \dfrac{t_f - t_0}{2} \displaystyle\sum_{i=1}^{N} w_i F(\boldsymbol{X}_i, \boldsymbol{U}_i, t_i), \\ \text{Subject to :} & \boldsymbol{D}\boldsymbol{X} = \dfrac{t_f - t_0}{2} \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}), \\ & \boldsymbol{e}(\boldsymbol{x}_1, \boldsymbol{x}_N, t_0, t_f) \leq \boldsymbol{0}, \\ & \boldsymbol{h}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}) \leq \boldsymbol{0}. \end{cases}
$$

Problem $(\mathrm{B}_{\mathrm{PS}})$ can be solved to obtain the discrete values of the states and controls at the N collocation points. To recover the continuous states, we can use Eq. (4.8) and *arbitrarily* interpolate $\boldsymbol{U}$. Note that we abuse notation in $(\mathrm{B}_{\mathrm{PS}})$, as every argument t is really $t(\tau, t_0, t_f)$ given by Eq. (4.2). However, the dependence is dropped for the sake of notational convenience.

Although we have presented the dynamic constraints of Problem $(\mathrm{B}_{\mathrm{PS}})$ in a matrix form, it is oftentimes easier to first formulate the PS problem using summations rather than matrix multiplications and convert to matrix notation afterwards. Some authors prefer this notation [101]. For an N_x-dimensional state, this takes the form,

$$
\dot{\boldsymbol{x}}(\tau_j) \approx \dot{\boldsymbol{X}}(\tau_j) = \sum_{j=1}^{N} D_{ij} \boldsymbol{X}_j = \boldsymbol{D}\boldsymbol{X} = \frac{t_f - t_0}{2} \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{\tau}), \quad (i = 1, \ldots, N), \tag{4.20}
$$

where we remind the reader that $\boldsymbol{X}_j$ denotes the j-th *row* of the matrix $\boldsymbol{X} \in \mathbb{R}^{N \times N_x}$.

Now that we have formulated a general PS discretization of Problem (B), we may begin discussing the aforementioned variants and their differences. A methodical approach will be taken for each variant in order to more clearly contrast each method. For the sake of clarity, a *single-interval* formulation will be considered for each variant. However, a multi-interval formulation will be shown for the LGR method implemented in TOPS. In each method, N will always denote the number of *collocation* points. This is because some schemes employ non-collocated discretization points.

4.2 Legendre-Gauss-Lobatto Points

The first set of points that will be discussed are the Legendre-Gauss-Lobatto (LGL) points. These are the most commonly used set of LG points and, in the opinion of the author, the easiest to implement. The LGL method is also the most well-understood of the Legendre-Gauss methods as it has been studied the longest [45, 61] and convergence proofs for the control have been developed by Ross and Karpenko [149]. The optimal control software DIDO uses LGL points [146], but it does not use the exact transcription shown in this thesis.

Consider collocation at N LGL points. The LGL points are obtained as [1],

$$\text{LGL} \quad \Rightarrow \text{the roots of } \dot{P}_{N-1}(\tau) \text{ together with the points } -1 \text{ and } 1.$$

Here, $P_N(\tau)$ is the N-th degree Legendre polynomial. Thus, $\dot{P}_{N-1}(\tau)$ is the derivative of the N-th degree Legendre polynomial with respect to τ. Note that quadrature approximations using the LGL points are exact for polynomials of degree up to $2N-3$ [59]. There is no closed-form expression for the LGL points, but they can be calculated numerically to machine precision [27]. Links to download such algorithms for specific transcriptions can be found in the preface of Shen et al. [159]. Additionally, numerous helpful PS algorithms are provided by Greg von Winckel at the MATLAB file exchange [183]. Fig. 4.1 shows a schematic of the

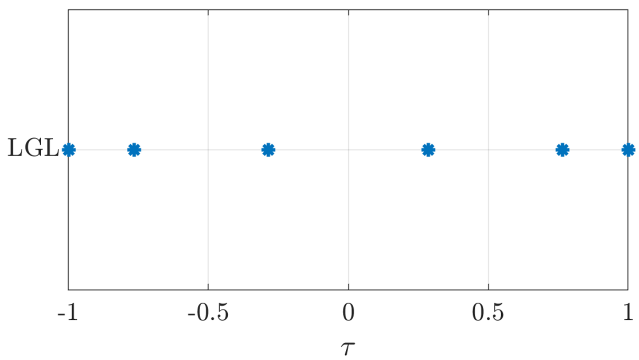


Figure 4.1: The LGL Points

distribution of six LGL points. Note that the LGL points lie in the closed interval $\tau \in [-1, 1]$.

The quadrature weights for the LGL transcription are given as [51, 159],

$$w_i^{LGL} = \frac{2}{M(M+1)\left[P_M(\tau_i)\right]^2}, \quad (i = 0, 1, \ldots, M), \tag{4.21}$$

and the differentiation matrix can be calculated as

$$D_{ij}^{LGL} = \begin{cases} -\dfrac{M(M+1)}{4}, & i = j = M, \\ \dfrac{P_M(\tau_i)}{P_M(\tau_J)} \dfrac{1}{(\tau_i - \tau_j)}, & i \neq j,\ 1 \leq i, j \leq M, \\ 0, & 2 \leq i = j \leq M-1, \\ \dfrac{M(M+1)}{4}, & i = j = M. \end{cases} \tag{4.22}$$

Note the indices of Eqs. (4.21) and (4.22) differ slightly from our notation, since we use N to denote the number of collocation points. Since the indices are $(i, j = 0, \ldots, M)$, set $M \leftarrow N-1$ to calculate weight and differentiation matrices of appropriate sizes. An algorithm that calculates PS differentiation matrices for an arbitrary grid can be found in [185]. Using Eq. (4.22), the state derivative becomes,

$$\dot{\boldsymbol{x}}(\tau_j) \approx \sum_{j=1}^{N} D_{ij}^{LGL} \boldsymbol{X}_j = \boldsymbol{D}^{LGL} \boldsymbol{X}^{LGL}, \tag{4.23}$$

where $(i = 1, 2 \ldots, N)$. Now, we may re-write Problem $(\mathrm{B_{PS}})$ as,

$$\boldsymbol{X} = \boldsymbol{X}^{LGL} \in \mathbb{R}^{N \times N_x},\ \boldsymbol{U} = \boldsymbol{U}^{LGL} \in \mathbb{R}^{N \times N_u},\ t_0 \in \mathbb{R},\ t_f \in \mathbb{R},$$

$$(\mathrm{B_{LGL}}) \begin{cases} \text{Minimize} \quad J\left[\boldsymbol{X}, \boldsymbol{U}, t_0, t_f\right] = E(\boldsymbol{x}_1, \boldsymbol{x}_N, t_0, t_f) \\ \qquad\qquad + \dfrac{t_f - t_0}{2} \displaystyle\sum_{i=1}^{N} w_i^{LGL} F(\boldsymbol{X}_i, \boldsymbol{U}_i, t_i), \\ \text{Subject to :} \quad \boldsymbol{D}^{LGL} \boldsymbol{X} = \dfrac{t_f - t_0}{2} \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}^{LGL}), \\ \qquad \boldsymbol{e}(\boldsymbol{x}_1, \boldsymbol{x}_N, t_0, t_f) \leq \boldsymbol{0}, \\ \qquad \boldsymbol{h}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}^{LGL}) \leq \boldsymbol{0}. \end{cases}$$

In Problem $(\mathrm{B}_{\mathrm{LGL}})$, we define,

$$\boldsymbol{t}^{LGL} = [t_1, t_2, \ldots, t_N]^\top, \tag{4.24}$$

to be the time points in the domain $[-1, 1]$ associated with the LGL points obtained using Eq. (4.2). This completes the *single-interval* LGL PS transcription.

A special feature of PS transcriptions is that they offer a mapping between the discrete-time Karush-Kuhn-Tucker multipliers used internally by NLP solvers and the continuous-time Lagrange multipliers (costates). This is referred to as the covector mapping theorem [49, 51, 60, 67, 69]. It is an incredibly powerful feature of PS methods and unifies direct and indirect PS methods. First, we define

$$\boldsymbol{\lambda}_{1:N} = \begin{bmatrix} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \\ \vdots \\ \boldsymbol{\lambda}_N \end{bmatrix}, \quad \boldsymbol{\Lambda}_{1:N} = \begin{bmatrix} \boldsymbol{\Lambda}_1 \\ \boldsymbol{\Lambda}_2 \\ \vdots \\ \boldsymbol{\Lambda}_N \end{bmatrix}, \quad \boldsymbol{\gamma}_{1:N} = \begin{bmatrix} \boldsymbol{\gamma}_1 \\ \boldsymbol{\gamma}_2 \\ \vdots \\ \boldsymbol{\gamma}_N \end{bmatrix}, \quad \boldsymbol{\Gamma}_{1:N} = \begin{bmatrix} \boldsymbol{\Gamma}_1 \\ \boldsymbol{\Gamma}_2 \\ \vdots \\ \boldsymbol{\Gamma}_N \end{bmatrix}. \tag{4.25}$$

In Eq. (4.25), $\boldsymbol{\lambda}_{1:N} \in \mathbb{R}^{N \times N_x}$ are the continuous-time costates evaluated at the collocation points and $\boldsymbol{\Lambda}_{1:N} \in \mathbb{R}^{N \times N_x}$ are their discrete time KKT counterparts. Similarly, $\boldsymbol{\gamma}_{1:N} \in \mathbb{R}^{N \times N_h}$ are the continuous-time multipliers associated with the path constraints, $\boldsymbol{h}$, and $\boldsymbol{\Gamma}_{1:N} \in \mathbb{R}^{N \times N_h}$ are their discrete-time KKT counterparts. The mapping between them is given below.

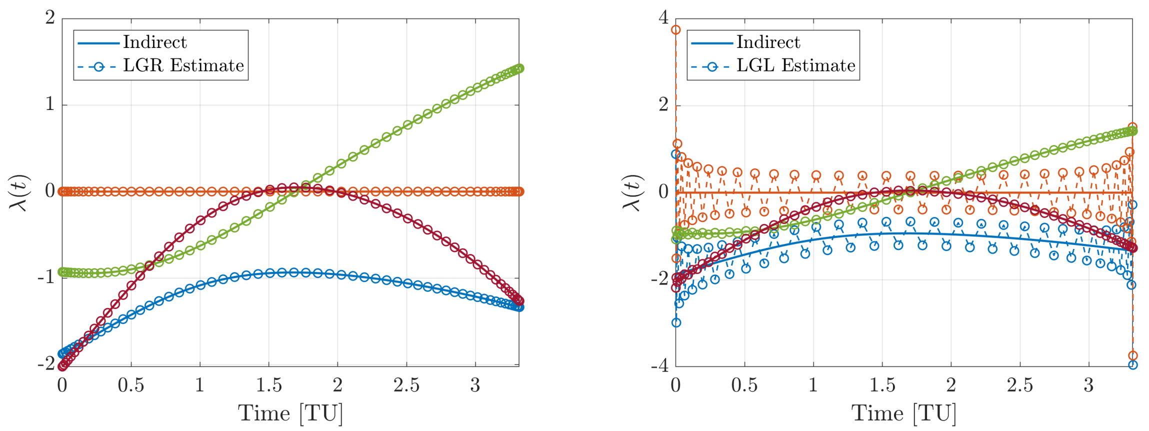
$$\boldsymbol{\lambda}_i^{LGL} = \frac{\boldsymbol{\Lambda}_i^{LGL}}{w_i^{LGL}}, \quad \boldsymbol{\gamma}_i^{LGL} = \frac{\boldsymbol{\Gamma}_i^{LGL}}{w_i^{LGL}}, \quad (i = 1, \ldots, N). \tag{4.26}$$

TOPS offers an option to use a multi-interval LGL transcription. See Appendix A for instructions on how to use this transcription in TOPS.

4.2.1 LGL Costate Estimate Inaccuracies

It is important to note that the costates obtained using the Lobatto transcription are noisy and exhibit oscillatory behavior about their optimal value [49]. This oscillation is due to a null-space in the matrix linear system of the LGL dynamic constraint [59] that allows for infinitely many solutions to be obtained for $\mathbf{\Lambda}^{LGL}$. As such, they must be filtered to obtain a reasonable approximation of the continuous-time costates. The LGR and Legendre-Gauss (LG) transcriptions do **not** exhibit this oscillatory behavior [59].

Consider the well-known orbit-raising problem posited by Bryson and Ho [23], stated fully in Section 6.2. This problem has been used as an optimal control example extensively in the literature. A solution was obtained using an indirect method to obtain the exact values of the costates. Next, TOPS was used to solve the problem using an LGR and then an LGL transcription. A global polynomial (one mesh segment) was used containing 60 collocation points for both cases. The state and control solutions were nearly identical using the LGR and LGL transcriptions. However, Fig. 4.2 shows a comparison of the costate estimates obtained using the two transcriptions. In Fig. 4.2, the dashed lines with markers represent the costate



(a) LGR Costate Estimate (b) LGL Costate Estimate

Figure 4.2: Costate Estimates for Orbit Raising Problem Using LGR and LGL Transcriptions

values obtained using a PS transcription, while the solid lines represent the indirect costates obtained using a single-shooting scheme. In Fig. 4.2a, the LGR estimate of the costates lies exactly on top of the indirect costates. However, in Fig. 4.2b, the LGL estimates of the costates exhibit varying degrees of oscillations about the true value. The exact estimate provided by the

LGR transcription is the primary reason TOPS defaults to an LGR scheme. Since the author of TOPS and researchers in the ACE lab use indirect methods frequently, the ability to obtain high-quality costate estimates is useful for initializing indirect single- or multiple-shooting methods to obtain high-resolution solutions to difficult problems using advanced regularized indirect methods [177].

4.3 Legendre-Gauss-Radau Points

The next set of points that will be discussed are the Legendre-Gauss-Radau points. These are less commonly used than LGL points and are much more "tricky" to implement than LGL points due to their asymmetry about the origin in the interval $[-1, 1]$. **These are the primary set of Gauss points implemented in TOPS**. Historically, the LGR points have been the least studied or used set of Guass points presented here [61]. However, this has changed in recent years, largely due to the efforts of Dr. Anil Rao and his collaborators [60] at the University of Florida. It is worth noting that for both LGR and LG points, no formal proof of control convergence has been produced. For this reason, some influential members of the PS optimal control field disapprove of their use [52]. However, in practice, Radau points exhibit an exponential convergence rate for smooth problems and proofs of costate convergence do exist [59, 87, 99].

Consider collocation at N LGR points. The LGR points are obtained as [159, 59],

$$\text{LGR} \quad \Rightarrow \text{the roots of } P_{N-1}(\tau) + P_N(\tau).$$

where $P_N(\tau)$ is the N-th degree Legendre polynomial. Note that quadrature approximations using the LGR points are exact for polynomials of degree up to $2N - 2$ [59]. Like the LGL points, there is no closed-form expression to obtain the LGR points. Algorithms that can calculate them to machine precision can be found at [184]. Fig. 4.3 shows a schematic of the distribution of six LGR and LGL points. Note that the LGR points lie in the half open interval $\tau \in [-1, 1)$ while the LGL points lie in the closed interval $\tau \in [-1, 1]$. The quadrature weights

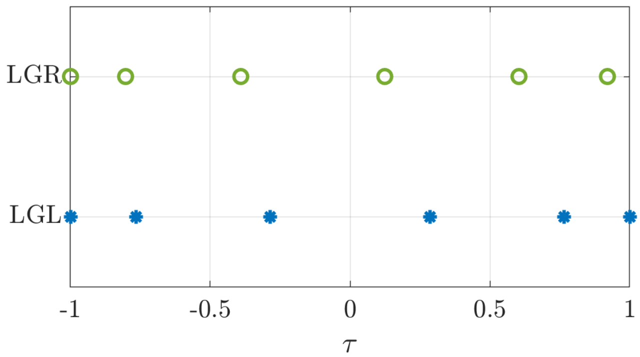


Figure 4.3: The LGL and LGR Points

for the LGR transcription are given as [159, 51],

$$w_i^{LGR} = \frac{1-\tau_i}{(M+1)^2\left[P_M(\tau_i)\right]^2}, \quad (i = 0, 1, \ldots, M), \tag{4.27}$$

and the differentiation matrix can be calculated as,

$$D_{ij}^{LGR} = \begin{cases} -\dfrac{M(M+2)}{4}, & i = j = 0, \\ \dfrac{P_M(\tau_i)}{P_M(\tau_j)} \dfrac{(1-\tau_j)}{(1-\tau_i)} \dfrac{1}{(\tau_i - \tau_j)} & i \neq j,\ 0 \leq i, j \leq M, \\ \dfrac{1}{2(1-\tau_i)}, & 1 \leq i = j \leq M. \end{cases} \tag{4.28}$$

In Eq. (4.27) and Eq. (4.28), the indices differ from our notation once again. We let $M \leftarrow N-1$ to calculate the weights and differentation matrix for N collocation points. Again, a MATLAB-based algorithm to calculate this differentiation matrix can be found at [185]. However, we are not yet finished with this transcription. The reader may have noticed that in Fig. 4.3 the set of LGR points do *not* include the point $\tau = 1$. Thus, if the transcription remains unchanged, endpoint boundary conditions cannot be enforced. As in [60], we introduce a *non-collocated* point $\tau_{N+1} = 1$, which is used to approximate the *state only* at the final time. Thus, the approximation of the state derivative becomes,

$$\dot{\boldsymbol{x}}(\tau_j) \approx \sum_{j=1}^{N+1} D_{ij}^{LGR} \boldsymbol{X}_j = \boldsymbol{D}^{LGR} \boldsymbol{X}^{LGR}, \tag{4.29}$$

where $(i = 1, 2, \ldots, N)$ and $\boldsymbol{D}^{LGR}$ is the N by $N+1$ **rectangular** LGR differentiation matrix. This extra column is associated with the Lagrange polynomial $\ell_{N+1}(\tau)$. To obtain this matrix using Eq. (4.28), evaluate the formula (or provided algorithm) for $N+1$ points and then delete the $(N+1)$-th *row* of the resulting matrix. We may denote the non-collocated discretization point as $\boldsymbol{x}_f \in \mathbb{R}^{N_x}$, a row vector. Thus, we may write,

$$\boldsymbol{X} = \boldsymbol{X}^{LGR} \in \mathbb{R}^{N \times N_x}, \;\; \boldsymbol{x}_f \in \mathbb{R}^{N_x} \;\; \boldsymbol{U} = \boldsymbol{U}^{LGR} \in \mathbb{R}^{N \times N_u}, \;\; t_0 \in \mathbb{R}, \;\; t_f \in \mathbb{R},$$

$$(\mathrm{B_{LGR}}) \begin{cases} \text{Minimize} & J\left[\boldsymbol{X}, \boldsymbol{x}_f, \boldsymbol{U}, t_0, t_f\right] = E(\boldsymbol{x}_1, \boldsymbol{x}_f, t_0, t_f) \\ & \qquad + \dfrac{t_f - t_0}{2} \displaystyle\sum_{i=1}^{N} w_i^{LGR} F(\boldsymbol{X}_i, \boldsymbol{U}_i, t_i), \\ \text{Subject to :} & \boldsymbol{D}^{LGR} \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{x}_f \end{bmatrix} = \dfrac{t_f - t_0}{2} \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}^{LGR}), \\ & \boldsymbol{e}(\boldsymbol{x}_1, \boldsymbol{x}_f, t_0, t_f) \leq \boldsymbol{0}, \\ & \boldsymbol{h}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}^{LGR}) \leq \boldsymbol{0}. \end{cases}$$

In Problem $(\mathrm{B_{LGR}})$ we define,

$$\boldsymbol{t}^{LGR} = [t_1, t_2, \ldots, t_N]^\top, \tag{4.30}$$

to be the transformed time points in the domain $[-1, 1)$ associated with the LGR points *only*. They are obtained using Eq. (4.2). Note that in Problem $(\mathrm{B_{LGR}})$, the matrix multiplication $\boldsymbol{D}^{LGR} \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{x}_f \end{bmatrix}$ produces an N by N resultant matrix, so the equality is consistent with regards to the dimensions of the left- and right-hand sides. The covector mapping theorem for the LGR PS method is given as,

$$\boldsymbol{\lambda}_i^{LGR} = \frac{\boldsymbol{\Lambda}_i^{LGR}}{w_i^{LGR}}, \qquad (i = 1, \ldots, N), \tag{4.31}$$

$$\boldsymbol{\lambda}_i^{LGR} = \left[\boldsymbol{D}_{(:,i)}^{LGR}\right]^\top \boldsymbol{\Lambda}_i^{LGR}, \qquad (i = N+1), \tag{4.32}$$

$$\boldsymbol{\gamma}_i^{LGR} = \frac{\boldsymbol{\Gamma}_i^{LGR}}{w_i^{LGR}}, \qquad (i = 1, \ldots, N), \tag{4.33}$$

where $\boldsymbol{D}^{LGR}_{(:,i)}$ denotes the i-th column of the LGR differentiation matrix and w_i^{LGR} are the LGR quadrature weights. Note that Eq. (4.31), Eq. (4.32), and Eq. (4.33) are taken from [33]. However, Françolin et al. [56] derive different costate estimate expressions that may potentially be more accurate. However, the estimate presented in [56] requires derivative information, and as such is excluded for the sake of ease of use.

There is one outstanding point regarding the LGR method that must be addressed. Since the non-collocated point is only introduced to the state, the final control value is not obtained using the standard LGR points and must be extrapolated. The remedy to this is to use the *flipped* LGR points, which are obtained as,

$$\text{Flipped LGR} \quad \Rightarrow \text{the roots of } -\left(P_{N-1}(\tau) + P_N(\tau)\right).$$

Thus, the flipped LGR points are just the negative LGR points. The flipped LGR points are identical to the standard set of LGR points, except for the fact that they are mirrored across the y-axis. To use these points, introduce the non-collocated point at $\tau = -1$ rather than $\tau = 1$ and follow the procedure above. The primary difference is to reverse the order of the quadrature weights once they are obtained. It is obvious that the control will no longer be obtained at the initial time. However, it is sometimes desirable to extrapolate the initial control value rather than the final control value.

4.3.1 Integral Formulation of LGR Method

In [58] it was shown that the differential LGR PS method given in Problem $(\mathrm{B}_{\mathrm{LGR}})$ has an equivalent integrated formulation for both the standard and flipped LGR points. This results from the property that,

$$\boldsymbol{D}_{(:,1)} = -\boldsymbol{D}_{(:,2:N+1)}\mathbf{1}, \tag{4.34}$$

where $\mathbf{1} \in \mathbb{R}^{N\times 1}$. This allows for a proof that $\boldsymbol{D}_{(:,2:N+1)}$ is non-singular. Defining $\boldsymbol{A} = \boldsymbol{D}^{-1}_{(:,2:N+1)}$, we may write,

$$\boldsymbol{X}_{2:N+1} = \boldsymbol{x}_1 + \frac{t_f - t_0}{2}\boldsymbol{A}_i \boldsymbol{f}(\boldsymbol{X}^{\mathrm{LGR}}, \boldsymbol{U}^{\mathrm{LGR}}, \boldsymbol{t}^{LGR}). \tag{4.35}$$

This integral formulation is equivalent to the differential formulation presented previously. Note that for a multiple-interval scheme, the row vector $\boldsymbol{x}_1$ is the state at $\tau_1^{(k)}$, the initial point in *every* interval k. The primary reason to implement this method is numerical, as the NLP problem resulting from using this collocation equation produces derivatives that exhibit somewhat more sparsity than those resulting from the differential formulation. In addition, it is an implicit integration scheme and may exhibit additional stability compared to the differential form.

4.4 Legendre-Gauss Points

The final set of Legendre-Gauss points that will be discussed are the Legendre-Gauss (LG) points themselves. In the experience of the author, these are the least used of the Gauss points. Similar to the LGR points, no state/control convergence proofs exist for the LG points. However, a proof of costate convergence does exist [59].

Consider collocation at N LG points. The LG points are obtained as [1],

$$\text{LG} \quad \Rightarrow \text{the roots of } P_N(\tau).$$

Note that interpolation and differentiation/quadrature operations using the LG points are exact for polynomials of degree up to $2N-1$ [59]. There is no closed form expression for the LG points. However, the locations of Gauss points can be calculated numerically to machine precision using algorithms found in the preface of Shen et al. [159]. Additional MATLAB based algorithms can be found in the MATLAB file exchange [182]. Fig. 4.4 shows a schematic of the distribution of LG, LGR, and LGL points. Note that the LG points lie on the open interval $\tau \in (-1,1)$. The quadrature weights for the LG transcription are given as [51, 159],

$$w_i^{LG} = \frac{2}{(1-\tau_i^2)\left[\dot{P}_{M+1}(\tau_i)\right]^2}, \quad (i = 0, 1, \ldots, M), \tag{4.36}$$

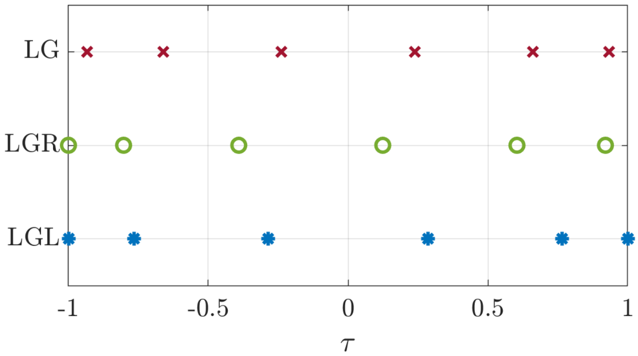


Figure 4.4: The LG, LGR, and LG Points

and the differentiation matrix can be calculated as,

$$D_{ij}^{LG} = \begin{cases} \dfrac{\dot{P}_M(\tau_i)}{(\tau_i - \tau_j)\dot{P}_M(\tau_j)}, & i \neq j,\ 0 \leq i, j \leq M, \\ \dfrac{\tau_i}{1-\tau_i^2}, & i = j, \end{cases} \tag{4.37}$$

again letting $M \leftarrow N-1$. A MATLAB-based algorithm that can be used to calculate this differentiation matrix can be found at [185]. Next, we let $\tau_0 = -1$ and $\tau_{N+1} = 1$ be non-collocated discretization points. We denote the state vectors associated with these points to be $\boldsymbol{x}_0$ and $\boldsymbol{x}_f$, respectively. Note that we do not approximate the state at every point with a Lagrange polynomial. Instead, we approximate the state at the collocation points and *only* the point τ_0 [59]. Thus, the state approximation becomes,

$$\dot{\boldsymbol{x}}(\tau_j) \approx \sum_{j=0}^{N} D_{ij}^{LG} \boldsymbol{X}_j = \boldsymbol{D}^{LG} \boldsymbol{X}^{LG}, \tag{4.38}$$

where N is the number of LG points and $\boldsymbol{D}^{LG}$ is the N by $N+1$ rectangular LG differentiation matrix. Using the property $\boldsymbol{D}_{(:,0)} = -\boldsymbol{D}_{(:,1:N)}\mathbf{1}$ for the LG differentiation matrix, Garg et al. [60] show that,

$$\boldsymbol{x}_f = \boldsymbol{x}_0 + \boldsymbol{w}_{LG}^{\top} \boldsymbol{f}(\boldsymbol{X}^{LG}, \boldsymbol{U}^{LG}, \boldsymbol{t}^{LG}), \tag{4.39}$$

where we define,

$$\boldsymbol{w}_{LG} = [w_1^{LG}, w_2^{LG}, \ldots, w_N^{LG}]^\top, \tag{4.40}$$

to be a column vector of the LG weights and,

$$\boldsymbol{t}^{LG} = [t_1, t_2, \ldots, t_N]^\top, \tag{4.41}$$

to be the time points in the domain $(-1, 1)$ associated with the LG points *only*. They are obtained using Eq. (4.2). Thus, we may write,

$$\boldsymbol{x}_0 \in \mathbb{R}, \;\; \boldsymbol{X} = \boldsymbol{X}^{LG} \in \mathbb{R}^{N \times N_x}, \;\; \boldsymbol{x}_f \in \mathbb{R}^{N_x}, \;\; \boldsymbol{U} = \boldsymbol{U}^{LG} \in \mathbb{R}^{N \times N_u}, \;\; t_0 \in \mathbb{R}, \;\; t_f \in \mathbb{R},$$

$$(\mathrm{B_{LG}}) \begin{cases} \text{Minimize} \quad J\left[\boldsymbol{x}_0, \boldsymbol{X}, \boldsymbol{x}_f, \boldsymbol{U}, t_0, t_f\right] = E(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f) \\ \qquad\qquad\qquad + \dfrac{t_f - t_0}{2} \displaystyle\sum_{i=1}^{N} w_i^{LG} F(\boldsymbol{X}_i, \boldsymbol{U}_i, t_i), \\ \text{Subject to :} \quad \boldsymbol{D}^{LG} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{X} \end{bmatrix} = \dfrac{t_f - t_0}{2} \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}^{LG}), \\ \qquad\qquad \boldsymbol{x}_f = \boldsymbol{x}_0 + \boldsymbol{w}_{LG}^\top \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}^{LG}), \\ \qquad\qquad \boldsymbol{e}(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f) \leq \boldsymbol{0}, \\ \qquad\qquad \boldsymbol{h}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}^{LG}) \leq \boldsymbol{0}. \end{cases}$$

It is obvious from this formulation that neither the initial nor the final control values can be obtained explicitly and must be extrapolated. In the opinion of the author, the primary reason to use these points is accuracy for computational cost, as the LG points offer the highest accuracy of the three transcriptions for the fewest number of collocation points [59]. The covector mapping theorem for the LG PS method is given as [33],

$$\boldsymbol{\lambda}_i^{LG} = \boldsymbol{\Lambda}_{N+1}^{LG} - \left[\boldsymbol{D}_{(:,0)}^{LG}\right]^\top \boldsymbol{\Lambda}^{LG}, \quad (i = 0), \tag{4.42}$$

$$\boldsymbol{\lambda}_i^{LG} = \mathrm{diag}(\boldsymbol{w})^{-1}\boldsymbol{\Lambda}^{LG}, \quad (i = 1, \ldots, N), \tag{4.43}$$

$$\boldsymbol{\lambda}_i^{LG} = \boldsymbol{\Lambda}_{N+1}^{LG}, \quad (i = N+1), \tag{4.44}$$

$$\boldsymbol{\gamma}_i^{LG} = \frac{\boldsymbol{\Gamma}_i^{LG}}{w_i^{LG}} \quad (i = 1, \ldots, N). \tag{4.45}$$

Note that the inequality constraint multipliers are only obtained at the collocation points.

4.5 Multi-Interval LGR Pseudospectral Transcription

Now the multi-interval formulation of the LGR transcription is presented. Note that this formulation can be easily adapted to other PS transcriptions if appropriate care is taken. **Multi-interval methods are primarily used for mesh refinement purposes**. See Section 5.2.2 for more details. Before continuing, the reader is informed that Section 4.5.1 presents a simplified form of the multi-interval LGR PS method. However, this section is included for completeness.

Let us partition the independent variable $\tau \in \mathcal{S} = [-1, 1]$ into K mesh intervals, $\mathcal{S}_k = [T_{k-1}, T_k]$, $(k = 1, 2, \ldots, K)$ such that $-1 = T_0 < T_1 < \cdots < T_K = 1$. Assuming $k > 1$, it follows that $\mathcal{S}_k \subset \mathcal{S}\ \forall\ k$. Thus, the set $\{S_k\}_{k=1}^K$ has the property that

$$\bigcup_{k=1}^{K} \mathcal{S}_k = \mathcal{S}. \tag{4.46}$$

In addition, each interval $\mathcal{S}_k$ has *no overlap* with any other interval. Thus, $T_k > T_{k-1}\ \forall\ k$. The superscript (k) and the subscript k will be used interchangeably to denote the mesh interval to which a particular value belongs. For example, we denote the i-th state in mesh interval k as $\boldsymbol{X}_i^{(k)}$, while denoting the mesh interval itself as $\mathcal{S}_k$. In the interest of reducing notational clutter, superscripts denoting the method being used (e.g., $\boldsymbol{X}^{LGR}$) will be dropped. The multiple-interval form of the Bolza optimal control problem is,

$$\boldsymbol{x} \in \mathbb{R}^{N_x}, \;\; \boldsymbol{u} \in \mathbb{R}^{N_u}, \;\; t_0 \in \mathbb{R}, \;\; t_f \in \mathbb{R}, \;\; (k = 1, \dots, K),$$

$$(\mathrm{B_{MI}}) \begin{cases} \text{Minimize} \quad J\left[\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot), t_0, t_f\right] = E(\boldsymbol{x}_0^{(1)}, \boldsymbol{x}_f^{(K)}, t_0, t_f) \\ \qquad\qquad\qquad + \dfrac{t_f - t_0}{2} \displaystyle\sum_{k=1}^{N} \int_{T_{k-1}}^{T_k} F(\boldsymbol{x}^{(k)}, \boldsymbol{u}^{(k)}, t^{(k)}) d\tau, \\ \text{Subject to} \quad \dfrac{d\boldsymbol{x}^{(k)}(\tau)}{d\tau} = \dfrac{t_f - t_0}{2} \boldsymbol{f}(\boldsymbol{x}^{(k)}, \boldsymbol{u}^{(k)}, t^{(k)}), \\ \qquad \boldsymbol{e}(\boldsymbol{x}_0^{(1)}, \boldsymbol{x}_f^{(K)}, t_0, t_f) \leq \boldsymbol{0}, \\ \qquad \boldsymbol{h}(\boldsymbol{x}^{(k)}, \boldsymbol{u}^{(k)}, t^{(k)}) \leq \boldsymbol{0}. \end{cases}$$

Following the procedure in [41], each interval $\mathcal{S}_k$ of Problem $(\mathrm{B_{MI}})$ is transformed to the independent variable $s \in [-1, 1]$. First, note that s and τ can be related as,

$$\tau = \frac{T_k - T_{k-1}}{2} s + \frac{T_k + T_{k-1}}{2}, \tag{4.47}$$

which implies that,

$$\frac{d\tau}{ds} = \frac{T_k - T_{k-1}}{2} := \sigma_k. \tag{4.48}$$

The state of the continuous-time Bolza OCP can be approximated in each mesh interval as,

$$\boldsymbol{x}^{(k)}(s) \approx \boldsymbol{X}^{(k)}(s) = \sum_{j=1}^{N_k+1} \boldsymbol{X}_j^{(k)} \ell_j^{(k)}(s), \tag{4.49}$$

where

$$\ell_j^{(k)}(s) = \prod_{\substack{l=1 \\ l \neq j}}^{N_k+1} \frac{s - s_l^{(k)}}{s_j^{(k)} - s_l^{(k)}}. \tag{4.50}$$

Differentiating Eq. (4.49) with respect to s gives us,

$$\dot{\boldsymbol{x}}^{(k)}(s) \approx \dot{\boldsymbol{X}}^{(k)}(s) = \sum_{j=1}^{N_k+1} \boldsymbol{X}_j^{(k)} \dot{\ell}_j^{(k)}(s). \tag{4.51}$$

Collocating at the N_k LGR points in each mesh interval, we obtain the dynamic constraints for interval $(k = 1, \ldots, K)$ as,

$$\sum_{j=1}^{N_k+1} D_{ij}^{(k)} \boldsymbol{X}_j^{(k)} = \frac{t_f - t_0}{2} \sigma_k \boldsymbol{f}(\boldsymbol{X}_i^{(k)}, \mathbf{U}_i^{(k)}, t_i^{(k)}), \quad (i = 1, \ldots, N_k), \tag{4.52}$$

where

$$D_{ij}^{(k)} = \frac{d\ell_j^{(k)}\left(s_i^{(k)}\right)}{ds} = \dot{\ell}_j^{(k)}\left(s_i^{(k)}\right), \quad (i = 1, \ldots, N_k), \quad (j = 1, \ldots, N_k + 1), \tag{4.53}$$

are the elements of the $N_k \times (N_k + 1)$ rectangular LGR differentiation matrix [58, 60] in the interval $\mathcal{S}_k$. Here, we have introduced a non-collocated point at the end of each mesh interval. Note that by transforming each interval to the domain $s \in [-1, 1]$, the derivatives of the Lagrange polynomial basis are defined over $[-1, 1]$ in each segment. Thus, $\boldsymbol{D}^{(k)}$ and $\boldsymbol{w}^{(k)}$ may be obtained using the standard formulas. Additionally, $\boldsymbol{t}^{(k)}$ is actually a function of $\boldsymbol{\tau}^{(k)}$, $\boldsymbol{s}^{(k)}$, T_k, T_{k-1}, t_0, and t_f and can be obtained by applying Eq. (4.47) and then Eq. (4.2). For obvious reasons, these dependencies are dropped. In addition to the standard equality constraints that are part of the NLP, we must enforce continuity of the state. Otherwise, the NLP will allow for a discontinuity or "jump" in the state value at the mesh points. This constraint takes the form,

$$\boldsymbol{x}_{N_k+1}^{(k-1)} = \boldsymbol{x}_1^{(k)}, \quad (k = 2, \ldots, K - 1). \tag{4.54}$$

However, the equality constraint given by Eq. (4.54) can be satisfied implicitly by *using the same NLP variable* for $\boldsymbol{x}_{N_k+1}^{(k)}$ and $\boldsymbol{x}_0^{(k+1)}$ at the interior mesh points, although this does make the NLP somewhat more difficult to implement. Next, we apply Gaussian quadrature to the cost to obtain,

$$\begin{aligned} J[\boldsymbol{X}, \boldsymbol{U}, t_0, t_f] = E\left(\boldsymbol{x}_1^{(1)}, \boldsymbol{x}_{N_k+1}^{(K)}, t_0, t_f\right) \\ + \frac{t_f - t_0}{2} \sum_{k=1}^{K} \sum_{i=1}^{N_k} \sigma_k w_i^{(k)} F\left(\boldsymbol{X}_i^{(k)}, \boldsymbol{U}_i^{(k)}, t_i^{(k)}\right). \end{aligned} \tag{4.55}$$

The discrete Bolza problem leads to the following NLP.

$$
\boldsymbol{X} \in \mathbb{R}^{(N_t+1)\times N_x},\;\; \boldsymbol{U} \in \mathbb{R}^{N_t \times N_u},\;\; t_0,\;\; t_f,
$$

$$
(\mathrm{B}^{\mathrm{LGR}}_{\mathrm{MI}}) \begin{cases} \text{Minimize} & J\left[\boldsymbol{X}, \boldsymbol{U}, t_0, t_f\right] = E(\boldsymbol{x}^{(1)}_1, \boldsymbol{x}_f, t_0, t_f) \\ & \qquad + \dfrac{t_f - t_0}{2} \displaystyle\sum_{k=1}^{K} \sum_{i=1}^{N_k} \sigma_k w^{(k)}_i F(\boldsymbol{X}^{(k)}_i, \mathbf{U}^{(k)}_i, t^{(k)}_i), \\ \text{Subject to :} & \displaystyle\sum_{j=1}^{N_k+1} D^{(k)}_{ij} \boldsymbol{X}^{(k)}_j = \dfrac{t_f - t_0}{2} \sigma_k \boldsymbol{f}(\boldsymbol{X}^{(k)}_i, \boldsymbol{U}^{(k)}_i, t^{(k)}_i), \\ & \boldsymbol{e}(\boldsymbol{x}^{(1)}_1, \boldsymbol{x}_f, t_0, t_f) \leq \mathbf{0}, \\ & \boldsymbol{h}(\boldsymbol{X}^{(k)}_i, \boldsymbol{U}^{(k)}_i, t^{(k)}_i) \leq \mathbf{0}, \end{cases}
$$

$$
(i = 1, \dots, N_k), \quad (j = 1, \dots, N_k + 1), \quad (k = 1, \dots, K).
$$

In $(\mathrm{B}^{\mathrm{LGR}}_{\mathrm{MI}})$, $N_t = \sum_{k=1}^{K} N_k$ is the total number of LGR support points. This completes the multi-interval LGR PS method. At this point, it may be difficult to interpret the overwhelming number of sums and indices. The next section aims to explain the details of the complex notation introduced during the derivation of $(\mathrm{B}^{\mathrm{LGR}}_{\mathrm{MI}})$ and re-express the problem in a simpler form.

4.5.1 Matrix Form of the Bolza NLP

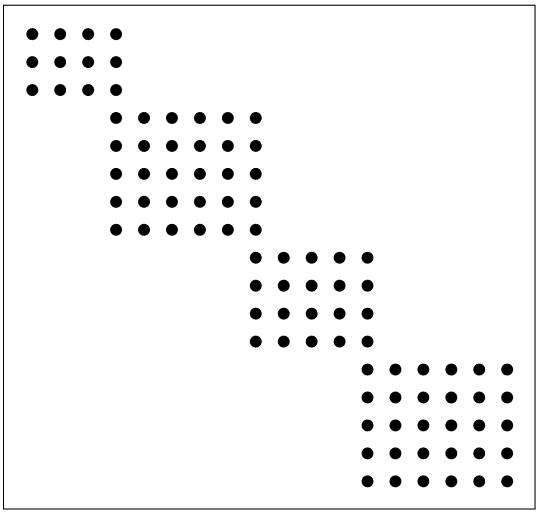
A common difficulty for researchers new to the field of PS direct methods is that the equations of the PS method are difficult to understand without first suffering through a painful dissection of these equations or seeing them implemented. Indeed, the author of this study shared similar complaints with his advisor when they began their study of PS methods. This section endeavors to alleviate the difficulties by posing Problem $(\mathrm{B}^{\mathrm{LGR}}_{\mathrm{MI}})$ in a manner more friendly to both the eye and implementation on a computer. In order to accomplish this, variables belonging to mesh intervals (k) will be arranged into matrices that can be used in a "global" sense. Let us first re-define the states, controls, and KKT multipliers as,

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{X}^{(1)} \\ \boldsymbol{X}^{(2)} \\ \vdots \\ \boldsymbol{X}^{(K)} \end{bmatrix}, \quad \boldsymbol{U} = \begin{bmatrix} \boldsymbol{U}^{(1)} \\ \boldsymbol{U}^{(2)} \\ \vdots \\ \boldsymbol{U}^{(K)} \end{bmatrix}, \quad \boldsymbol{\Lambda} = \begin{bmatrix} \boldsymbol{\Lambda}^{(1)} \\ \boldsymbol{\Lambda}^{(2)} \\ \vdots \\ \boldsymbol{\Lambda}^{(K)} \end{bmatrix}, \quad \boldsymbol{\Gamma} = \begin{bmatrix} \boldsymbol{\Gamma}^{(1)} \\ \boldsymbol{\Gamma}^{(2)} \\ \vdots \\ \boldsymbol{\Gamma}^{(K)} \end{bmatrix}, \tag{4.56}$$

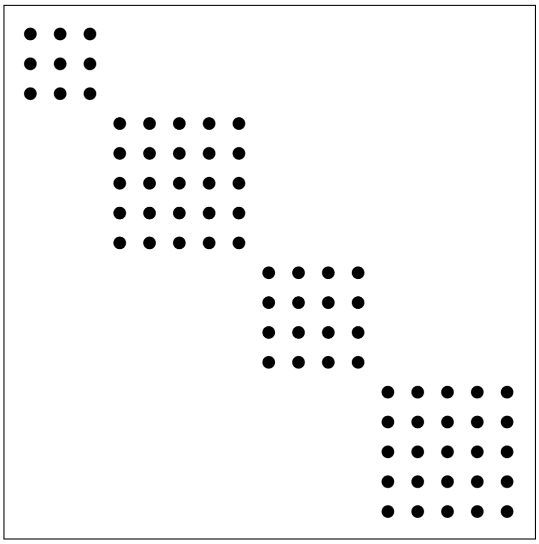
where $\boldsymbol{X} \in \mathbb{R}^{N_t \times N_x}$, $\boldsymbol{U} \in \mathbb{R}^{N_t \times N_u}$, $\boldsymbol{\Lambda} \in \mathbb{R}^{(N_t+1) \times N_x}$, and $\boldsymbol{\Gamma} \in \mathbb{R}^{N_t \times N_h}$. Notice that the non-collocated point is **not** included in $\boldsymbol{X}$. Instead, define $\boldsymbol{x}_f \in \mathbb{R}^{1 \times N_x}$ as the row vector of non-collocated state NLP variables at t_f. Next, define a composite differentiation matrix as,

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{D}^{(1)}_{N_1 \times (N_1+1)} & \mathbf{0}_{N_1 \times N_2} & \cdots & \mathbf{0}_{N_1 \times N_K} \\ \mathbf{0}_{N_2 \times N_1} & \boldsymbol{D}^{(2)}_{N_2 \times (N_2+1)} & \cdots & \mathbf{0}_{N_2 \times N_K} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_{N_K \times N_1} & \mathbf{0}_{N_K \times N_2} & \cdots & \boldsymbol{D}^{(K)}_{N_K \times (N_K+1)} \end{bmatrix} \in \mathbb{R}^{N_t \times (N_t+1)}. \tag{4.57}$$

In Eq. (4.57), $N_t = N_1 + N_2 + \ldots + N_K$ and subscripts denote the matrix dimension. Note in Eq. (4.57) that the all-zero blocks have one less column than the differentiation matrix blocks. This is because the same NLP variable is used for both the (non-collocated) segment end-point and for the first point of the next segment. Thus, the differentiation matrices must be overlapped so that the last column of $\boldsymbol{D}^{(k-1)}$ lies in the same column and directly above the first column of $\boldsymbol{D}^{(k)}$. A visualization of the sparsity pattern that this produces for an LGR composite differentiation matrix is shown in Fig. 4.5a. Fig. 4.5b shows the sparsity pattern for the integral formulation. This sparsity pattern is specific only to the Radau PS method, although this overlapping method could potentially be used for any other transcription. Let $\boldsymbol{w}^{(k)} = [w_i^{(k)}]_{i=1}^{N_k}$ denote the LGR weights in interval k and let $\mathbf{1}_{n \times m}$ represent an $n \times m$ matrix of all ones. Then, $\boldsymbol{\sigma}^{(k)} = \sigma_k \mathbf{1}_{N_k \times 1}$ represents the weight scaling vector in segment k. Finally, let $\boldsymbol{t}^{(k)} = [t_i]_{i=1}^{N_k}$ denote the times at the LGR point in segment k and $\boldsymbol{s}^{(k)} = [s_i]_{i=1}^{N_k}$ denote the LGR support points in $\mathcal{S}_k = [T_{k-1}, T_k]$ where $(k = 1, \ldots, K)$. We may define several global



(a) Composite Differentiation Matrix



(b) Composite Integration Matrix

Figure 4.5: LGR Composite D/I Matrices Sparsity Patterns.

matrices as follows.

$$\boldsymbol{t} = \begin{bmatrix} \boldsymbol{t}^{(1)} \\ \boldsymbol{t}^{(2)} \\ \vdots \\ \boldsymbol{t}^{(K)} \end{bmatrix}, \qquad \boldsymbol{s} = \begin{bmatrix} \boldsymbol{s}^{(1)} \\ \boldsymbol{s}^{(2)} \\ \vdots \\ \boldsymbol{s}^{(K)} \end{bmatrix}, \qquad \boldsymbol{w} = \begin{bmatrix} \boldsymbol{w}^{(1)} \\ \boldsymbol{w}^{(2)} \\ \vdots \\ \boldsymbol{w}^{(K)} \end{bmatrix}, \qquad \boldsymbol{\sigma} = \mathrm{diag} \begin{bmatrix} \boldsymbol{\sigma}^{(1)} \\ \boldsymbol{\sigma}^{(2)} \\ \vdots \\ \boldsymbol{\sigma}^{(K)} \end{bmatrix}. \tag{4.58}$$

Note that $\boldsymbol{t}, \boldsymbol{s}, \boldsymbol{w} \in \mathbb{R}^{N_t \times 1}$ while $\boldsymbol{\sigma} \in \mathbb{R}^{N_t \times N_t}$. Although $\boldsymbol{s}$ is not used directly in this formulation, it must be used to obtain $\boldsymbol{t}$ in the NLP solver for free-final-time problems, where t_f is a decision variable. Additionally, it is used in Section 5.4 to transform certain derivatives.

Now, Problem $(\mathrm{B}_{\mathrm{MI}}^{\mathrm{LGR}})$ may be restated using a compact matrix notation.

$$
\begin{aligned}
&\qquad\qquad \boldsymbol{X} \in \mathbb{R}^{N_t \times N_x},\ \ \boldsymbol{x}_f \in \mathbb{R}^{1 \times N_x},\ \ \boldsymbol{U} \in \mathbb{R}^{N_t \times N_u},\ \ t_0,\ \ t_f,\\
&(\mathrm{B}_{\mathrm{MAT}}^{\mathrm{LGR}}) \begin{cases}
\text{Minimize} \quad J\left[\boldsymbol{X}, \boldsymbol{x}_f, \boldsymbol{U}, t_0, t_f\right] = E(\boldsymbol{x}_1^{(1)}, \boldsymbol{x}_f, t_0, t_f) \\
\qquad\qquad\qquad\qquad\qquad\qquad + \dfrac{t_f - t_0}{2} \boldsymbol{w}^\top \boldsymbol{\sigma} F\left(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}\right), \\
\text{Subject to :} \qquad \boldsymbol{D} \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{x}_f \end{bmatrix} = \dfrac{t_f - t_0}{2} \boldsymbol{\sigma} \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}), \\
\qquad \boldsymbol{e}(\boldsymbol{x}_1^{(1)}, \boldsymbol{x}_f, t_0, t_f) \leq \boldsymbol{0}, \\
\qquad \boldsymbol{h}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}) \leq \boldsymbol{0}.
\end{cases}
\end{aligned}
$$

Finally, the costate estimate may be re-written as,

$$\boldsymbol{\lambda}_{1:N_t} = \mathrm{diag}(\boldsymbol{w})^{-1} \boldsymbol{\Lambda}_{1:N_t}, \tag{4.59}$$

$$\boldsymbol{\lambda}_{N_t+1} = \boldsymbol{D}_{(:,N_t+1)} \boldsymbol{\Lambda}_{N_t+1}, \tag{4.60}$$

$$\gamma_{1:N_t} = \boldsymbol{\sigma}^{-1} \mathrm{diag}(\boldsymbol{w})^{-1} \boldsymbol{\Gamma}_{1:N_t}. \tag{4.61}$$

The benefits of re-formulating Problem $(\mathrm{B}_{\mathrm{MI}}^{\mathrm{LGR}})$ into Problem $(\mathrm{B}_{\mathrm{MAT}}^{\mathrm{LGR}})$ are twofold. The first is readability. The second and the more practical reason, is that for interpreted computing languages such as MATLAB or Python, sparse matrix operations are generally much faster than loops. Looping can become quite inefficient when inside a function that must be evaluated many times. Assembling sparse weight, scaling, and differentiation matrices prior to optimization allows loops to be eliminated and computational efficiency to be increased in general. Now that we have established a general, programming-friendly formulation for discrete OCPs using an LGR PS method, a numerical implementation is introduced.

4.6 Chebyshev-Gauss-Lobatto Pseudospectral Method

Another very common set of Gauss points are the Chebyshev-Gauss-Lobatto (CGL) points. CGL points are popular for a number of reasons. The first is that when approximating a function, a Chebyshev expansion is very close to the best polynomial approximation in the infinity norm [27, 179]. In addition to this, the CGL points, wights, and differentiation matrix are computed using a *closed-form expression*, making them very attractive from a computational perspective [27, 50, 69]. The CGL points are given as [178, 69],

$$\tau_k = \cos\left[\frac{\pi(M-k)}{M}\right], \quad (k = 0, 1, \ldots, M). \tag{4.62}$$

These points lie in the interval $[-1, 1]$ and are the extrema of the N-th order Chebyshev polynomial,

$$T_M(\tau) = \cos\left[M\cos^{-1}(\tau)\right]. \tag{4.63}$$

The state is approximated using a basis of Lagrange basis polynomials as,

$$\boldsymbol{x}(\tau) \approx \boldsymbol{X}(\tau) = \sum_{k=1}^{N} \boldsymbol{x}_k \phi_k(\tau), \tag{4.64}$$

where,

$$\phi_k(\tau) = \frac{(-1)^{k+1}}{M^2 c_k}\frac{(1-\tau^2)\dot{T}_M(\tau)}{\tau - \tau_k}, \tag{4.65}$$

$$c_k = \begin{cases} 2, & \text{if } \ k = 0, M, \\ 1, & \text{if } \ 1 \leq k \leq M-1. \end{cases} \tag{4.66}$$

For M even, the weights are given by

$$w_s^{CGL} = \begin{cases} \dfrac{1}{M^2-1}, & (s = 0, M), \\ \dfrac{4}{M}\displaystyle\sum_{j=0}^{M/2''} \dfrac{1}{1-4j^2}\cos\left(\dfrac{2\pi js}{M}\right), & (s = 1, 2, \ldots, M/2), \end{cases} \tag{4.67}$$

and for M odd, we have

$$w_s^{CGL} = \begin{cases} \dfrac{1}{M^2}, & (s = 0, M), \\ \dfrac{4}{M}\displaystyle\sum_{j=0}^{(M-1)/2''} \dfrac{1}{1-4j^2}\cos\left(\dfrac{2\pi js}{M}\right), & (s = 1, 2, \ldots, (M-1)/2), \end{cases} \tag{4.68}$$

where the double prime superscript for the summation index means that the first and last elements of the sumations should be halved [69]. For N collocation points, take $M \leftarrow N-1$. Note that these weights are the Clenshaw-Curtis quadrature weights [179].

This integration scheme is exact for polynomials of degree $N-1$. While this is slightly less accurate the Gaussian schemes, the weights and matrices can be calculated using Fast-Fourier-Transform (FFT) algorithms. Additionally, the discrete integration scheme is convergent for any continuous function [69] and its *practical* accuracy is as good as Gauss quadrature, since the factor-of-2 accuracy exhibited by Gauss schemes is rarely realized in computational algorithms [179]. Next, the differentation matrix is given by,

$$D_{kj}^{CGL} = \begin{cases} \dfrac{c_k}{c_j}\dfrac{(-1)^{j+k}}{\tau_j - \tau_k}, & j \neq k, \\ \dfrac{\tau_k}{2(1-\tau_k^2)}, & 1 \leq j = k \leq M-1, \\ -\dfrac{2M^2+1}{6}, & j = k = 0, \\ \dfrac{2M^2+1}{6}, & j = k = M, \end{cases} \tag{4.69}$$

where we take $M \leq N-1$ for N collocation points. Thus, the single-interval CGL NLP problem is given by,

$$\boldsymbol{X} = \boldsymbol{X}^{CGL} \in \mathbb{R}^{N \times N_x}, \;\; \boldsymbol{U} = \boldsymbol{U}^{CGL} \in \mathbb{R}^{N \times N_u}, \;\; t_0 \in \mathbb{R}, \;\; t_f \in \mathbb{R}$$

$$(\mathrm{B_{CGL}}) \begin{cases} \text{Minimize} & J[\boldsymbol{X}, \boldsymbol{U}, t_0, t_f] = E(\boldsymbol{x}_1, \boldsymbol{x}_N, t_0, t_f) \\ & \qquad + \dfrac{t_f - t_0}{2} \displaystyle\sum_{i=1}^{N} w_i^{CGL} F(\boldsymbol{X}_i, \boldsymbol{U}_i, t_i), \\ \text{Subject to :} & \boldsymbol{D}^{CGL}\boldsymbol{X} = \dfrac{t_f - t_0}{2}\boldsymbol{f}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}^{CGL}), \\ & \boldsymbol{e}(\boldsymbol{x}_1, \boldsymbol{x}_N, t_0, t_f) \leq \boldsymbol{0}, \\ & \boldsymbol{h}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}^{CGL}) \leq \boldsymbol{0}, \end{cases}$$

where we take,

$$\boldsymbol{t}^{CGL} = [t_1, t_2, \ldots, t_N]^\top, \tag{4.70}$$

to be the discrete time values associated with the CGL collocation points. A multi-interval form can be adapted from Section 4.5. The covector mapping for the CGL transcription is given as,

$$\boldsymbol{\lambda}_i^{CGL} = \frac{\boldsymbol{\Lambda}_i^{CGL}}{w_i^{CGL}}, \quad \boldsymbol{\gamma}_i^{CGL} = \frac{\boldsymbol{\Gamma}_i^{CGL}}{w_i^{CGL}}, \quad (i = 1, \dots, N). \tag{4.71}$$

A CGL scheme is implemented in the TOPS solver and can be used by setting the user-flag `p.opts.transcription = 'CGL'`.

4.7 Covector Mapping Theorem

The covector mapping theorem is a powerful feature of PS methods. It was first proved by Fahroo and Ross [49] for the LGL points, but several proofs for the LG, LGR, and CGL transcriptions have been developed since then [59, 12, 69]. This section will briefly detail the relationship between the KKT conditions and a PS discretization of the continuous costate equation provided in [59, 60, 61]. This derivation of the mappings for a generalized LGR NLP problem was presented by Darby et al. [33]. However, Françolin et al. [56] presents another derivation that is potentially more accurate.

Consider a simplified single-interval form of the continuous Problem (B) given earlier as,

$$\boldsymbol{x} \in \mathbb{R}^{N_x}, \;\; \boldsymbol{u} \in \mathbb{R}^{N_u},$$
$$(\mathrm{B_S}) \begin{cases} \text{Minimize} \quad J\left[\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot)\right] = E(\boldsymbol{x}_0, \boldsymbol{x}_f) + \int_{-1}^{1} F(\boldsymbol{x}, \boldsymbol{u}, \tau) d\tau, \\ \text{Subject to :} \qquad \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, \tau), \\ \qquad \boldsymbol{e}(\boldsymbol{x}_0, \boldsymbol{x}_f) = \boldsymbol{0}, \\ \qquad \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}, \tau) \leq \boldsymbol{0}. \end{cases}$$

We assume that the time interval has already been scaled to $[-1, 1]$. We also adopt the notation $\boldsymbol{e} = \boldsymbol{0}$ to better represent the derivation process. The idea behind the covector mapping theorem is to derive the first-order optimality conditions for the continuous problem, derive the KKT conditions for the discrete problem, and then compare the resulting expressions. The augmented Hamiltonian and endpoint Lagrangian for Problem $(\mathrm{B_S})$ are [33],

$$\mathcal{H}(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{u},\boldsymbol{\gamma}) = F(\boldsymbol{x},\boldsymbol{u},\tau) + \langle \boldsymbol{\lambda}, \boldsymbol{f}(\boldsymbol{x},\boldsymbol{u},\tau)\rangle - \langle \boldsymbol{\gamma}, \boldsymbol{h}(\boldsymbol{x},\boldsymbol{u},\tau)\rangle\,, \tag{4.72}$$

$$\Phi(\boldsymbol{x}_0,\boldsymbol{x}_f,\boldsymbol{\nu}) = E(\boldsymbol{x}_0,\boldsymbol{x}_f) - \langle \boldsymbol{\nu}, \boldsymbol{e}(\boldsymbol{x}_0,\boldsymbol{x}_f)\rangle\,. \tag{4.73}$$

In Eqs. (4.72) and (4.73), $\boldsymbol{\lambda}$, $\boldsymbol{\gamma}$, and $\boldsymbol{\nu}$ are the Lagrange multipliers associated with the state, inequality, and equality constraints, respectively, and share the same dimensions. Assuming a fixed time horizon and applying the first order optimality conditions, we obtain,

$$\mathbf{0} = \nabla_{\boldsymbol{u}}\mathcal{H} = \nabla_{\boldsymbol{u}}\left(F + \langle \boldsymbol{\lambda}, \boldsymbol{f}\rangle - \langle \gamma, \boldsymbol{h}\rangle\right), \tag{4.74}$$

$$\dot{\boldsymbol{\lambda}} = \nabla_{\boldsymbol{x}}\mathcal{H} = \nabla_{\boldsymbol{x}}\left(F + \langle \boldsymbol{\lambda}, \boldsymbol{f}\rangle - \langle \gamma, \boldsymbol{h}\rangle\right), \tag{4.75}$$

$$\boldsymbol{\lambda}_0 = -\nabla_{\boldsymbol{x}_0}\Phi = -\nabla_{\boldsymbol{x}_0}\left(E - \langle \boldsymbol{\nu}, \boldsymbol{e}\rangle\right), \tag{4.76}$$

$$\boldsymbol{\lambda}_f = \nabla_{\boldsymbol{x}_f}\Phi = \nabla_{\boldsymbol{x}_f}\left(E - \langle \boldsymbol{\nu}, \boldsymbol{e}\rangle\right). \tag{4.77}$$

From the complementary slackness condition, the Lagrange multiplier $\boldsymbol{\gamma}$ takes the value,

$$\gamma_j(\tau) = 0, \quad \text{when}, \quad h_j(\tau) < 0, \quad 1 \le j \le N_h, \tag{4.78}$$

$$\gamma_j(\tau) < 0, \quad \text{when}, \quad h_j(\tau) = 0, \quad 1 \le j \le N_h. \tag{4.79}$$

Next, consider the discrete form of Problem $(\mathrm{B_S})$ obtained using an LGR transcription.

$$\boldsymbol{X} \in \mathbb{R}^{N\times N_x},\ \ \boldsymbol{x}_f \in \mathbb{R}^{1\times N_x},\ \ \boldsymbol{U} \in \mathbb{R}^{N\times N_u},$$

$$(\mathrm{B_S^{LGR}})\begin{cases} \text{Minimize} \quad J\left[\boldsymbol{X},\boldsymbol{x}_f,\boldsymbol{U},t_0,t_f\right] = E(\boldsymbol{x}_0,\boldsymbol{x}_f) + \boldsymbol{w}^{\top} F\left(\boldsymbol{X},\boldsymbol{U},\boldsymbol{\tau}\right), \\ \text{Subject to :} \qquad \boldsymbol{D}\begin{bmatrix}\boldsymbol{X}\\ \boldsymbol{x}_f\end{bmatrix} = \boldsymbol{f}(\boldsymbol{X},\boldsymbol{U},\boldsymbol{\tau}), \\ \qquad \boldsymbol{e}(\boldsymbol{x}_0,\boldsymbol{x}_f) = \mathbf{0}, \\ \qquad \boldsymbol{h}(\boldsymbol{X},\boldsymbol{U},\boldsymbol{\tau}) \le \mathbf{0}. \end{cases}$$

The Lagrangian of $(\mathrm{B}_\mathrm{S}^\mathrm{LGR})$ is given by,

$$\mathcal{L} = E(\boldsymbol{x}_0, \boldsymbol{x}_f) + \langle \boldsymbol{w}, F(\boldsymbol{X}, \boldsymbol{U}, \tau) \rangle - \langle \boldsymbol{\mu}, \boldsymbol{e}(\boldsymbol{x}_0, \boldsymbol{x}_f) \rangle \\ - \left\langle \boldsymbol{\Lambda}, \boldsymbol{D} \begin{bmatrix} \boldsymbol{X}, \\ \boldsymbol{x}_f \end{bmatrix} - \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{\tau}) \right\rangle - \langle \boldsymbol{\Gamma}, \boldsymbol{h}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{\tau}) \rangle, \quad (4.80)$$

The KKT conditions are given by,

$$\nabla_{\boldsymbol{X}} \mathcal{L} = \boldsymbol{0}, \quad \nabla_{\boldsymbol{U}} \mathcal{L} = \boldsymbol{0}. \quad (4.81)$$

We will see that once we take these derivatives, the results may be re-structured to be identical to the first-order necessary conditions of optimality. The intermediate steps will be skipped for brevity, but a detailed step-by-step description of the procedure can be found in Fahroo and Ross [49]. The discrete results are,

$$\boldsymbol{0} = \nabla_U \left(\langle \boldsymbol{w}, F \rangle + \langle \boldsymbol{\Lambda}, \boldsymbol{f} \rangle - \langle \boldsymbol{\Gamma}, \boldsymbol{h} \rangle \right), \quad (4.82)$$

$$\boldsymbol{D}_{(:,1)}^\top \boldsymbol{\Lambda} = \nabla_{\boldsymbol{x}_1} \left(\langle \boldsymbol{w}, F \rangle + \langle \boldsymbol{\Lambda}, \boldsymbol{f} \rangle - \langle \boldsymbol{\Gamma}, \boldsymbol{h} \rangle \right) + \nabla_{\boldsymbol{x}_1} \left(E - \langle \boldsymbol{\mu}, \boldsymbol{e} \rangle \right), \quad (4.83)$$

$$\boldsymbol{D}_{(:,2:N)}^\top \boldsymbol{\Lambda} = \nabla_{\boldsymbol{X}_{2:N}} \left(\langle \boldsymbol{w}, F \rangle + \langle \boldsymbol{\Lambda}, \boldsymbol{f} \rangle - \langle \boldsymbol{\Gamma}, \boldsymbol{h} \rangle \right), \quad (4.84)$$

$$\boldsymbol{D}_{(:,N+1)}^\top \boldsymbol{\Lambda} = \nabla_{\boldsymbol{x}_f} \left(E - \langle \boldsymbol{\mu}, \boldsymbol{e} \rangle \right). \quad (4.85)$$

If we multiply Eqs. (4.82) to (4.84) by $\mathrm{diag}(\boldsymbol{w})^{-1}$ and compare them to Eq. (4.74) and Eq. (4.75), we can see that,

$$\boldsymbol{\lambda}_i = \frac{\boldsymbol{\Lambda}_i}{w_i}, \quad (i = 1, \ldots, N), \quad (4.86)$$

$$\boldsymbol{\gamma}_i = \frac{\boldsymbol{\Gamma}_i}{w_i}, \quad (i = 1, \ldots, N). \quad (4.87)$$

For the multiple-interval formulation, the RHS of Eq. (4.87) gains a coefficient of $\boldsymbol{\sigma}$, as shown earlier. In addition, we can take the change of variables, $\boldsymbol{\pi} = \boldsymbol{D}_{(:,N+1)}^\top \boldsymbol{\Lambda}$. We also define the matrix $\boldsymbol{D}^\dagger$ as follows.

$$D^\dagger = -D_{11} - \frac{1}{w_1} \qquad D^\dagger_{ij} = -\frac{w_j}{w_i} D_{ji}. \tag{4.88}$$

Substituting Eqs. (4.86) to (4.88) into Eqs. (4.82) to (4.85) and simplifying, we obtain the transformed adjoint system as,

$$\mathbf{0} = \nabla_U \mathcal{H}, \tag{4.89}$$

$$\boldsymbol{D}^\dagger_{(:,1)} \boldsymbol{\lambda} = -\nabla_{\boldsymbol{X}_1} \mathcal{H} + \frac{1}{w_1} \nabla_{\boldsymbol{X}_1} \left(E - \langle \boldsymbol{\mu}, \boldsymbol{e} \rangle \right), \tag{4.90}$$

$$\boldsymbol{D}^\dagger_{(:,2:N)} \boldsymbol{\Lambda} = -\nabla_{\boldsymbol{X}_{2:N}} \mathcal{H}, \tag{4.91}$$

$$\boldsymbol{\pi} = \nabla_{\boldsymbol{x}_f} \left(E - \langle \boldsymbol{\mu}, \boldsymbol{e} \rangle \right). \tag{4.92}$$

From Eq. (4.77) and Eq. (4.92), we see that,

$$\lambda_{N+1} = \pi = \boldsymbol{D}^\top_{(:,N+1)} \boldsymbol{\Lambda}. \tag{4.93}$$

This completes the covector mapping theorem for a single interval. The reader should be aware that this derivation was intentionally simplistic and skipped several important steps. This was done in the spirit of illustrating the broad strokes of the covector mapping theorem. If the reader desires to inflict further agony upon themselves, they may find numerous (and more rigorous) derivations throughout the literature [49, 12, 52, 51, 58, 69, 59, 61].

Chapter 5

How to Use Pseudospectral Methods

Now that the theory behind PS methods has be thoroughly established, we move on to practical application of the methods presented in Chapter 4. If the reader has skipped the previous chapter, the author reminds them that they proceed at their own peril. Since a basic PS method is deceptively easy to implement, a lack of understanding of the *why* that motivates some of the applications presented here can cause things to go wrong as the reader tries to improve the efficiency of their algorithm. The primary difficulty will most likely occur when trying to provide analytical derivatives or create/implement a mesh-refinement algorithm. The reader is encouraged to at least read Section 4.1 and Section 4.5.1 before proceeding.

5.1 A Pseudocode Pseudospectral Algorithm

In this section, the author presents a brief MATLAB-friendly pseudocode PS algorithm that summarizes the key parts of the algorithm implemented in TOPS. However, the code for TOPS will be made publicly available under appropriate license later under ACE lab's GitHub account `https://github.com/TheAceLab`, so the reader is encouraged to browse it for themselves once they have absorbed the basics here.

Most NLP solvers are called in the following way.

```
% ===== Example NLP Solver Call ===== %
% NLP Solvers need a few things:
% 1. Objective/constraint functions
```

```
% 2. Initial guess (z0)
% 3. Box constraints/variable bounds

[solution, exit_data] = NLP_Solver(objective_fun, constraint_fun, z0, ...
    lower_bounds, upper_bounds, p);
```

Here, the structure `p` contains relevant problem data that we calculate out-of-the-loop (OOTL). By OOTL, the author means that the user should generate this data prior to calling the NLP solver and store it appropriately in `p`. In addition, most NLP solvers treat the optimization variables (here denoted "`z0`") as a column vector. This means that all states and controls must be "stacked" on top of each other. For the LGR transcription, the state and control matrices take the form,

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{X}^{(1)} \\ \vdots \\ \boldsymbol{X}^{(K)} \\ \boldsymbol{x}_f \end{bmatrix}, \qquad \boldsymbol{U} = \begin{bmatrix} \boldsymbol{U}^{(1)} \\ \vdots \\ \boldsymbol{U}^{(K)} \end{bmatrix}. \tag{5.1}$$

We have slightly redefined the matrix $\boldsymbol{X}$ originally defined in Section 4.5 for clarity. The author prefers to form the NLP optimization vector as,

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{X}_{(:,1)} \\ \vdots \\ \boldsymbol{X}_{(:,N_x)} \\ \boldsymbol{U}_{(:,1)} \\ \vdots \\ \boldsymbol{U}_{(:,N_u)} \\ t_0 \\ t_f \end{bmatrix} = \begin{bmatrix} \boldsymbol{X}_{(:)} \\ \boldsymbol{U}_{(:)} \\ t_0 \\ t_f \end{bmatrix}. \tag{5.2}$$

In the rightmost expression, we have used the notation defined in Section 1.1. In other words, place each column of the state and control matrices underneath the previous column, and then stack the state on top of the control. This is referred to as "unrolling." This allows for easy, linear indexing and, in the opinion of the author, makes derivative calculations easier. To recover the rolled form of the states/controls, the user can write, `state_indices ... = reshape(1:(N+1)*Nx, (N+1), Nx)`. This will create a state indexing matrix with the same dimensions as the state matrix, $\boldsymbol{X}$. Thus, calling `X = z(state_indices)` returns the state in the appropriate dimensions. A control indexing matrix can be generated similarly for N points. An LGR PS objective function typically looks like this.

```
% ===== My objective function ===== %
% Inputs:
% z: Current NLP optimization vector.
% p: Problem data.
% Outputs:
% J: Cost evaluated at z.
% ================================= %
J = function objective_fun(z,p)
% NLP solvers typically use a column optimization vector, but we
% need our states and controls as a matrix! We can get the
% indices of these matrices OOTL.
X  = z(p.state_indices);   % A ((N+1) x Nx) matrix.
Xc = X(1:end-1,:);         % A (N x Nx) matrix (excludes final state).
U  = z(p.control_indices); % An (N x Nu) matrix.
x1 = X(1,:);               % Initial state.
xf = X(end,:);             % Final state (not collocated).

% For fixed time problems...
t0 = p.t0;
tf = p.tf;

% For variable time problems...
if p.variable_final_time
    t0 = z(p.t0_index);
    tf = z(p.tf_index);
end
```

```

% Get a time vector for the running cost.
t = tau_to_t_affine(p.tau,t0,tf);

% These are functions defined by the user that take the current
% state and evaluate the endpoint and running cost functions.
E = user_endpoint_cost(x1,xf,t0,tf,p);
F = user_running_cost(Xc,U,t,p);

% Now we do PS quadrature. It's very easy!
w = p.quadrature_weights; % This is a column vector calculated OOTL
J = E + diag(w)*F; % And we're done!
end % objective function
```

Here, the objective function calls two user-defined functions that will calculate the endpoint cost and running cost Lagrangian. There are safer ways to do this using user-defined symbolic expressions rather than functions, but that will not be discussed here, as it is a computational issue not specific to the PS method. Now that we have completed our objective function, we can take a look at the constraints function. This function usually calculates both the equality and inequality constraints.

```
% ===== My Example Constraint Function ===== %
% Inputs:
% z: Current NLP optimization vector.
% p: Problem data.
% Outputs:
% equality: Equality constraints.
% inequality: Inequality constraints.
% ========================================== %
[equality,inequality] = function constraint_fun(z,p)
% First, we need our states and controls in matrix form.
X  = z(p.state_indices);    % States plus non-collocated state value.
Xc = X(1:end-1,:);          % Only the collocated states.
```

```
U  = z(p.control_indices); % Controls (same number of rows as Xc).
x1 = X(1,:);               % Initial state.
xf = X(end,:);             % Final state (not collocated).

% For fixed time problems...
t0 = p.t0;
tf = p.tf;

% For variable time problems...
if p.variable_time
    t0 = z(p.t0_index); % Initial/final times are part of z.
    tf = z(p.tf_index);
end

% Get a time vector for the dynamics.
t = tau_to_t_affine(p.tau,t0,tf);

% Now we calculate our dynamics constraints.
D      = p.differentiation_matrix; % We calculate this OOTL.
defect = D*X - 0.5*(tf - t0)*user_dynamics_fun(Xc,U,t,p);

% Get event constraints, e.
e = user_event_fun(x1, xf, t0, tf, p);

% Get inequality constraints, h.
h = user_inequality_fun(Xc, U, t, p);

% Stack everything!
equality   = [defect; e];
inequality = [h];
end
```

Upper and lower bounds are easy to construct. Note that most NLP solvers apply upper and lower bounds to $\boldsymbol{z}$, not $\boldsymbol{X}$ or $\boldsymbol{U}$, such that $\boldsymbol{z}^L \leq \boldsymbol{z} \leq \boldsymbol{z}^U$, where $\boldsymbol{z}^L$ and $\boldsymbol{z}^U$ are the lower and upper bounds of the variables in $\boldsymbol{z}$, respectively. The primary values that should be stored in p are initial and final times, Gaussian point distributions of $\tau \in [-1, 1]$ for the affine scaling given by Eq. (4.2), and the differentiation/integration matrices, as well as other problem-specific information. Anything that can be calculated *outside* the functions passed to the NLP solver should be calculated outside those functions. Store anything compatible with sparse formats in sparse matrices, such as the global LGR differentiation matrix discussed in Section 4.5. Sparse matrix operations can greatly increase the efficiency of the algorithm for problems with thousands or tens of thousands of constraints. In fact, the combined size of the constraint vectors for the LGR transcription is calculated as,

$$N_c = N_t(N_x + N_h) + N_e. \tag{5.3}$$

There are many ways to increase the efficiency of a PS algorithm. However, this can become a complex task, so the pseudocode algorithm here is extremely simplified.

5.2 Mesh Refinement

Mesh refinement is an important step in solving OCPs. This is because as OCPs become more difficult, the need for an algorithmic method of determining how to size or alter the number of points within a mesh becomes paramount. Simply increasing the number of nodes or dividing a mesh arbitrarily becomes computationally intractable as the dimension of the problem increases. In fact, this approach can lead to a degradation in the quality of the solution, even though a more dense mesh is being used [34]. As such, it is evident that an algorithmic approach is needed. In order to implement such an algorithm, two elements are required:

1. An error estimate for the current solution.
2. A logic for altering the current mesh based on the error estimate and other factors.

This section will discuss these two topics and present the primary mesh refinement algorithm used by TOPS. A more recent algorithm that could potentially outperform it is also introduced, although it has not been fully implement in TOPS. In addition, several techniques that have been developed to improve mesh refinement for non-smooth problems are introduced and their merits and drawbacks are discussed.

5.2.1 *hp* and *ph* Mesh Refinement

Mesh refinement methods are typically classified as *h* methods or *p* methods [18, 16, 83]. The *h* methods have a long history of use with Runge-Kutta and Euler methods. They use a fixed-degree polynomial to approximate the state in each mesh segment. Convergence to a high-accuracy solution is achieved by splitting a segment in which the error is high into multiple mesh segments [18]. The advent of *p* methods coincided with the development of PS methods [131]. Original implementations approximated a single segment using a globoal polynomial with numerous support/collocation points. Error reduction is achieved in a *p* method by increasing the degree of the interpolating polynomial. For a PS method, this is synonymous with increasing the number of collocation points. For problems with smooth solutions, this method converges at an exponential rate [54].

However, both *h* and *p* methods have their drawbacks. In the case of an *h* method, an extremely fine mesh may be required to achieve the desired accuracy. With a *p* method, an accurate solution may require the use of an unreasonably large polynomial approximation. The solution presented in recent years is to *combine* the two methods into an adaptive mesh-refinement algorithm. These *hp* and *ph* methods allow the degree of the approximating polynomial and the number of mesh segments to vary [10]. In an *hp* method, mesh segment division is given priority, followed by increasing polynomial degree if necessary. In a *ph* method, the order is reversed, and an increase in polynomial degree is prioritized.

A *ph*-Adaptive Mesh Refinement Algorithm

The method presented here was introduced by Patterson et al. [131]. The error estimate used in this method is based on the Euler Runge-Kutta scheme (an RK12 scheme) that approximates the absolute error in the solution of $\dot{y}(t) = f(y(t))$ as,

$$E = |\hat{y}_{t+1} - y_{t+1}|, \tag{5.4}$$

$$y_{t+1} = y_t + hf(y_t), \tag{5.5}$$

$$\hat{y}_{t+1} = y_t + hf(\bar{y}), \tag{5.6}$$

$$\bar{y} = y_t + \frac{h}{2}f(y_t). \tag{5.7}$$

The approach taken in this algorithm is very similar. For the sake of clarity, a single-segment scheme will be considered. However, this exact process can be performed on each mesh segment with no modification. Consider a state and control solution obtained for N LGR collocation points, denoted $\tau = \{\tau_1, \tau_2, \ldots, \tau_N\}$. Note that Patterson et al. [131] and Garg et al. [60] show that errors in the state are tightly coupled with errors in the costate, so applying this algorithm to the costate does not improve the error estimate. Thus, we want to estimate the error in the state *only* at $M = N + 1$ LGR points given by $\hat{\tau} = \{\hat{\tau}_1, \hat{\tau}_2, \ldots, \hat{\tau}_M\}$. Let the state and control be interpolated at the M LGR points plus a non-collocated $M + 1$ point using,

$$\boldsymbol{X}(\hat{\tau}_i) = \sum_{j=1}^{N+1} \boldsymbol{x}_j \ell_j(\hat{\tau}_i), \quad \boldsymbol{U}(\hat{\tau}_i) = \sum_{j=1}^{N} \boldsymbol{u}_j \ell_j(\hat{\tau}_i), \quad (i = 1, \ldots, M). \tag{5.8}$$

It is worth mentioning that many Lagrange interpolation algorithms are freely available to download. Additionally, the control is *arbitrarily* interpolated using Lagrange polynomials in Eq. (5.8). In the experience of the author, this method works best. Note in Eq. (5.8) that although we interpolate at M points, the support points are still the N LGR points. Next, we use the implicit integral form of the LGR collocation method to construct a better estimate of the state as,

$$\hat{\boldsymbol{X}}(\hat{\tau}_{i+1}) = \boldsymbol{x}_1 + \frac{t_f - t_0}{2}\boldsymbol{A}\boldsymbol{f}\left[\boldsymbol{X}(\hat{\tau}_i), \boldsymbol{U}(\hat{\tau}_i), \hat{\tau}_i\right], \quad (i = 1, \ldots, M), \tag{5.9}$$

where $\boldsymbol{A}$ is an LGR integration matrix of size $M \times M$. Note here the difference between $\hat{\boldsymbol{X}}(\hat{\tau}_i)$, which is obtained by collocation, and $\boldsymbol{X}(\hat{\tau}_i)$, which is simply interpolated. We may construct the errors in the i-th component of the state as,

$$\boldsymbol{E}(\hat{\tau}_i) = \left|\hat{\boldsymbol{X}}(\hat{\tau}_i) - \boldsymbol{X}(\hat{\tau}_i)\right|, \qquad (i = 1, \ldots, M+1), \tag{5.10}$$

$$\boldsymbol{e}_{\text{rel}}(\hat{\tau}_i) = \frac{\boldsymbol{E}(\hat{\tau}_i)}{1 + \max\limits_{j \in 1,\ldots,M+1} |\boldsymbol{X}(\hat{\tau}_j)|}, \qquad (i = 1, \ldots, M+1), \tag{5.11}$$

where $\boldsymbol{E}(\hat{\tau}_i) \in \mathbb{R}^{1 \times N_x}$ is the absolute error and $\boldsymbol{e}_{\text{rel}}(\hat{\tau}_i) \in \mathbb{R}^{1 \times N_x}$ is the relative error. The division in Eq. (5.11) is performed element-wise, as we are dividing an $N_k \times N_x$ matrix by an $1 \times N_x$ row vector resulting from the $\max(\cdot)$ operation. The maximum relative error is defined as,

$$e_{\max} = \max_{\substack{j \in 1,\ldots,N_x \\ i \in 1,\ldots,M+1}} e_j(\hat{\tau}_i), \tag{5.12}$$

where $e_j(\hat{\tau}_i)$ denotes the i-th row and j-th column of $\boldsymbol{e}_{\text{rel}}$. Based on a bound to the error, Patterson et al. [131] show that the number of points that should be added to a segment for a *target* mesh error ε should be,

$$P = \left\lceil \log_N \left(\frac{e_{\max}}{\varepsilon} \right) \right\rceil, \tag{5.13}$$

where ε is the tolerance on the maximum relative error chosen by the user and $\lceil \cdot \rceil$ means to round the argument towards $+\infty$ to the nearest integer. Thus, for a given mesh interval, the new number of points $\bar{N}$ is given by,

$$\bar{N} = N + P. \tag{5.14}$$

Note that a logarithmic function with an arbitrary base can be expressed as,

$$\log_N(x) = \frac{\ln(x)}{\ln(N)}. \tag{5.15}$$

Next, let the user define the parameters $N_{\min}$ and $N_{\max}$ which are the minimum and maximum polynomial degrees for the segment, respectively. If $\bar{N} > N_{\max}$, then the segment is divided into B subintervals, given by,

$$B = \max\left(\left\lceil \frac{\bar{N}}{N_{\min}} \right\rceil, 2\right). \tag{5.16}$$

Each new segment is set to have $N_{\min}$ collocation points. This ensures that each new segment will contain $N_{\min}$ collocation points and the sum of these collocation points is given by $BN_{\min}$. The full algorithm is given below.

1. Generate an initial mesh with K segments with N_k points in each segment.

2. Generate the PS elements for the current mesh.

3. Solve the problem on the current mesh.

4. Set $k = 1$. Begin iterating over the mesh segments.

   4.1. Interpolate the state and control at $M^{(k)} = N^{(k)} + 1$ points.

   4.2. Calculate $e_{\max}^{(k)}$ for the current mesh segment using Eq. (5.12).

   4.3. If $e_{\max}^{(k)} > \varepsilon$, continue. Otherwise, go to Step 4.6.

   4.4. For the current mesh segment, calculate $\bar{N}$ and B.

   4.5. If $\bar{N} > N_{\max}$, divide the current segment into B even segments and set the number of collocation points in each segment to $N_{\min}$. Otherwise, set the number of points in the segment to $\bar{N}$.

   4.6. Let $k \leftarrow k + 1$.

5. If $e_{\max}^{(k)} < \varepsilon$ in every mesh segment, quit and return the current solution to the user. Otherwise, continue.

6. Return to Step 2.

This concludes the ph-adaptive mesh refinement algorithm. To use this algorithm in TOPS, set `p.mesh.algorithm = "ph"`.

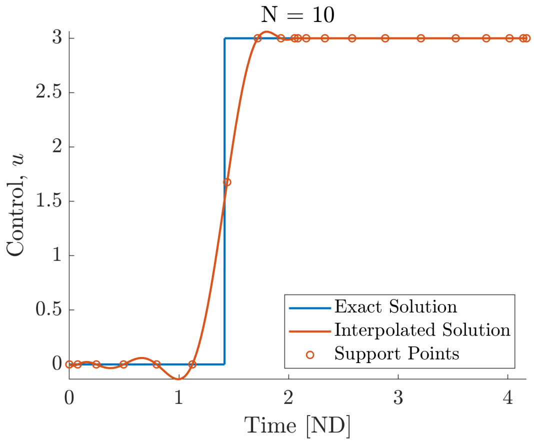
5.2.2 Discontinuity Detection

Discontinuity detection is a topic that has been studied extensively in the literature of PS optimal control. Discontinuity detection is also an important step for direct methods that use a polynomial basis to approximate the states. Thus, they are important for PS methods. To illustrate the importance of discontinuity detection, consider the following story that is based on true events.

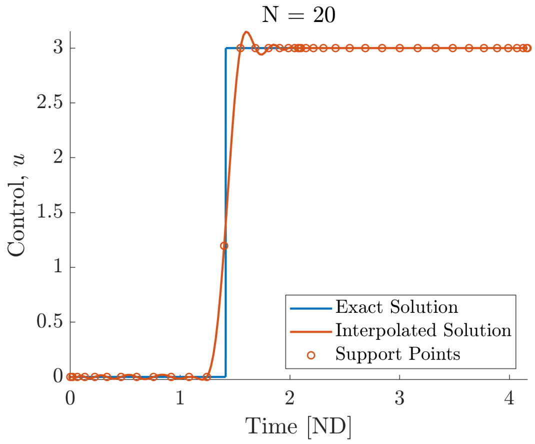
An over-enthusiastic young graduate student has just finished coding their own PS optimization software and decides to test it out on the apparently simple problem presented in Section 6.1. This problem is ideal, as it has a closed-form solution and exhibits a single bang-on arc control profile. The graduate student inputs the problem parameters into their new software and designs a mesh with two segments and 10 collocation points in each. They click "run," and to their excitement, the solver converges to a solution. The obtained solution is shown in Fig. 5.1a. The student notices the support points they chose lie on or nearly on the exact solution. This is encouraging and a good sign of convergence. However, they also notice that the interpolated solution is inaccurate. Enlightened by a stroke of genius, the student decides to double the number of collocation points in each segment. They run the solver again, and obtain another solution that is shown in Fig. 5.1b. Disappointingly, the results are not much better. Finally, in a fit of rage, the student doubles the number of points again. To their chagrin, the result they obtained in Fig. 5.1c is worse than the previous one! The student gives up and goes home for the day. After a good night of rest, the student realizes that if they place a new knot at the exact location of the discontinuity, then they will not have to interpolate over the discontinuity! From the exact solution, they know that the thrust turns on at,

$$t = 1.41516\ [\mathrm{TU}]. \tag{5.17}$$

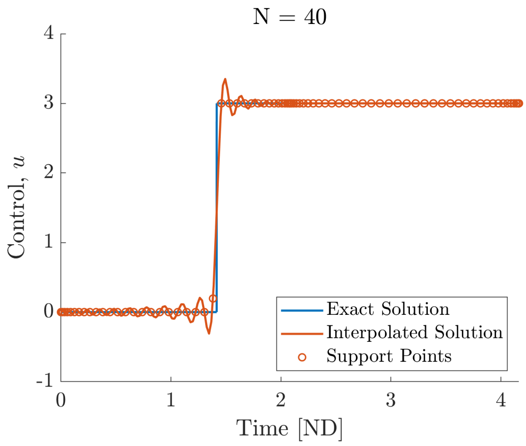
They manually place a mesh knot at exactly that time, and run the algorithm. The results are shown in Fig. 5.2. Needless to say, the graduate student is thrilled. However, they will not always have a closed-form solution to every problem to examine. If they did, there would be no need for their new software. Thus, they realize that their software requires an algorithmic



(a) Control Profile for $N = 10$



(b) Control Profile for $N = 20$



(c) Control Profile for $N = 40$

Figure 5.1: Moon lander control profiles exhibiting oscillation.

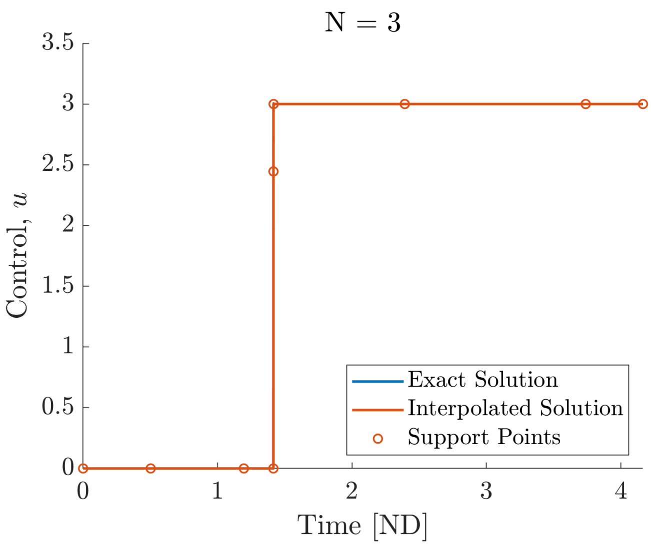


Figure 5.2: Solution for a manually-placed Knot.

way to detect the location of these discontinuities. Short of this, it needs a way to approximate their location and place more knots in the vicinity of discontinuities. The simplest solution is to use the mesh refinement algorithm presented in Section 5.2.1. This algorithm works quite well at locating and "bracketing" a discontinuity. The solution for the problem is shown in Fig. 5.3. Note that the solution is nearly identical to the true solution. However, the mesh refinement

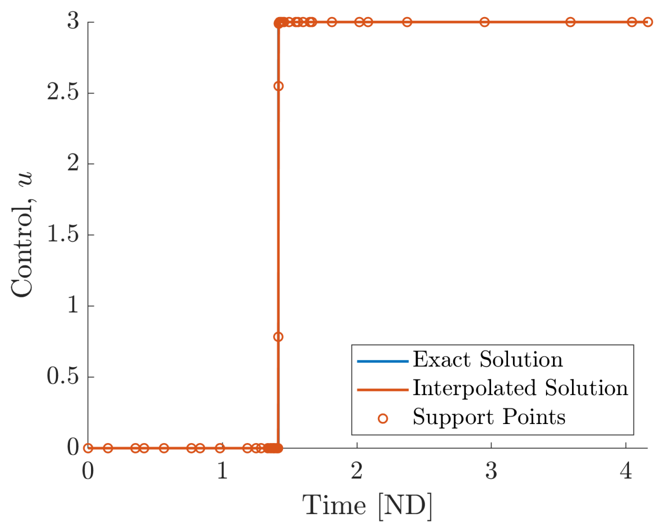


Figure 5.3: Solution using the *ph*-adaptive mesh-refinement.

algorithm uses many more points close to the discontinuity than the manual knotting method. Regardless, the *ph* adaptive method requires *no* user input other than defining the desired mesh error tolerance and the initial mesh parameters.

In the experience of the author, the *ph*-adaptive mesh refinement algorithm in Section 5.2.1 is the most successful mesh refinement algorithm that the author has implemented themselves. This conclusion was drawn after implementing several different mesh refinement algorithms. The reason for this is primarily because it is extremely difficult to create a "catch-all" mesh refinement algorithm that is efficient and can exactly place knots. However, the method presented in [131] has proven effective for a wide variety of problems that the author has solved, discontinuous or otherwise.

The next mesh-refinement algorithm that is included in TOPS was introduced by Liu et al. [101]. Consider two approximations of the state, given by,

$$x(\tau) \approx \sum_{i=0}^{N} \hat{a}_i p_i(\tau), \qquad x(\tau) \approx \sum_{i=1}^{N+1} x_i \ell_i(\tau), \tag{5.18}$$

where $p_i(\tau)$ is a basis of interpolating *Legendre* polynomials, $\hat{a}_i$ are the Legendre polynomial coefficients, $\ell_i(\tau)$ are the *Lagrange* basis polynomials, and x_i are the values of the states at the support points. The premise of this mesh-refinement algorithm is that the decay rate of the Legendre polynomial coefficients can be used to indicate whether a function is smooth or discontinuous [186]. Mavriplis [113] and Liu et al. [101] show that the state degree of discontinuity in a mesh segment is estimated to be the decay rate of the Legendre polynomial coefficients in the mesh interval. The decay rate $\sigma > 0$ is obtained using an exponential least-squares fit of the form [113],

$$\hat{a}_i = c10^{-\sigma i}, \quad (i = 0, 1, \ldots, N). \tag{5.19}$$

Note that Eq. (5.19) should be evaluated at $(i = 0, 1, \ldots, N)$. The Legendre polynomial coefficients $\hat{a}_i$ for state $x \in \mathbb{R}^{(N+1)\times 1}$ can be approximated as,

$$\begin{bmatrix} \hat{a}_0 \\ \hat{a}_1 \\ \vdots \\ \hat{a}_N \end{bmatrix} = \begin{bmatrix} p_0(\tau_1) & p_1(\tau_1) & \cdots & p_N(\tau_1) \\ p_0(\tau_2) & p_1(\tau_2) & \cdots & p_N(\tau_2) \\ \vdots & \vdots & \ddots & \vdots \\ p_0(\tau_{N+1}) & p_2(\tau_{N+1}) & \cdots & p_N(\tau_{N+1}) \end{bmatrix}^{-1} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N+1} \end{bmatrix}, \tag{5.20}$$

Using Eqs. (5.19) and (5.20), we may use any least-squares fitting method to solve a root-finding sub-problem to determine σ. Assume that for a mesh segment, the error approximate given by Eq. (5.12), $e^{(k)}_{\max}$, is greater than the user-defined tolerance. If σ for the segment is greater than a user-defined threshold $\overline{\sigma}$, then, the mesh interval is considered *smooth*. If $\sigma \leq \overline{\sigma}$, the segment is considered *non-smooth* and the approximation is improved by dividing the mesh segment. The new number of points in a smooth mesh segment is given by,

$$\bar{N} = N + \left\lceil \frac{\log_{10}\left(\frac{e_{\max}}{\varepsilon}\right)}{\sigma} \right\rceil, \tag{5.21}$$

where N is the previous number of points and ε is the mesh error tolerance. The number of segments that a non-smooth mesh segment should be divided into is,

$$H = \left\lceil \frac{\bar{N}}{N} \right\rceil. \tag{5.22}$$

The mesh-refinement algorithm based on the decay rate is given as follows.

1. Generate and initial mesh with K segments and N_k points in each segment.
2. Generate the PS elements for the current mesh.
3. Solve the problem on the current mesh.
4. Set $k = 1$ and begin iterating over the mesh segments.
   - 4.1. Calculate $e_{\max}^{(k)}$ for the segment according to Eq. (5.12).
   - 4.2. If $e_{\max}^{(k)} > \varepsilon$, continue. Otherwise, go to Step 4.8.
   - 4.3. For the current mesh segment, calculate σ, $\bar{N}_k$, and $H^{(k)}$. Calculate $B^{(k)}$ according to Eq. (5.12).
   - 4.4. If $\sigma^{(k)} > \bar{\sigma}$ and $\bar{N}_k < N_{\max}$, set the number of points in the segment to $\bar{N}_k$.
   - 4.5. If $\sigma^{(k)} > \bar{\sigma}$ and $\bar{N}_k > N_{\max}$, divide the segment into $B^{(k)}$ evenly spaced segments.
   - 4.6. If $\sigma^{(k)} < \bar{\sigma}$, divide the mesh into $H^{(k)}$ evenly spaced segments.
   - 4.7. Set $N_k = N_{\min}$ in all newly created segments.
   - 4.8. Let $k \leftarrow k + 1$.
5. If $e_{\max}^{(k)} < \varepsilon$ in every segment, quit and return the solution to the user.
6. Otherwise, go to Step 2.

This algorithm is referred to as the *hp*-legendre adaptive mesh refinement algorithm. To use this algorithm in TOPS, set `p.mesh.algorithm = "hp-legendre"`.

It is worth noting that Gong et al. [67] refers to the Legendre polynomial coefficients as the *spectral* coefficients. These coefficients can be obtained for the states, controls, and costates using Eq. (5.20) by replacing the state coefficients with the desired values. Gong et al. [67] use the norm of the spectral coefficients as an error measure. If the maximum norm of these coefficients is less than some value, the algorithm stops. This is referred to as the Jackson stopping criterion. However, the authors of [67] do not recommend using this stopping criterion for discontinuous problems. As such, it is not incorporated in TOPS.

Knotting Methods

The third and final mesh-refinement algorithm that is implemented in TOPS is based on an idea presented by Ross and Fahroo [148] and Gong et al. [67]. In these papers, the PS differentiation matrix is used to determine the location of discontinuities in the control. Assume the control is approximated as,

$$\boldsymbol{U}(\tau) = \sum_{i=1}^{N} \boldsymbol{u}_i \ell_i(\tau). \tag{5.23}$$

This is an arbitrary choice of interpolation. Collocating at the LGR points, we can approximate the control derivative using the PS differentiation matrix $\boldsymbol{D}_N = \boldsymbol{D}_{(:,1:N)}$ as,

$$\dot{\boldsymbol{U}} = \boldsymbol{D}_N \boldsymbol{U}. \tag{5.24}$$

However, if we consider the normalization,

$$\dot{\bar{\boldsymbol{U}}} = \frac{|\boldsymbol{D_N U}|}{\max\limits_{\substack{j \in 1,\ldots,N_u \\ i \in 1,\ldots,N}} |\boldsymbol{D_N U}|_{ij}}, \tag{5.25}$$

we ensure that $\dot{\bar{\boldsymbol{U}}} \in [0,1]$. Here, $|\cdot|$ denotes the absolute value of the matrix. Consider the moon-lander problem presented in Section 6.1. TOPS was used to solve the problem on a mesh with 10 segments and three points in each segment. Fig. 5.4 shows a normalized rough control solution as well as the value of Eq. (5.25) for the control solution. Fig. 5.4 shows that the

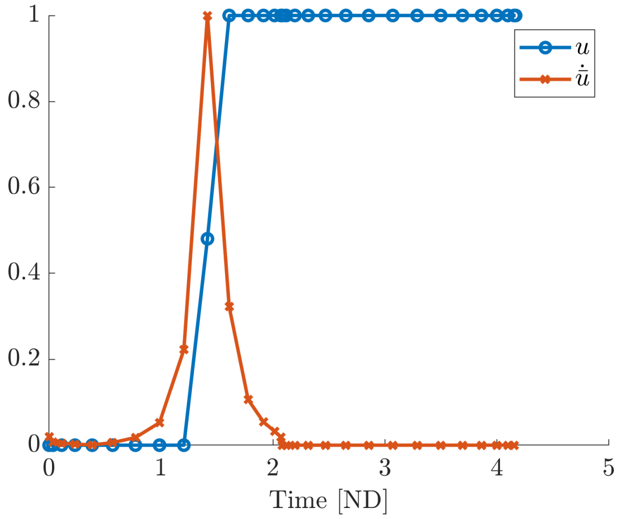


Figure 5.4: Control derivative approximation.

control switch instant is indicated or "captured" by Eq. (5.25). As such, this method can be used in a mesh-refinement algorithm to determine whether a segment should be divided. The mesh-refinement algorithm is given below.

1. Generate an initial mesh and select a user-defined derivative threshold, $\sigma \in [0, 1]$.

2. Generate the PS elements for the current mesh.

3. Solve the problem on the current mesh.

4. Begin iterating over mesh segments.

5. Get the matrix value $\dot{\boldsymbol{U}}$ using the *global* differentiation matrix.

6. Set $k = 1$ and begin iterating over mesh segments.

    6.1. Calculate $e_{\max}^{(k)}$ according to Eq. (5.12).

    6.2. If $e_{\max}^{(k)} < \varepsilon$, skip this iteration. Otherwise, continue.

    6.3. Find all the points in k for which $\dot{\boldsymbol{U}}^{(k)}(\tau_i^{(k)}) \geq \sigma$.

        i. If this condition is true for adjacent points, place the new knot at the midpoint of those points.

ii. If this condition is true at the first point in the segment (i.e., an existing knot), place the new knot halfway between the existing knot and the last point of the previous segment.

iii. If this condition is true at $\tau_1^{(1)} = -1 = t_0$ or $\tau_{N+1}^{(K)} = +1 = t_f$, then divide the mesh segment in the middle.

6.4. If $\dot{\bar{\boldsymbol{U}}}^{(k)}(\tau_i^{(k)}) < \sigma$ for all points in the current segment, increase points/divide according to the *ph*-adaptive mesh refinement found in Section 5.2.1.

6.5. Set the number of points in all newly created mesh segments equal to $N_{\min}$.

6.6. Let $k \leftarrow k+1$.

7. If $e_{\max}^{(k)} < \varepsilon$ for all k, quit and return to the user.

8. Return to Step 2.

The author has dubbed this algorithm an *hph*-adaptive mesh-refinement algorithm. To use this algorithm in TOPS, set `p.mesh.algorithm = "hph"`. After thorough testing, it seems to be less effective than the error-based algorithm given in Section 5.2.1. However, for the sake of variety, it was not removed from TOPS. Should they feel so inclined, the reader is encouraged to improve upon this algorithm. It is worth noting that Gong et al. [67] suggests applying Eq. (5.25) to states in order to detect discontinuities. However, when applying the operation $\boldsymbol{D}\boldsymbol{X}$, we simply obtain $\boldsymbol{f}$. If we instead apply the operation given by,

$$\dot{\bar{\boldsymbol{f}}} = \frac{|\boldsymbol{D}_{\boldsymbol{N}}\boldsymbol{f}|}{\max\limits_{\substack{j \in 1,\dots,N_x \\ i \in 1,\dots,N}} |\boldsymbol{D}_{\boldsymbol{N}}\boldsymbol{f}|_{ij}}, \tag{5.26}$$

we obtain *new information* about any discontinuities in the state, as we are approximating the acceleration or jerk profile of the dynamics. In addition, this method can detect discontinuities that are not due to controls, much like the method given in Section 5.2.1. Consider the result of this operation when applied to the Moon-Lander problem in Fig. 5.5. It can be seen that we recover the control derivative profile. However, for more complex problems, this method can fail to produce a mesh division due to a poorly behaved outlier derivative. Numerically, this

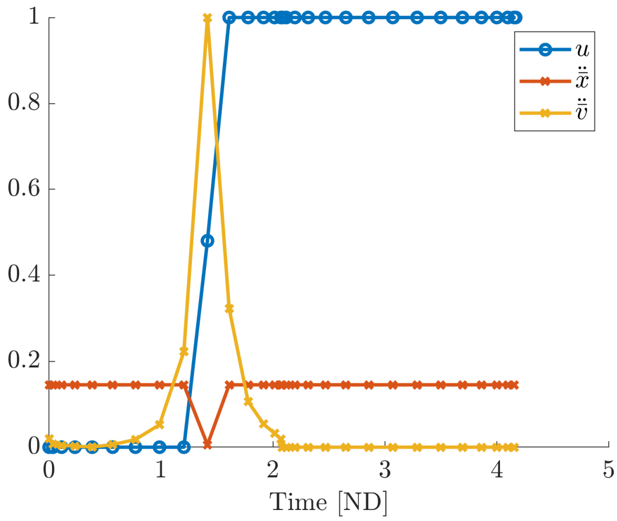


Figure 5.5: Second State Derivative

means that there is some large value in $|\boldsymbol{D}\boldsymbol{f}|$ that reduces the magnitude of other discontinuities that may contribute to solution error.

The final topic that is presented is an *explicit* knotting method. By this, the author means that the locations of the knots are considered to be optimization variables. Consider the LGR PS collocation and quadrature equations for the multi-interval method given by,

$$\boldsymbol{D}\begin{bmatrix}\boldsymbol{X}\\ \boldsymbol{x}_f\end{bmatrix} = \frac{t_f - t_0}{2}\boldsymbol{\sigma}\boldsymbol{f}(\boldsymbol{X},\boldsymbol{U},\boldsymbol{t}), \tag{5.27}$$

$$J\left[\boldsymbol{X},\boldsymbol{x}_f,\boldsymbol{U},t_0,t_f\right] = E(\boldsymbol{x}_1^{(1)},\boldsymbol{x}_f,t_0,t_f) + \frac{t_f - t_0}{2}\boldsymbol{w}^\top\boldsymbol{\sigma}F\left(\boldsymbol{X},\boldsymbol{U},\boldsymbol{t}\right), \tag{5.28}$$

where,

$$\boldsymbol{\sigma} = \begin{bmatrix}\sigma_1\mathbf{1}_{N_1\times 1}\\ \sigma_2\mathbf{1}_{N_2\times 1}\\ \vdots\\ \sigma_K\mathbf{1}_{N_K\times 1}\end{bmatrix}, \tag{5.29}$$

and

$$\sigma_k = \frac{T_k - T_{k-1}}{2}, \quad (k = 2,\ldots,K). \tag{5.30}$$

Here, K is the total number of segments. If we let the interior mesh points T_k, where $(k = 2,3,\ldots,K-1)$ be optimization variables, the solver should (in theory) automatically place the knots at their optimal locations. The new optimization vector is,

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{X}_{(:)} \\ \boldsymbol{U}_{(:)} \\ t_0 \\ t_f \\ \boldsymbol{T} \end{bmatrix}, \tag{5.31}$$

where we define the column vector $\boldsymbol{T}$ to be,

$$\boldsymbol{T} = [T_1, T_2, \ldots, T_{K-1}]^\top. \tag{5.32}$$

Two additional constraints are necessary. They are given by,

$$\sum_{k=1}^{K} \sigma_k = 1, \tag{5.33}$$

$$\sigma_k > 0, \tag{5.34}$$

for $(k = 1, \ldots, K)$. The equality constraint in Eq. (5.33) ensures that the derivative of the scaling factor remains equal to 1 so that the knots remain in the interval $[-1, 1]$. This, combined with the constraint in Eq. (5.34), ensures that the knots remain in ascending order. In theory and for some simple problems, this method works shockingly well. However, as problems become more difficult with multiple rapidly changing states and controls, it is the author's experience that this particular form of knotting method fails completely. Knots tend to "squeeze" themselves together at arbitrary points, creating dense mesh segments that degrade the solution quality. In practice, post-optimization knotting is safer, more stable, and less sensitive to the initial mesh.

Anti-Aliasing and Testing Optimality

An interesting feature in direct optimal control methods is that the converged solution is often an alias of the true continuous-time solution sampled at the support points of the direct solution

[150]. Consider again the Moon-Lander problem given in Section 6.1. The problem is solved using TOPS for two even mesh intervals with 15 points in each. Fig. 5.6 shows a control solution that exhibits *aliasing*. In this solution, the control solution support points lie exactly on

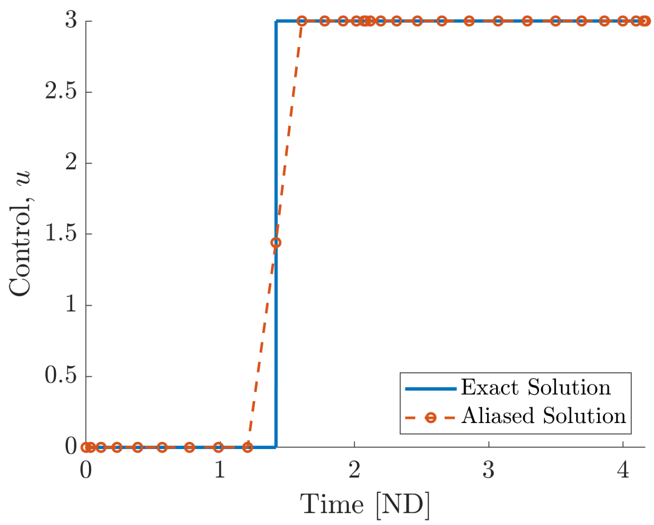


Figure 5.6: Control solution exhibiting aliasing.

the true solution. However, if one interpolates the direct solution (see the dashed orange line in Fig. 5.6), the true optimal control is not recovered. Ross et al. [150] develop a mesh-refinement algorithm using this principle. The algorithm first breaks the solution into segments and obtains a low-grade solution on a rough grid. Following this, the solution is used to construct a continuous-time control polynomial using linear or spline interpolants. This is then used to propagate the initial dynamics to the next phase, upon which the process is repeated.

Although this mesh-refinement method is intended primarily as a computationally cheap on-board optimization algorithm, it does introduce a very useful self-test for feasibility and optimality in lieu of an indirect solution. This test can be performed with the following steps.

1. Solve the problem to a requested mesh accuracy.
2. Using the discrete control solution, reconstruct the continuous-time control using arbitrary interpolation.
3. Numerically propagate the equations of motion from the initial conditions using a time-marching integrator (such as `ode45` in MATLAB) to the final time.

4. Determine the absolute difference between the design boundary conditions and the final states obtained through propagation, $\epsilon = \left| \boldsymbol{x}_f - \boldsymbol{x}^f \right|$.

5. If the residuals calculated are within an acceptable engineering threshold, the solution is verifiably feasible.

Ross et al. [151] discuss the necessity of this feasibility check for flight applications. In general, a residual on the order of 10^{-3} denotes a flight-worthy solution using this optimality check.

To conclude this section, a short survey of useful papers covering mesh-refinement algorithms for PS methods is presented, should the reader decide to implement their own algorithm. Ross and Fahroo [148] presented an early attempt at algorithmic mesh refinement for solving discontinuous problems using PS methods that implements the differentiation matrix. This work was the inspiration for the *hph*-adaptive mesh-refinement algorithm. More recently, Koeppen et al. [92] and Ross and Proulx [144] introduced the use of Birkhoff basis functions as an evolution on the PS method that is insensitive to the size of the problem. Darby et al. [34] and Patterson et al. [131] introduced two *hp*- and *ph*-adaptive mesh-refinement algorithms that are specifically designed for PS methods and very capable of capturing discontinuities. The *ph*-adaptive mesh-refinement algorithm used by TOPS is directly adapted from Patterson et al. [131]. Liu et al. [100] introduces an *hp* algorithm that uses the second derivative of the state to approximate the location of discontinuities. Liu et al. [101] uses the decay rates of Legendre polynomial coefficients in order to approximate the locations of discontinuities. This algorithm was the inspiration for the *hp*-Legendre adaptive mesh-refinement algorithm used in TOPS. Agamawi et al. [4] and Pager and Rao [123] use the covector mapping theorem and exploit the separability of the Hamiltonian to obtain a general form of the PMP switching function using jump functions or hyper-dual differentiation [3]. This general switching function is then used to approximate the locations of the control switching locations. In addition, it can be used to determine the presence and location of singular arcs.

5.3 Calculating Derivatives

One of the critical and most effective ways to increase the convergence performance of any NLP solver is to provide analytical derivatives of the objective and constraints with respect to the decision variables [43, 44]. All state-of-the-art gradient-based NLP solvers require first and/or second derivatives of the NLP functions to be provided. There are generally two types of NLP solvers: 1) Quasi-Newton NLP solvers, which use only first derivatives of the objective and constraint functions [129]. These solvers typically approximate the second derivatives using a *quasi-Newton* approximation algorithm. Examples of popular quasi-Newton solvers include SNOPT [65] and NPSOL [64], both of which are available in several programming languages. 2) Newton NLP solvers, which require first *and* second derivatives. The objective gradient and constraint Jacobian are calculated in addition to the Hessian (second derivative) of the Lagrangian.

Consider the general form of an NLP problem as

$$\text{(NLP)}\begin{cases} \text{Minimize} & J(\boldsymbol{z}), \\ \text{Subject to}: & \boldsymbol{c}(\boldsymbol{z}) \leq 0, \\ & \boldsymbol{c}_{\text{eq}}(\boldsymbol{z}) = \boldsymbol{0}. \end{cases}$$

In most NLP solvers, the Lagrangian is defined as,

$$\mathcal{L} = \sigma J(\boldsymbol{z}) + \boldsymbol{\Gamma}^{\top}\boldsymbol{c}(\boldsymbol{z}) + \boldsymbol{\Lambda}^{\top}\boldsymbol{c}_{\text{eq}}(\boldsymbol{z}), \tag{5.35}$$

where $\boldsymbol{z}$ is the column vector of decision variables, $J(\boldsymbol{z})$ is the cost, $c(\boldsymbol{z})$ is the column vector of inequality constraints, and $c_{\text{eq}}(\boldsymbol{z})$ is the column vector of equality constraints. Additionally, $\boldsymbol{\Gamma}$ and $\boldsymbol{\Lambda}$ are the KKT multipliers associated with $\boldsymbol{c}$ and $\boldsymbol{c}_{\text{eq}}$, respectively. The multiplier σ is an NLP solver input used to extract the objective and constraint terms individually. Typically, analytical second-derivatives are supplied to Newton solvers, which results in great increases in computational speed and accuracy over the approximations used by quasi-Newton solvers. Two popular second-derivative solvers include IPOPT [115, 19] and KNITRO [24]. Since every

NLP solver requires the derivatives of the problem functions, it is evident that providing those derivatives in an efficient and accurate manner is extremely important to convergence. This importance is compounded by the fact that the majority of time required for an NLP solver to obtain a solution is spent computing derivatives [3]. Thus, the efficiency aspect of computing derivatives is incredibly important to the computational feasibility of large OCPs.

The PS optimization software TOPS is capable of using first and second derivatives. Since it is primarily for academic and educational use, the solver aims to be as accessible and "ready-to-go" upon download in its final version. Thus, the NLP solver that is provided with TOPS is MATLAB's fmincon NLP solver. This solver will automatically switch between a sequential-quadratic-programming (SQP) algorithm if using finite differences or first derivatives and an interior-point algorithm if using second derivatives. However, TOPS is capable of using both SNOPT and IPOPT if installations are found on the MATLAB path. Once installed, switching between these solvers is as simple as changing the flag, `p.opts.solver = 'fmincon'` to `p.opts.solver = 'ipopt'` or `p.opts.solver = 'snopt'`. Currently, all analytical first and second derivatives are *automatically* provided to the solver by the open-source MATLAB-based automatic differentiation (AD) software *ADiGator* [190]. Currently, IPOPT is recommended for two reasons.

1. IPOPT is open source and free to download, so any user with a MATLAB license may download and use it with TOPS. The author recommends using Dr. Enrico Bertolazzi's re-write of the IPOPT MATLAB interface. It is up-to-date and compatible with recent releases of MATLAB. Instructions to download and install it are available at [13].

2. IPOPT will generally out-perform SNOPT in vectorized second-derivative mode. Note that if only first derivatives are currently supplied, fmincon's SQP algorithm will marginally outperform IPOPT in first derivative mode and SNOPT will drastically outperform IPOPT in first derivative mode. The choice is up to the user, since first and second derivatives are automatically generated by TOPS by setting `p.opts.derivative_level` to `1` for first derivatives and `2` for second derivatives. Setting it to `0` will use finite differences. Further work to increase the efficiency of the second derivative generation is currently ongoing.

The remainder of this section will discuss several different methods of calculating the derivatives of the NLP functions. The accuracy of each method will be compared and the advantages and disadvantages of each will be highlighted. Finally, the analytical first-derivatives of the NLP functions and their sparsity patterns are presented.

5.3.1 Finite Differences

The easiest way of computing derivatives is the finite-difference method. This method is employed by most NLP solvers in the absence of user-supplied derivative information. Consider a real-valued function, $f(x) : \mathbb{R} \to \mathbb{R}$. Consider the n-th order Taylor expansion of $f(x+h)$ about x, an arbitrary point in the domain of f, where h is assumed to be small.

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \cdots + \frac{h^n}{n!}f^{(n)}(x) + \mathcal{O}(h^{n+1}), \quad (5.36)$$

where $f^{(n)}(x)$ denotes the n-th derivative of $f(x)$. We can rearrange Eq. (5.36) as follows.

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h^2). \quad (5.37)$$

Truncating any term greater and/or equal to the second-order terms of the Taylor series by assuming a small h, we obtain,

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}. \quad (5.38)$$

This is the well-known *forward finite difference* derivative approximation. We can obtain an expression for the central finite difference formula using a similar approach as

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}. \quad (5.39)$$

Expressions for second derivatives are obtained as,

$$f''(x) \approx \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2}, \qquad \text{(Forward)}, \tag{5.40}$$

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}, \qquad \text{(Central)}. \tag{5.41}$$

Note that the finite difference expressions are *only* an approximation that theoretically becomes exact as $h \to 0$, since we ignore higher-order terms in the expansion (see Eq. (5.36)). However, this is not the case in practice. Not only are finite difference approximations subject to truncation error, they are also subject to the machine roundoff error caused by the addition and/or subtraction operations performed on f in the numerator. Roundoff or subtractive cancellation error occurs when two numbers become so close to each other that a computer cannot tell them apart due to the finite number of bits representing the number [120]. Thus, a computer cannot tell the difference between $f(x)$ and $f(x+h)$ for a small h. In practice, this means that as h is reduced, there is a lower bound to the approximation error. Decreasing h any lower will *increase* the error [3] as subtractive cancellation errors begin to dominate. Thus, one must select h according to,

$$h = h_{\mathcal{O}(1)}(1 + |x|), \tag{5.42}$$

where x is the magnitude of the independent variable of interest and $h_{\mathcal{O}(1)}(\cdot)$ is a function that selects the optimal step size for a function whose input and output are $\mathcal{O}(1)$. More information can be found in [66].

5.3.2 Hyper-Complex Differentiation

The next type of differentiation method that will be discussed is hyper-complex differentiation. This method is usually divided into bi-complex step and hyper-dual differentiation methods. However, hyper-dual numbers are an extension or generalization of complex numbers [53], so they are considered together here. First, consider the Taylor series expansion using an imaginary step value, ih, where $i^2 = -1$.

$$f(x+ih) = f(x) + ihf'(x) - \frac{h^2}{2!}f''(x) - i\frac{h^3}{3!}f'''(x) + \cdots. \tag{5.43}$$

We can separate Eq. (5.43) into its real and imaginary parts as

$$f(x+ih) = \left(f(x) - \frac{h^2}{2!} f''(x) + \cdots \right) + ih \left(f'(x) - \frac{h^2}{3!} f'''(x) + \cdots \right). \tag{5.44}$$

Re-grouping and combining higher-order terms, we obtain the following expression.

$$f(x+ih) = f(x) + ihf'(x) + \mathcal{O}(h^2). \tag{5.45}$$

Thus, we see that the first derivative is given by the imaginary part of Eq. (5.45) divided by the step size, h. Truncating the higher-order terms, we obtain,

$$f(x) = \text{Real}[f(x+ih)], \qquad f'(x) \approx \frac{\text{Imag}[f(x+ih)]}{h}. \tag{5.46}$$

Eq. (5.46) is referred to as the bi-complex step derivative approximation. Truncation error is reduced by reducing the step size h. However, note that there are no additions or subtractions performed on f in the numerator. Thus, the only source of error is truncation error and reducing h to an arbitrarily small number *does not introduce subtractive cancellation error*. Thus, bi-complex step derivatives are accurate to machine precision for a sufficiently small h. Note that although Eq. (5.46) is not subject to roundoff error, the bi-complex arithmetic that is required to evaluate the function is still subject to roundoff error [94, 3]. This limits the step size from being too small. In practice, h typically takes values between 10^{-14} and 10^{-20}.

Next, consider the concept of a dual number, given by,

$$d = x + \epsilon_1 y, \tag{5.47}$$

where $d \in \mathbb{D}^1$, $x, y \in \mathbb{R} \equiv \mathbb{D}^0$, and ϵ_1 denotes the imaginary component of the dual number in the dual plane $\mathbb{D}^1$. This new imaginary number has the properties that $\epsilon_1^2 = 0$ and $\epsilon_1 \neq 0$. This can be thought of in a similar manner to $i^2 = -1$. If we consider a higher-order dual number plane (or a *hyper-dual* plane), we get,

$$w = d_1 + \epsilon_2 d_2, \tag{5.48}$$

where $d_1, d_2 \in \mathbb{D}^1$ and ϵ_2 designates an imaginary component that is distinct from the imaginary component in the ϵ_1 direction. This new imaginary direction shares the properties of the original in that $\epsilon_2^2 = 0$ while $\epsilon_2 \neq 0$. They also possess the commutative property, $\epsilon_1\epsilon_2 = \epsilon_2\epsilon_1$, and the nullity property, $(\epsilon_1\epsilon_2)^2 = 0$ [3]. Based on this definition, all powers of a Taylor series expansion of $f(x+h_1\epsilon_1+h_2\epsilon_2+0\epsilon_1\epsilon_2)$ containing third or higher derivatives become identically zero and truncate *exactly*. Thus, we have,

$$f(x + h_1\epsilon_1 + h_2\epsilon_2 + 0\epsilon_1\epsilon_2) = f(x) + h_1 f'(x)\epsilon_1 + h_2 f'(x)\epsilon_2 + h_1 h_2 f''(x)\epsilon_1\epsilon_2. \tag{5.49}$$

For a univariate function, we have the following expressions.

$$f(x) = \text{Real}[f(x + h\epsilon_1)], \qquad f'(x) = \frac{\text{Eps}_1[f(x + h\epsilon_1)]}{h}, \tag{5.50}$$

where Eps_1 denotes the component of the hyper-dual number associated with ϵ_1. For multivariate functions $f(\boldsymbol{x})$ where $\boldsymbol{x} \in \mathbb{R}^n$, we have the following expressions [3].

$$\frac{\partial f(x,y)}{\partial x} = \frac{\text{Eps}_1[f(x + h\epsilon_1, y + h\epsilon_2)]}{h}, \tag{5.51}$$

$$\frac{\partial f(x,y)}{\partial y} = \frac{\text{Eps}_2[f(x + h\epsilon_1, y + h\epsilon_2)]}{h}, \tag{5.52}$$

$$\frac{\partial^2 f(x,y)}{\partial x^2} = \frac{\text{Eps}_{1,2}[f(x + h\epsilon_1 + h\epsilon_2, y)]}{h^2}, \tag{5.53}$$

$$\frac{\partial^2 f(x,y)}{\partial y^2} = \frac{\text{Eps}_{1,2}[f(x, y + h\epsilon_1 + h\epsilon_2)]}{h^2}, \tag{5.54}$$

$$\frac{\partial^2 f(x,y)}{\partial x \partial y} = \frac{\text{Eps}_{1,2}[f(x, y + h\epsilon_1 + h\epsilon_2)]}{h^2}. \tag{5.55}$$

Here, Eps_1 and Eps_2 denote the imaginary parts associated with ϵ_1 and ϵ_2, respectively, while Eps_{12} denotes the imaginary part associated with $\epsilon_1\epsilon_2$. Due to the properties of hyper-dual arithmetic, these derivative equations avoid susceptibility to roundoff during the function evaluation [3]. Since there is no truncation error, these derivatives are *exact* for all step sizes. Thus, a step size of $h = 1$ may be taken arbitrarily. It is worth noting that hyper-dual differentiation needs only a single function evaluation to obtain the function value and its first and second derivatives. While this seems to imply that hyper-dual differentiation may require less evaluation time than finite differences, this is usually not the case due to the additional arithmetic involved with the hyper-dual operations. Hyper-dual numbers are generally slightly more expensive than a finite difference approximation, taking between one and 3.5 times more operations than central finite differencing [53]. However, they are substantially more accurate. An NLP solver using hyper-dual or bi-complex derivatives will generally converge in fewer iterations than a solver using finite differences. This may not be the case for bi-complex step methods, which may take even more arithmetic operations due to the complexity of bi-complex arithmetic [94]. MATLAB implementations of hyper-dual number classes can be found at [5] and [120].

Application of the bi-complex step derivative approximation to spacecraft low-thrust trajectory optimization is demonstrated in [172] for calculating the state transition matrix (STM) while regularizing the control. The STM is a method for mapping sensitivities along a *continuous* trajectory, which can be used to construct the requited derivative of the residuals with respect to the unknown initial costate values when a single-shooting scheme is used for solving TPBVPs using an indirect method. Proper computation of the STMs is an important step for solving challenging low-thrust trajectory optimization problems [132, 137, 138]. Application of the bi-complex step derivative approximation for calculating the contribution of the gravitational harmonics (up to any degree and order), which is needed for constructing the costates differential equations (in a numerical manner) is demonstrated in [167].

To conclude this section, we will compare the accuracy of the three derivative approximation methods introduced for first-order derivatives. Consider the relative error in some quantity x to be given by,

$$\epsilon_r = \frac{|x - \hat{x}|}{1 + |x|}, \tag{5.56}$$

where $\hat{x}$ is an approximation of x. The accuracy of the three methods is tested on the example function,

$$f(x) = \frac{x^2}{\sin(x) + xe^x}, \tag{5.57}$$

evaluated at $x = \pi/4$. The analytical derivative of Eq. (5.57) is given by,

$$f'(x) = \frac{2x\sin(x) - x^2\cos(x) + (x^2 - x^3)e^x}{[\sin(x) + xe^x]^2}. \tag{5.58}$$

The hyper-dual class implemented [5] was used to calculate the hyper-dual derivatives. It can be seen in Fig. 5.7 that the central finite differences approximation error decreases un-

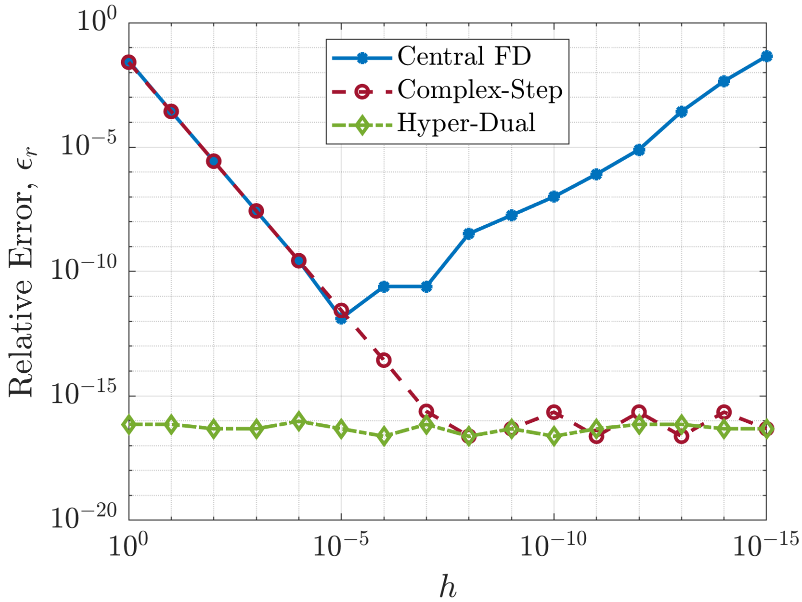


Figure 5.7: First-order derivative approximation.

til $h \approx 10^{-5}$ and then begins to increase again due to roundoff error. The bi-complex step derivative approximation error decreases as h decreases until it reaches machine precision and $\epsilon_r \approx 10^{-16}$. The hyper-dual derivative approximation remains at a constant machine precision error regardless of step size. Note that TOPS is currently capable of using bi-complex step differentiation for first derivative calculations by setting `p.opts.derivative_type = 'CS'`. In the future, hyper-dual differentiation will be implemented.

5.3.3 Automatic/Algorithmic Differentiation

The final method of obtaining derivative that will be briefly discussed is automatic or algorithmic differentiation (AD). AD is an open field of research. The general idea behind AD is to decompose a function into its elementary operations and then apply the calculus chain rule to those operations in order to evaluate the derivative [112]. Since calculus rules are being applied, the derivatives calculated using AD are *exact* to machine precision. There are two ways to implement AD:

1) Operator-overloading (OO) . This computational technique applies the principles of overloaded functions and dual numbers in order to evaluate a function and its derivative simultaneously. Typically, OO techniques are implemented using a custom pseudo-dual-number class that allows simultaneous storage of the current function value and its derivative. This class also overloads basic mathematical functions with complex or hyper-dual arithmetic rules. Simply evaluating the function with an overloaded class variable allows for the function and its derivative to be calculated.

2) Source-to-source (S2S) . This type of AD examines the function source code and uses it to produce an entirely new function that calculates the original function value and the derivative. Typically, S2S is slightly faster than OO since dual number arithmetic is not being used and the derivative calculations are hard coded.

There are also two *modes* of AD: forward- and reverse-mode AD. The details of these two types of AD are beyond the scope of this study. If the user wishes to learn more about the theory and differences of each, see Griewank and Walther [72] and Griewank [71]. For practical implementation, see Neidinger [119]. Suffice it to say that forward-mode AD is excellent at calculating gradients while reverse-mode AD is more appropriate for calculating the derivative adjoint, which is used frequently in reinforcement learning. TOPS uses the open-source source-to-source forward-mode AD software *ADiGator* [190] to supply first derivatives and second derivatives of the NLP functions. The primary reason this software is used is due to its automatic exploitation of sparsity. Derivatives that are zero are not calculated, thus saving significant amounts of computational effort. This will be discussed further in Section 5.4.

Up until this point, we have discussed several methods of calculating the derivatives of an NLP function. But, we have not yet fully defined these NLP derivatives. The NLP first derivatives are shown below.

$$\nabla_{\boldsymbol{z}} J(\boldsymbol{z}), \qquad \nabla_{\boldsymbol{z}} \boldsymbol{c}(\boldsymbol{z}), \qquad \nabla_{\boldsymbol{z}} \boldsymbol{c}_{\text{eq}}(\boldsymbol{z}). \tag{5.59}$$

In Eq. (5.59), $\boldsymbol{z}$ is the NLP decision vector given in Eq. (5.2), $J(\boldsymbol{z})$ is the cost function, and $\boldsymbol{c}(\boldsymbol{z})$ and $\boldsymbol{c}_{\text{eq}}(\boldsymbol{z})$ are the inequality and equality constraints, respectively. They are both column vectors. The only second derivative required is the Hessian of the Lagrangian, given by,

$$\nabla_{\boldsymbol{zz}} \mathcal{L} = \nabla_{\boldsymbol{zz}} [J(\boldsymbol{z}) + \boldsymbol{\Gamma}^\top \boldsymbol{c}(\boldsymbol{z}) + \boldsymbol{\Lambda}^\top \boldsymbol{c}_{\text{eq}}(\boldsymbol{z})]. \tag{5.60}$$

Here, $\boldsymbol{\Gamma}$ and $\boldsymbol{\Lambda}$ are the KKT multipliers associated with the inequality and equality constraints, respectively. They are the same size as $\boldsymbol{c}$ and $\boldsymbol{c}_{\text{eq}}$ such that each element of the KKT vectors corresponds to a value in the constraint vectors. The naive (and usually inefficient) way to generate these derivatives is to simply construct functions that calculate J, $\boldsymbol{c}$, and $\boldsymbol{c}_{\text{eq}}$ and then use finite differences or dual-number differentiation to calculate the derivative with respect to each element of $\boldsymbol{z}$. However, we can avoid this by taking advantage of the structure of a PS method to *only calculate non-zero derivatives*. Consider the example Jacobian sparsity pattern given in Fig. 5.8. Fig. 5.8 considers an arbitrary free final time example problem with four states $\boldsymbol{X}_{1:4}$, one control $\boldsymbol{U}_1$, path constraints $\boldsymbol{h}$, and event constraints $\boldsymbol{e}$. It is evident that there is a clear sparsity pattern and that only calculating the non-zero elements is necessary. If a method of automatically locating and calculating these non-zero derivatives can be formulated, efficiency can be drastically increased. This is because most of the time spent solving an NLP problem is spent calculating derivatives [129, 3].

Consider the matrix form of the Radau PS method given in Problem $(\text{B}^{\text{LGR}}_{\text{MAT}})$. In this section, we will only give exact formulas for *first* derivatives. However, for second derivative expressions, the reader is encouraged to read Patterson and Rao [129], which contains the first derivative expressions given in this section as well as second derivative expressions. First, we

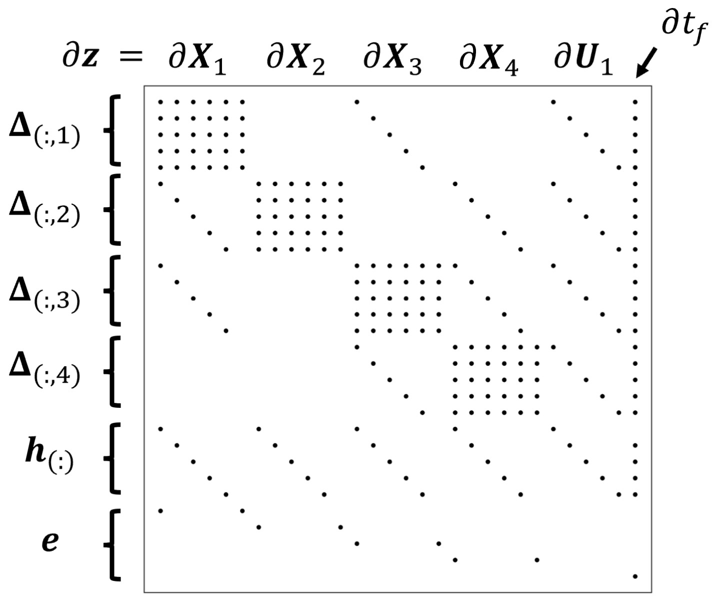


Figure 5.8: Example Jacobian Sparsity Pattern

must form a global vector quantity in addition to those defined in the previous sections. First, let,

$$\boldsymbol{F} \in \mathbb{R}^{N_t \times 1} = F(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}), \tag{5.61}$$

$$\boldsymbol{f} \in \mathbb{R}^{N_t \times N_x} = \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}). \tag{5.62}$$

Here, we evaluate the scalar overloaded operator $F \in \mathbb{R}$ at the global decision variable matrices, $\boldsymbol{X}$, $\boldsymbol{U}$, and $\boldsymbol{t}$. The result is a column vector. Similarly, evaluation of $\boldsymbol{f}$ at the global decision variables produces a matrix. When we need to highlight this operation for an arbitrary function or variable $\boldsymbol{p}$, we will use the notation,

$$[\boldsymbol{p}]_{s=1}^{N_t} = \begin{bmatrix} \boldsymbol{p}(\boldsymbol{X}(t_1), \boldsymbol{U}(t_1), t_1) \\ \boldsymbol{p}(\boldsymbol{X}(t_2), \boldsymbol{U}(t_2), t_2) \\ \vdots \\ \boldsymbol{p}(\boldsymbol{X}(t_{N_t}), \boldsymbol{U}(t_{N_t}), t_{N_t}) \end{bmatrix}. \tag{5.63}$$

Here, $[\boldsymbol{p}]_{s=1}^{N_t}$ is a shorthand expression for evaluating the argument in the brackets at the NLP variables associated with time t_s, where $(s = 1, \dots, N_t)$. The result is a column vector if

p is a scalar, and a matrix if $\boldsymbol{p}$ is a vector. We then let,

$$\boldsymbol{\Delta} \in \mathbb{R}^{N_t \times N_x} = \boldsymbol{D} \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{x}_f \end{bmatrix} - \frac{t_f - t_0}{2} \boldsymbol{\sigma} \boldsymbol{f}, \tag{5.64}$$

be the *defect* constraints for the dynamics, $\boldsymbol{f}$. Next, let,

$$\boldsymbol{h} \in \mathbb{R}^{N \times N_h} = \boldsymbol{h}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}), \tag{5.65}$$

$$\boldsymbol{e} \in \mathbb{R}^{N_e \times 1} = \boldsymbol{e}(\boldsymbol{x}_1^{(1)}, \boldsymbol{x}_f, t_0, t_f), \tag{5.66}$$

be the matrix and vector associated with the path inequality and boundary equality constraints when evaluated at the global decision vector and endpoints, respectively. For the sake of notational expedience, we combine all constraints into a single vector, $\boldsymbol{C}$. We also break up the cost function into two terms. Thus, the decision vector, constraint vector, and cost are given as,

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{X}_{(:)} \\ \boldsymbol{U}_{(:)} \\ t_0 \\ t_f \end{bmatrix}, \qquad \boldsymbol{C} = \begin{bmatrix} \boldsymbol{\Delta}_{(:)} \\ \boldsymbol{h}_{(:)} \\ \boldsymbol{e} \end{bmatrix}, \qquad J(\boldsymbol{z}) = \omega(\boldsymbol{z}) + \theta(\boldsymbol{z}), \tag{5.67}$$

where in Eq. (5.67), we have

$$\omega(\boldsymbol{z}) = E(\boldsymbol{x}_1^{(1)}, \boldsymbol{x}_f, t_0, t_f), \qquad \theta = \frac{t_f - t_0}{2} \boldsymbol{w}^\top \boldsymbol{\sigma} \boldsymbol{F}. \tag{5.68}$$

The reader is reminded that the notation $\boldsymbol{X}_{(:)}$ is defined in Section 1.1 and is encouraged to review the notational conventions presented in that section before continuing. Now, we may consider the first derivative of the objective function.

5.4.1 Gradient of the Objective

Recall that the NLP derivatives are obtained by differentiating each element with respect to the optimization vector, $\boldsymbol{z}$. Thus, from Eq. (5.67), our objective gradient is given by,

$$\nabla_{\boldsymbol{z}} J = \nabla_{\boldsymbol{z}} \omega + \nabla_{\boldsymbol{z}} \theta. \tag{5.69}$$

We can split the gradient $\nabla_{\boldsymbol{z}}$ into its unrolled components,

$$\nabla_{\boldsymbol{z}} = [\nabla_{\boldsymbol{X}_{(:)}}, \nabla_{\boldsymbol{U}_{(:)}}, \nabla_{t_0}, \nabla_{t_f}]. \tag{5.70}$$

Recall also that even though $\boldsymbol{X}_{(:)}$ and $\boldsymbol{U}_{(:)}$ are column vectors, the result of the gradient with respect to a column vector is still a row vector. Thus, the derivatives of the endpoint cost with respect to $\boldsymbol{z}$ is given as,

$$\nabla_{\boldsymbol{z}} \omega = [\nabla_{\boldsymbol{X}_{(:)}} \omega, \nabla_{\boldsymbol{U}_{(:)}} \omega, \nabla_{t_0} \omega, \nabla_{t_f} \omega]. \tag{5.71}$$

The elements of this derivative are given below as

$$\nabla_{\boldsymbol{X}_{(:)}} \omega = [\nabla_{\boldsymbol{X}_{(:,1)}} \omega,\ \nabla_{\boldsymbol{X}_{(:,2)}} \omega, \dots,\ \nabla_{\boldsymbol{X}_{(:,N_x)}} \omega], \tag{5.72}$$

$$\nabla_{\boldsymbol{U}_{(:)}} \omega = [\mathbf{0}_{1 \times N_t N_u}], \tag{5.73}$$

$$\nabla_{t_0} \omega = \frac{\partial E}{\partial t_0}, \tag{5.74}$$

$$\nabla_{t_f} \omega = \frac{\partial E}{\partial t_f}. \tag{5.75}$$

Here, the derivative with respect to $\boldsymbol{U}_{(:)}$ is always zero, since the endpoint cost is not a function of $\boldsymbol{U}$ for the LGR transcription. The elements of Eq. (5.72) are given by,

$$\nabla_{\boldsymbol{X}_{(:,i)}} \omega = \left[\frac{\partial E}{\partial x_i(t_0)},\ \mathbf{0}_{1 \times (N_t - 1)},\ \frac{\partial E}{\partial x_i(t_f)} \right], \quad (i = 1, \dots, N_x). \tag{5.76}$$

This completes the derivative elements for the objective gradient. The advantage of this form is that it is *general*, and it requires far fewer derivative evaluations than simply iterating over the elements of $\boldsymbol{z}$ to get the derivative of ω with respect to each. While iteration over $\boldsymbol{z}$ would take $N_t(N_x+N_u)+N_x+2$ function evaluations, taking advantage of this sparse structure requires only $2N_x + 2$ function evaluations. Next, we generate derivatives of the running cost given by Eq. (5.68) with respect to $\boldsymbol{z}$. This is given by,

$$\nabla_{\boldsymbol{z}}\theta = [\nabla_{\boldsymbol{X}_{(:)}}\theta, \nabla_{\boldsymbol{U}_{(:)}}\theta, \nabla_{t_0}\theta, \nabla_{t_f}\theta], \tag{5.77}$$

where,

$$\nabla_{\boldsymbol{X}_{(:)}}\theta = [\nabla_{\boldsymbol{X}_{(:,1)}}\theta,\ \nabla_{\boldsymbol{X}_{(:,2)}}\theta, \ldots,\ \nabla_{\boldsymbol{X}_{(:,N_x)}}\theta], \tag{5.78}$$

$$\nabla_{\boldsymbol{U}_{(:)}}\theta = [\nabla_{\boldsymbol{U}_{(:,1)}}\theta,\ \nabla_{\boldsymbol{U}_{(:,2)}}\theta, \ldots,\ \nabla_{\boldsymbol{U}_{(:,N_u)}}\theta]. \tag{5.79}$$

Since the running cost depends on control, derivatives with respect to $\boldsymbol{U}$ no longer disappear. Although it may initially seem like the running cost may not depend on t_f for the LGR PS method, this is not the case. Recall that the overloaded vector functions $\boldsymbol{f}$ and $\boldsymbol{F}$ in Problem $(\mathrm{B}^{\mathrm{LGR}}_{\mathrm{MAT}})$ are functions of $\boldsymbol{t}$ and that $\boldsymbol{t}$ is obtained from $\boldsymbol{s}$ using,

$$\boldsymbol{t} = \frac{t_f - t_0}{2}\boldsymbol{s} + \frac{t_f + t_0}{2}. \tag{5.80}$$

From Eq. (5.80), the implicit dependencies on t_0 and t_f become clear. To obtain the derivative of these vectorized functions, we may use the chain rule, such that

$$\frac{\partial \boldsymbol{x}}{\partial t_0} = \frac{\partial \boldsymbol{x}}{\partial t}\frac{\partial t}{\partial t_0}, \tag{5.81}$$

$$\frac{\partial \boldsymbol{x}}{\partial t_f} = \frac{\partial \boldsymbol{x}}{\partial t}\frac{\partial t}{\partial t_f}. \tag{5.82}$$

Since $\boldsymbol{t}$ is a column vector, we obtain these partial derivatives from Eq. (5.80) as,

$$\boldsymbol{\alpha} := \frac{\partial \boldsymbol{t}}{\partial t_0} = \text{diag}\frac{1-\boldsymbol{s}}{2}, \tag{5.83}$$

$$\boldsymbol{\beta} := \frac{\partial \boldsymbol{t}}{\partial t_f} = \text{diag}\frac{1+\boldsymbol{s}}{2}. \tag{5.84}$$

Using these expressions, we obtain the derivative elements $\nabla_{\boldsymbol{X}_{(:,i)}}\theta$, $\nabla_{\boldsymbol{U}_{(:,i)}}\theta$, $\nabla_{t_0}\theta$, and $\nabla_{t_f}\theta$ as,

$$\nabla_{\boldsymbol{X}_{(:,i)}}\theta = \left[\frac{t_f - t_0}{2}\left\{\text{diag}(\boldsymbol{w})\boldsymbol{\sigma}\left[\frac{\partial F}{\partial x_i}\right]_{s=1}^{N_t}\right\}^\top, \ 0\right], \quad (i = 1, \ldots, N_x), \tag{5.85}$$

$$\nabla_{\boldsymbol{U}_{(:,j)}}\theta = \frac{t_f - t_0}{2}\left\{\text{diag}(\boldsymbol{w})\boldsymbol{\sigma}\left[\frac{\partial F}{\partial u_j}\right]_{s=1}^{N_t}\right\}^\top, \qquad (j = 1, \ldots, N_u), \tag{5.86}$$

$$\nabla_{t_0}\theta = -\frac{1}{2}\boldsymbol{w}^\top\boldsymbol{\sigma}\boldsymbol{F} + \frac{t_f - t_0}{2}\boldsymbol{w}^\top\left\{\boldsymbol{\alpha}\boldsymbol{\sigma}\left[\frac{\partial F}{\partial t}\right]_{s=1}^{N_t}\right\}, \tag{5.87}$$

$$\nabla_{t_f}\theta = \frac{1}{2}\boldsymbol{w}^\top\boldsymbol{\sigma}\boldsymbol{F} + \frac{t_f - t_0}{2}\boldsymbol{w}^\top\left\{\boldsymbol{\beta}\boldsymbol{\sigma}\left[\frac{\partial F}{\partial t}\right]_{s=1}^{N_t}\right\}. \tag{5.88}$$

Note that scalar 0 as the last element of Eq. (5.85) is the derivative with respect to the non-collocated point $\boldsymbol{x}_f$. Although these highly-vectorized equations may look imposing, they will become much easier to understand should the reader examine the `compute-all` functions used by TOPS. Using all of these equations, one may compute the derivatives of ω and θ with respect to the sub-elements of $\boldsymbol{z}$. The objective gradient is then computed according to Eq. (5.69).

5.4.2 Jacobian of the Constraints

Next, we consider the derivative of the constraints with respect to the decision vector. This is defined as,

$$\nabla_{\boldsymbol{z}}\boldsymbol{C} = \begin{bmatrix} \nabla_{\boldsymbol{z}}\boldsymbol{\Delta}_{(:)} \\ \nabla_{\boldsymbol{z}}\boldsymbol{h}_{(:)} \\ \nabla_{\boldsymbol{z}}\boldsymbol{e} \end{bmatrix}. \tag{5.89}$$

The first derivatives of the defect constraints, can be computed according to,

$$\nabla_{\boldsymbol{z}} \boldsymbol{\Delta}_{(:,k)} = \left[\nabla_{\boldsymbol{X}_{(:)}} \boldsymbol{\Delta}_{(:,k)},\ \nabla_{\boldsymbol{U}_{(:)}} \boldsymbol{\Delta}_{(:,k)}, \nabla_{t_0} \boldsymbol{\Delta}_{(:,k)}, \nabla_{t_f} \boldsymbol{\Delta}_{(:,k)} \right], \tag{5.90}$$

for $(k = 1, \ldots, N_x)$. The $\boldsymbol{X}_{(:)}$ and $\boldsymbol{U}_{(:)}$ elements of this derivative can be broken down as,

$$\nabla_{\boldsymbol{X}_{(:)}} \boldsymbol{\Delta}_{(:,k)} = \left[\nabla_{\boldsymbol{X}_{(:,1)}} \boldsymbol{\Delta}_{(:,k)},\ \nabla_{\boldsymbol{X}_{(:,2)}} \boldsymbol{\Delta}_{(:,k)}, \ldots,\ \nabla_{\boldsymbol{X}_{(:,N_x)}} \boldsymbol{\Delta}_{(:,k)} \right], \tag{5.91}$$

$$\nabla_{\boldsymbol{U}_{(:)}} \boldsymbol{\Delta}_{(:,k)} = \left[\nabla_{\boldsymbol{U}_{(:,1)}} \boldsymbol{\Delta}_{(:,k)},\ \nabla_{\boldsymbol{U}_{(:,2)}} \boldsymbol{\Delta}_{(:,k)}, \ldots,\ \nabla_{\boldsymbol{U}_{(:,N_u)}} \boldsymbol{\Delta}_{(:,k)} \right], \tag{5.92}$$

where $(k = 1, \ldots, N_x)$. The reader is again reminded that the derivative element $\nabla_{\boldsymbol{X}_{(:,i)}} \boldsymbol{\Delta}_{(:,k)}$ is a *row vector*, even though the argument $\boldsymbol{\Delta}_{(:,k)}$ is a column vector. The first derivatives of the defect constraints given by $\nabla_{\boldsymbol{X}_{(:,i)}} \boldsymbol{\Delta}_{(:,k)}$ and $\nabla_{\boldsymbol{U}_{(:,j)}} \boldsymbol{\Delta}_{(:,k)}$ for $(i = 1, \ldots, N_x)$ and $(j = 1, \ldots, N_u)$ can be obtained as,

$$\nabla_{\boldsymbol{X}_{(:,i)}} \boldsymbol{\Delta}_{(:,k)} = \left[\delta_{ik} \boldsymbol{D}_{(:,1:N_t)} - \frac{t_f - t_0}{2} \mathrm{diag} \left(\boldsymbol{\sigma} \left[\frac{\partial f_k}{\partial x_i} \right]_{s=1}^{N_t} \right),\ \ \delta_{ik} \boldsymbol{D}_{(:,N_t+1)} \right], \tag{5.93}$$

$$\nabla_{\boldsymbol{U}_{(:,j)}} \boldsymbol{\Delta}_{(:,k)} = -\frac{t_f - t_0}{2} \mathrm{diag} \left(\boldsymbol{\sigma} \left[\frac{\partial f_i}{\partial u_j} \right]_{s=1}^{N_t} \right), \tag{5.94}$$

for $(i, k = 1, \ldots, N_x)$ and $(j = 1, \ldots, N_u)$. In Eq. (5.93), δ_{ik} is the Kronecker delta function, given by,

$$\delta_{ik} = \begin{cases} 1, & i = k, \\ 0, & \text{otherwise.} \end{cases} \tag{5.95}$$

Note that although Eq. (5.93) is concise, it is difficult to express it clearly in a single mathematical expression. The reader is encouraged to explore the vectorized optimization examples of the *ADiGator* toolbox [190] for several examples of how this equation can be implemented computationally. A copy of *ADiGator* can be downloaded for free from Dr. Matthew Weinstein's Github page. The derivatives with respect to time, given by $\nabla_{t_0} \boldsymbol{\Delta}_{(:,k)}$ and $\nabla_{t_0} \boldsymbol{\Delta}_{(:,k)}$ are obtained as,

$$\nabla_{t_0}\boldsymbol{\Delta}_{(:,k)} = \frac{1}{2}[f_k]_{s=1}^{N_t} - \frac{t_f - t_0}{2}\boldsymbol{\alpha\sigma}\left[\frac{\partial f_k}{\partial t}\right]_{s=1}^{N_t}, \tag{5.96}$$

$$\nabla_{t_f}\boldsymbol{\Delta}_{(:,k)} = -\frac{1}{2}[f_k]_{s=1}^{N_t} - \frac{t_f - t_0}{2}\boldsymbol{\beta\sigma}\left[\frac{\partial f_k}{\partial t}\right]_{s=1}^{N_t}, \tag{5.97}$$

for $(k = 1, \ldots, N_x)$. Thus, we have obtained the derivatives of the defect constraints with respect to the optimization vector. The first derivative of the path constraints is given by,

$$\nabla_{\boldsymbol{z}}\boldsymbol{h}_{(:,p)} = \left[\nabla_{\boldsymbol{X}_{(:)}}\boldsymbol{h}_{(:,p)},\ \nabla_{\boldsymbol{U}_{(:)}}\boldsymbol{h}_{(:,p)}, \nabla_{t_0}\boldsymbol{h}_{(:,p)}, \nabla_{t_f}\boldsymbol{h}_{(:,p)}\right], \tag{5.98}$$

where $(p = 1, \ldots, N_h)$. The elements associated with the $\boldsymbol{X}$ and $\boldsymbol{U}$ components can be further broken down as,

$$\nabla_{\boldsymbol{X}_{(:)}}\boldsymbol{h}_{(:,p)} = \left[\nabla_{\boldsymbol{X}_{(:,1)}}\boldsymbol{h}_{(:,p)},\ \nabla_{\boldsymbol{X}_{(:,2)}}\boldsymbol{h}_{(:,p)}, \ldots, \nabla_{\boldsymbol{X}_{(:,N_x)}}\boldsymbol{h}_{(:,p)}\right], \tag{5.99}$$

$$\nabla_{\boldsymbol{U}_{(:)}}\boldsymbol{h}_{(:,p)} = \left[\nabla_{\boldsymbol{U}_{(:,1)}}\boldsymbol{h}_{(:,p)},\ \nabla_{\boldsymbol{U}_{(:,2)}}\boldsymbol{h}_{(:,p)}, \ldots, \nabla_{\boldsymbol{U}_{(:,N_u)}}\boldsymbol{h}_{(:,p)}\right]. \tag{5.100}$$

The elements $\nabla_{\boldsymbol{X}_{(:,i)}}\boldsymbol{h}_{(:,p)}$ and $\nabla_{\boldsymbol{U}_{(:,j)}}\boldsymbol{h}_{(:,p)}$ can be calculated as,

$$\nabla_{\boldsymbol{X}_{(:,i)}}\boldsymbol{h}_{(:,p)} = \left[\operatorname{diag}\left[\frac{\partial h_p}{\partial x_i}\right]_{s=1}^{N_t},\ \boldsymbol{0}_{N_t\times 1}\right], \tag{5.101}$$

$$\nabla_{\boldsymbol{U}_{(:,j)}}\boldsymbol{h}_{(:,p)} = \operatorname{diag}\left[\frac{\partial h_p}{\partial u_j}\right]_{s=1}^{N_t}, \tag{5.102}$$

where $(i = 1, \ldots, N_x)$, $(j = 1, \ldots, N_u)$, and $(p = 1, \ldots, N_h)$. The derivatives associated with the initial and final times, $\nabla_{t_0}\boldsymbol{h}_{(:,p)}$ and $\nabla_{t_f}\boldsymbol{h}_{(:,p)}$ can be calculated as,

$$\nabla_{t_0}\boldsymbol{h}_{(:,p)} = \boldsymbol{\alpha}\left[\frac{\partial h_p}{\partial t}\right]_{s=1}^{N_t}, \tag{5.103}$$

$$\nabla_{t_0}\boldsymbol{h}_{(:,p)} = \boldsymbol{\beta}\left[\frac{\partial h_p}{\partial t}\right]_{s=1}^{N_t}. \tag{5.104}$$

Using the sparse derivative forms, one may calculate the derivatives of the path constraints with respect to $\boldsymbol{X}$, $\boldsymbol{U}$, and $\boldsymbol{t}$ and simply evaluate these derivative expressions at the

N_t collocation points. We still have to calculate the derivative of the boundary conditions, $\boldsymbol{e}(\boldsymbol{x}_1^{(1)}, \boldsymbol{x}_f, t_0, t_f)$. The first derivatives are given as,

$$\nabla_{\boldsymbol{z}} e_q = \left[\nabla_{\boldsymbol{X}_{(:)}} e_q, \nabla_{\boldsymbol{U}_{(:)}} e_q, \nabla_{t_0} e_q, \nabla_{t_f} e_q\right], \tag{5.105}$$

for $(q = 1, \ldots, n_e)$. The derivative elements with respect to $\boldsymbol{X}$ and $\boldsymbol{U}$ are given by,

$$\nabla_{\boldsymbol{X}_{(:)}} = \left[\nabla_{\boldsymbol{X}_{(:,1)}} e_q,\ \nabla_{\boldsymbol{X}_{(:,2)}} e_q,\ \ldots,\ \nabla_{\boldsymbol{X}_{(:,N_x)}} e_q\right], \tag{5.106}$$

$$\nabla_{\boldsymbol{U}_{(:)}} = \left[\boldsymbol{0}_{1\times N_t N_u}\right]. \tag{5.107}$$

Since we are using the LGR transcription, the control is not obtained at the final value. However, even for other transcriptions, the boundary constraints are not usually expressed as a function of final control values, so the derivative with respect to $\boldsymbol{U}$ remains zero regardless of transcription. The derivative elements $\nabla_{\boldsymbol{X}_{(:,i)}} e_q$ can be obtained in a sparse manner as,

$$\nabla_{\boldsymbol{X}_{(:,i)}} e_q = \left[\frac{\partial e_q}{\partial x_i(t_0)},\ \boldsymbol{0}_{1\times(N_t-1)},\ \frac{\partial e_q}{\partial x_i(t_f)}\right], \tag{5.108}$$

where $(q = 1, \ldots, N_e)$. The derivative elements $\nabla_{t_0} e_q$ and $\nabla_{t_f} e_q$ are given by,

$$\nabla_{t_0} e_q = \frac{\partial e_q}{\partial t_0}, \tag{5.109}$$

$$\nabla_{t_f} e_q = \frac{\partial e_q}{\partial t_f}. \tag{5.110}$$

This concludes the exact expressions for the first derivatives. For the sake of expediency, the second derivative Lagrangian Hessian expressions are not included, as they are quite lengthy. However, the reader is again encouraged to read Patterson and Rao [129] if they desire to understand the computation of second derivatives in a sparse manner.

The author cannot express how invaluable *ADiGator* [190] is to the AD functionality of TOPS, as well as for increasing the efficiency of any NLP solver by providing exact derivatives.

Throughout this section, there are terms that take the form,

$$\nabla_{\boldsymbol{x}} f = \left[\frac{\partial f}{\partial x}\right]_{s=1}^{N_t}, \tag{5.111}$$

where $\boldsymbol{x}$ is some row/column vector that has a value at every point t_s. This expression is a partial derivative that must be evaluated N_t times and then inserted into various expressions to calculate the derivatives. When using finite differences or dual-number differentiation, this will require a loop that iterates over the values of $\boldsymbol{x}$ to get the derivative of f with respect to x_i at every point in $\boldsymbol{x}$, where $(i = 1, \ldots, N_x)$. Vectorization of this process is usually very difficult. However, *ADiGator* has a "vectorized" mode of operation that can generate a single derivative expression for an arbitrary number of points, N_t. Thus, a derivative file need only be generated *once* and then the derivatives can be evaluated in a vectorized form for an arbitrary N_t. This can drastically improve the performance of an NLP solver being used in an interpreted language like MATLAB, as loops are entirely eliminated from the process of obtaining derivatives. The reader is once again encouraged to download a copy of *ADiGator*, read the manual, and explore the examples to understand this process. TOPS also contains several routines for calculating these derivatives using *ADiGator*, although they are not commented as thoroughly.

5.5 Nested Implementation of the Objective and Constraint Functions

Another very useful computational "trick" for improving the efficiency of many NLP solvers is a process known as NLP nesting. To explain this process, it is necessary to understand that most NLP solvers call the objective and constraint functions separately. This can be wasteful if the objective and constraint function share variables or perform the same calculation. In addition to this, they can sometimes evaluate the objective and constraints at the same point multiple times in a row. Thus, it can improve computational efficiency to compute the objective and constraints in a single function and evaluate it only when the point of evaluation changes. This can be accomplished as follows.

1. Write a "compute-all" function that computes both the objective and constraints in the same variable scope.

2. Write a function that wraps the NLP solver and can store recent values output by the compute-all function.

3. Write a nested objective and constraint function within the wrapper function that are passed as the objective and constraint functions to the NLP solver.

4. Write an if-else statement within these nested functions that checks if the input value from the NLP solver is the same as the previous input value, which is stored outside these functions.

5. If the value passed by the NLP solver is not the same, call the compute-all function to compute the objective and constraints.

6. If the value passed by the NLP solver is the same, do not call the compute-all function and simply return the previously stored values.

In order to illustrate this, a pseudocode function is provided as an example.

```
% ================================
% Example Nested NLP Function
% ================================
function [z_sol, f_sol, exitflag] = solve_NLP(z0,options)
% First, we initialize variables for the storage of re-used values.
z_last  = []; % The last value passed by the NLP solver.
obj_all = []; % We store the objective value here.
c_all   = []; % We store the inequalities here.
ceq_all = []; % We store the equalities here.

% This is where we call our NLP solver. We use fmincon for this
% example.
[z_sol, f_sol, exitflag] = ...
    fmincon(obj_fun,z0,[],[],[],[],[],[],constr_fun,options);

% Next, we define our nested objective and constraint functions.
```

```
function obj = obj_fun(z)
    % Here, we check if the z being passed is equal to the last
    % value of z that was passed.
    if z ≠ z_last
        % If z isn't the same as the previous z value, we call
        % compute_all.
        [obj_all,c_all,ceq_all] = compute_all(z);
        z_last = z;
    end

    % Set the output.
    obj = obj_all;
end

function [c,ceq] = constr_fun(z)
    % We repeat the process performed in obj_fun.
    if z ≠ z_last
        % If z isn't the same as the previous z value, we call
        % compute_all.
        [obj_all,c_all,ceq_all] = compute_all(z);
    end

    % Set the output.
    c   = c_all;
    ceq = ceq_all;
end
end
```

This concludes the example code. Although this is a pseudocode, the user may replicate this code exactly and simply write their own "compute-all" function. In the experience of the author, this can halve the time required to obtain a solution to the PS NLP. Since it is an exceedingly easy and simple change to implement, the reader is encouraged to do so when solving any NLP problem. This process is extendable to first and second derivatives. These share many calculations during their computations, which can lead to further increases in efficiency.

5.6 Automatic Scaling

A somewhat controversial aspect of solving NLPs is the process of automatic scaling. This is a process by which the NLP variables are scaled using an affine transformation such that the lower and upper bounds of the variables are 0 and 1, respectively [156]. Some scaling methods scale the variables to the range $[-1, 1]$ [141], while some scale them to the range $[-0.5, 0.5]$ [130]. These scaling ranges are arbitrary and primarily a preference of the user. In theory, NLP scaling should increase the efficiency and accuracy of the NLP solver, since all design variables are of the same magnitude and near machine zero. This is where floating-point arithmetic is most accurate, since there are more decimal places available for computations and roundoff error is reduced. However, Ross et al. [151] shows that oftentimes this procedure of scaling all NLP variables to a range close to zero can have an unintended effect on the KKT multipliers and costates. Although the NLP variables become well-scaled, the KKT multipliers become badly scaled. The KKT multipliers for the automatically scaled problem can become extremely large depending on the problem and drive the NLP solver to a poor solution.

In the experience of the author, this proposition is true for many practical problems. The simple act of automatic scaling, while improving the computational speed of the solver, can destabilize the solution and cause convergence to a pseudo-minimizer. Ross et al. [151] propose a heuristic process by which a set of designer scaling units is chosen through trial-and-error to "balance" the magnitude of the scaled states and costates in order to maximize the performance of the NLP solver. However, this is still a "guess-and-check" method that requires significant user input, so TOPS provides an automatic scaling technique that the user may apply to their problem. If the scaling technique does not produce satisfactory results, the user may fall back on the scaling and balancing technique or rely on traditional or canonical scaling techniques (which depends on the problem) for the state parameters.

5.6.1 Affine Scaling

The scaling technique used by TOPS is presented here. Consider an arbitrary variable $x \in [l, u]$, where l is a lower bound for x and u is an upper bound for x. If the user desires to scale the variables to the range $[0, 1]$, the following procedure can be used.

$$l \leq x \leq u,$$

$$0 \leq x - l \leq u - l,$$

$$0 \leq \frac{1}{u-l}x - \frac{l}{u-l} \leq 1,$$

Here, we take,

$$\tilde{x} = K_x x + b, \tag{5.112}$$

where,

$$K_x = \frac{1}{u-l}, \quad b = -\frac{l}{u-l}. \tag{5.113}$$

We may extend this to the NLP optimization vector, $\boldsymbol{z} \in [\boldsymbol{z}_l, \boldsymbol{z}_u]$, where $\boldsymbol{z}_l$ and $\boldsymbol{z}_u$ correspond element-wise to the lower and upper bounds of $\boldsymbol{z}$. The equivalent affine scaling formula is [156, 110],

$$\tilde{\boldsymbol{z}} = \boldsymbol{K}_{\boldsymbol{z}}\boldsymbol{z} + \boldsymbol{b}, \tag{5.114}$$

where,

$$\boldsymbol{K}_{\boldsymbol{z}} = \text{diag}\left(\frac{1}{\boldsymbol{z}_u - \boldsymbol{z}_l}\right), \quad \boldsymbol{b} = -\frac{\boldsymbol{z}_l}{\boldsymbol{z}_u - \boldsymbol{z}_l}. \tag{5.115}$$

Here, $\boldsymbol{K}_{\boldsymbol{z}} \in \mathbb{R}^{N_z \times N_z}$ and $\boldsymbol{b} \in \mathbb{R}^{N_z \times 1}$. In practice, the NLP states are scaled to $\widetilde{\boldsymbol{z}}$ prior to the optimization process and are unscaled to $\boldsymbol{z}$ in the compute-all function to evaluate the user-functions. Thus, the inverse transformation of Eq. (5.115) is given by,

$$\boldsymbol{z} = \boldsymbol{K}_{\boldsymbol{z}}^{-1}(\tilde{\boldsymbol{z}} - \boldsymbol{b}). \tag{5.116}$$

This process of scaling and unscaling requires that the output of certain user functions be *re*-scaled. For an *isoscaling* method, this re-scaling process is only applied to derivatives, which include the defect constraints and the NLP gradient and Jacobians. The derivative of the scaled state given in Eq. (5.114) with respect to any variable is,

$$\dot{\tilde{\boldsymbol{z}}} = \boldsymbol{K}_{\boldsymbol{z}}\dot{\boldsymbol{z}}. \tag{5.117}$$

After applying this to the dynamics function, we obtain,

$$\widetilde{\boldsymbol{f}}_{(:)} = \boldsymbol{K_x} \boldsymbol{f}_{(:)}, \tag{5.118}$$

where $\boldsymbol{K_x}$ is the block-diagonal portion of $\boldsymbol{K_z}$ that only includes state-scaling information. It is obtained by taking rows and columns 1 through $(N_t + 1)N_x$ from $\boldsymbol{K_z}$. Applying this to the defect constraints, we obtain,

$$\widetilde{\boldsymbol{\Delta}} = \boldsymbol{K_x} \boldsymbol{\Delta}. \tag{5.119}$$

Note that when scaling using this method, the derivatives are no longer with respect to $\boldsymbol{z}$, but instead with respect to $\widetilde{\boldsymbol{z}}$. Thus, in order to find the derivatives with respect to $\widetilde{\boldsymbol{z}}$, we use chain rule, such that,

$$\frac{\partial \boldsymbol{C}}{\partial \tilde{\boldsymbol{z}}} = \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \tilde{\boldsymbol{z}}}. \tag{5.120}$$

We obtain this chain rule derivative term as,

$$\frac{\partial \boldsymbol{z}}{\partial \tilde{\boldsymbol{z}}} = \boldsymbol{K_z}^{-1}. \tag{5.121}$$

Thus, the constraint Jacobian becomes,

$$\widetilde{\nabla}_{\boldsymbol{z}} \boldsymbol{C} = \nabla_{\boldsymbol{z}} \boldsymbol{C} \cdot \boldsymbol{K_z}^{-1}, \tag{5.122}$$

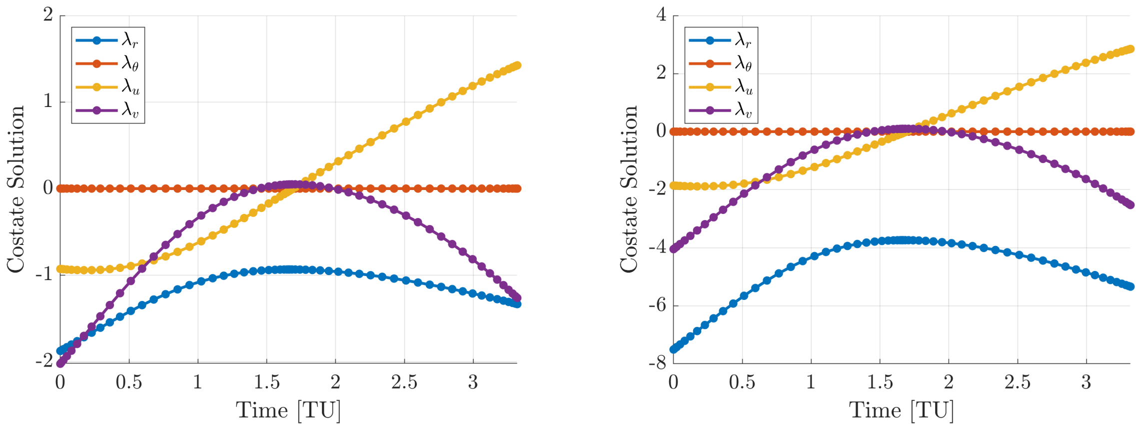
where $\widetilde{\nabla}_{\boldsymbol{z}}$ denotes the derivative with respect to $\tilde{\boldsymbol{z}}$. The back-scaled costates are given as,

$$\boldsymbol{\lambda} = \boldsymbol{K_z} \tilde{\boldsymbol{\lambda}}. \tag{5.123}$$

In this isoscaling method, the objective and any additional inequality or equality constraints are not scaled. In the experience of the author, this method is the most consistent scaling method. In addition, it only applies to the states and controls, which are the only elements of an OCP that are usually scaled using canonical or designer units.

5.6.2 Adverse Effects of Scaling

An unfortunate side effect of any scaling method can be seen when examining the costate solution for a scaled OCP [151]. Consider the Orbit-Raising problem, defined in Section 6.2. This problem was solved twice, once with scaling, and once without scaling. The costate solutions for each case are shown in Fig. 5.9. It is evident by comparing Fig. 5.9a to Fig. 5.9b



(a) Non-scaled Solution

(b) Scaled Solution

Figure 5.9: Orbit-Raising Costate Solutions

that using the isoscaling method immediately doubles the maximum magnitudes of the costate solutions. For this particular problem, large costate values do not cause a problem, as the maximum values are still close to zero. However, for some problems (such as the Earth to Dionysus problem shown in Section 6.3), the scaled costate values can become extremely large, on the order of 10^9. This produces issues in the discrete Lagrangian of the problem, given by,

$$\mathcal{L} = \sigma J + \boldsymbol{\lambda}^\top \boldsymbol{c}. \tag{5.124}$$

In Eq. (5.124), the dot product operation given by $\boldsymbol{\lambda}^\top \boldsymbol{c}$ involves multiplying a value on the order of 10^9 ($\boldsymbol{\lambda}_i$) by a value on the order of 1 ($\boldsymbol{c}_i$). This value is then summed with other values that have undergone similar multiplications. Due to the limits of floating-point precision, significant digits are lost in this operation due to truncation error. This results in a loss of constraint information and potential non-convergence of the solver or convergence to a pseudo-minimizer. [151] suggests selecting "designer" units to scale the problem, solving the problem on a rough mesh, and then manually iterating on these scaling coefficients until the magnitudes of the state and costate are roughly equal. This is referred to as "balancing." In theory, balancing will maximize the performance of the solver. However, it requires significant user input and is primarily applicable to direct methods. The user should be aware of this numerical issue present in all constrained OCPs.

5.6.3 Projected-Jacobian Rows Normalization Scaling

An interesting type of scaling that the author has encountered is the Projected-Jacobian Rows Normalization (PJRN) scaling method, introduced by Sagliano [154]. The study mentioned considers the condition number of the Jacobian as a measure of the quality of a scaling method. This is because the Jacobian defines the search direction for gradient-descent solvers during the iterative solution process. Thus, the logic follows that a well-conditioned Jacobian will produce fewer rounding errors than a poorly-conditioned Jacobian and provide a more accurate search direction. The aim of the scaling method is to scale the Jacobian such that the 2-norm of each row has a magnitude equal to one. This is accomplished as follows. Assume the states are scaled according to Eq. (5.114) and Eq. (5.115). Also consider an NLP Jacobian given by,

$$\text{Jac} = \begin{bmatrix} \nabla_{\boldsymbol{z}} J \\ \nabla_{\boldsymbol{z}} \boldsymbol{\Delta} \\ \nabla_{\boldsymbol{z}} \boldsymbol{G} \end{bmatrix}, \tag{5.125}$$

where we have included the objective gradient in the expression. This is for notational convenience. In addition, we split the constraint Jacobian $\nabla_{\boldsymbol{z}} \boldsymbol{C}$ into the defect constraints, $\nabla_{\boldsymbol{z}} \boldsymbol{\Delta}$ and non-defect constraints, $\nabla_{\boldsymbol{z}} \boldsymbol{G}$. Recall that the defect and non-defect constraints are column vectors. Consider scaling each element of the derivatives as,

$$\widetilde{J} = K_J J, \tag{5.126}$$

$$\widetilde{\boldsymbol{\Delta}} = \boldsymbol{K_f} \boldsymbol{\Delta}, \tag{5.127}$$

$$\widetilde{\boldsymbol{G}} = \boldsymbol{K_g} \boldsymbol{G}. \tag{5.128}$$

Here, K_j is a parameter, which normalizes the cost function, J, while $\boldsymbol{K_f} \in \mathbb{R}^{N_t N_x \times N_t N_x}$ and $\boldsymbol{K_g} \in \mathbb{R}^{N_t N_g \times N_t N_g}$ are diagonal scaling matrices for the defect constraints and the non-defect constraints, respectively. In the PJRN method, the diagonal matrices $\boldsymbol{K_f}$ and $\boldsymbol{K_g}$ are selected as,

$$\boldsymbol{K}_{\boldsymbol{f}_{ii}} = \frac{1}{\|(\nabla_{\boldsymbol{z}} \boldsymbol{\Delta} \cdot \boldsymbol{K}_{\boldsymbol{x}}^{-1})_i\|_2}, \qquad \boldsymbol{K}_{\boldsymbol{g}_{ii}} = \frac{1}{\|(\nabla_{\boldsymbol{z}} \boldsymbol{G} \cdot \boldsymbol{K}_{\boldsymbol{x}}^{-1})_i\|_2}, \tag{5.129}$$

where $(\cdot)_i$ denotes the i-th *row* of the matrix produced by the matrix multiplication in the denominator. Note that all off-diagonal elements of $\boldsymbol{K_f}$ and $\boldsymbol{K_g}$ are zero. Although K_J is often a user-defined parameter [155], it can be computed as,

$$K_j = \frac{1}{\|(\nabla_{\boldsymbol{z}} J \cdot \boldsymbol{K}_{\boldsymbol{x}}^{-1})_i\|_2}, \tag{5.130}$$

in the case of no manual user scaling. The scaled Jacobian of the PJRN method is given by,

$$\widetilde{\text{Jac}} = \begin{bmatrix} \widetilde{\nabla}_{\boldsymbol{z}} \widetilde{J} \\ \widetilde{\nabla}_{\boldsymbol{z}} \widetilde{\boldsymbol{\Delta}} \\ \widetilde{\nabla}_{\boldsymbol{z}} \widetilde{\boldsymbol{G}} \end{bmatrix} = \begin{bmatrix} K_J \cdot \nabla_{\boldsymbol{z}} J \cdot \boldsymbol{K}_{\boldsymbol{x}}^{-1} \\ \boldsymbol{K_f} \cdot \nabla_{\boldsymbol{z}} \boldsymbol{\Delta} \cdot \boldsymbol{K}_{\boldsymbol{x}}^{-1} \\ \boldsymbol{K_g} \cdot \nabla_{\boldsymbol{z}} \boldsymbol{G} \cdot \boldsymbol{K}_{\boldsymbol{x}}^{-1} \end{bmatrix}. \tag{5.131}$$

Thus, in the PJRN method, the objective and constraints are scaled according to Eq. (5.131). The result of Eq. (5.131) is a Jacobian whose rows have a norm equal to one and whose condition number is very low. In addition, [156] introduces a transformation for the KKT multipliers, given by,

$$\Lambda = \left[K_J \boldsymbol{K}_{\boldsymbol{f}}^{-1}\right]^{-1} \tilde{\Lambda}, \qquad \Gamma = \left[K_j \boldsymbol{K}_{\boldsymbol{g}}^{-1}\right]^{-1} \tilde{\Gamma}, \tag{5.132}$$

where $\tilde{\lambda}$ and $\tilde{\gamma}$ are the KKT multipliers associated with the PJRN scaled problem. The continuous multipliers may be obtained using expressions given in Section 4.5.

In the qualitative experience of the author, this method seems to be less effective than the affine scaling method for many problems. It is less computationally expensive than other Jacobian normalization methods, such as the one described in Rao et al. [141], Patterson and Rao [130], which evaluates the Jacobian at randomly sampled points distributed about $\boldsymbol{z}$ and obtains the scaling terms as,

$$\boldsymbol{K}_{\boldsymbol{f}_{ii}} = \text{mean}\frac{1}{\|(\nabla_{\boldsymbol{z}}\Delta)_i\|}, \qquad \boldsymbol{K}_{\boldsymbol{g}_{ii}} = \text{mean}\frac{1}{\|(\nabla_{\boldsymbol{z}}G)_i\|}, \tag{5.133}$$

where the mean is evaluated for the number of randomly sampled points. In the experience of the author, PJRN scaling can work extremely well for increasing computational speed for some problems. However, the PJRN method seems to suffer from instability. By instability, the author means that when using scaling methods for which the scaling coefficients are different for each time point, an NLP solver may converge to an infeasible or suboptimal solution for a given problem. Ross et al. [151] suggests that this instability is due to the presence of **unaccounted terms for additional dynamics** that result from the use of an implicitly time-varying scaling coefficient, which is the case for the PJRN and averaging methods just discussed. This is because when transforming a scaled state to the continuous domain, one obtains an expression of the form,

$$\tilde{\boldsymbol{x}}(t) = \boldsymbol{K}_{\boldsymbol{x}}(t)\boldsymbol{x}(t) + \boldsymbol{b}(t), \tag{5.134}$$

where $\boldsymbol{K}_{\boldsymbol{x}}(t)$ and $\boldsymbol{b}(t)$ become functions of time due to the fact that the scaling coefficients are different for each value of $\boldsymbol{x}(t_i)$. Thus, there is an implicit time dependence for the scaling coefficients. The result of differentiating Eq. (5.134) is given as,

$$\frac{d\tilde{\boldsymbol{x}}(t)}{dt} = \boldsymbol{K}_{\boldsymbol{x}}(t)\frac{d\boldsymbol{x}(t)}{dt} + \frac{d\boldsymbol{K}_{\boldsymbol{x}}(\boldsymbol{t})}{dt}\boldsymbol{x}(t) + \frac{d\boldsymbol{b}(t)}{dt}. \tag{5.135}$$

In Eq. (5.135), the derivatives of $\boldsymbol{K}_{\boldsymbol{x}}$ and $\boldsymbol{b}$ no longer vanish because they are *no longer constants with respect to time*. There is no clear way to approximate these additional derivative terms. In the PJRN and averaging methods, the new time-varying terms are not accounted for in the dynamics due to the assumption that the derivatives of $\boldsymbol{K}_{\boldsymbol{z}}$ and $\boldsymbol{b}_{\boldsymbol{z}}$ with respect to time vanish. For this reason, the author has only implemented an affine (constant $\boldsymbol{K}$ and $\boldsymbol{b}$) automatic scaling routine in TOPS.

Chapter 6

Example Problems

In this section, several example problems will be solved in TOPS and compared to their indirect solutions. The problems will increase in difficulty, beginning with a very simple problem with a closed-form solution to a state-of-the-art variable-specific-impulse propulsion system model for the orbit transfer problems. The author would like to acknowledge that Dr. Daniel Herber's open-source basic PS solver [78] was extremely useful for the author when learning PS methods. The author has built TOPS to be both an educational tool to those new to PS methods as well as a practical, effective open-source PS software. All solutions obtained in this section were obtained on a Surface Book 2 Windows Laptop using an Intel(R) Core(TM) i7-8650U CPU @ 1.90 GHz clock speed with 8.0 GB of VRAM.

6.1 Moon-Lander Problem

The first example problem is one that has been used as a demonstration problem throughout the paper. The problem is given in [114] as an optimal thrust soft lunar landing problem. It is a one-dimensional problem with simple dynamics and a discontinuous control profile exhibiting a single bang-on arc. The problem is stated as follows.

$$\boldsymbol{x}(t) = [h, v], \;\; \boldsymbol{u}(t) = u, \;\; t_f,$$

$$(\mathrm{ML}) \begin{cases} \text{Minimize} \quad J\left[\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot), t_f\right] = \displaystyle\int_{t_0}^{t_f} u(t)\, dt, \\ \text{Subject to :} \; \dot{h} = v, \\ \qquad\qquad\quad\;\; \dot{v} = -g + u, \\ \qquad\qquad\quad\;\; 0 \leq u \leq 3, \\ \qquad\qquad\quad\;\; (h_0, v_0, t_0) = (10\; m, -2\; m/s, 0), \\ \qquad\qquad\quad\;\; (h_f, v_f, t_f) = (0, 0, free)\,. \end{cases}$$

Here, $g = 1.5\; m/s^2$. Note also that t_f is free, although this is not a minimum-time problem. Usually, leaving t_f unconstrained for such cases will cause $t_f \to \infty$ (see p.32 of [145]), but in this case the dynamics naturally constrain the system to satisfy the boundary conditions in a finite time. The exact solution to this problem is given by,

$$h^*(t) = \begin{cases} -\frac{3}{4}t^2 + v_0 t + h_0, & t \leq s^*, \\ \frac{3}{4}t^2 + (-3s^* + v_0)t + \frac{3}{2}(s^*)^2 + h_0, & t \geq s^*, \end{cases} \tag{6.1}$$

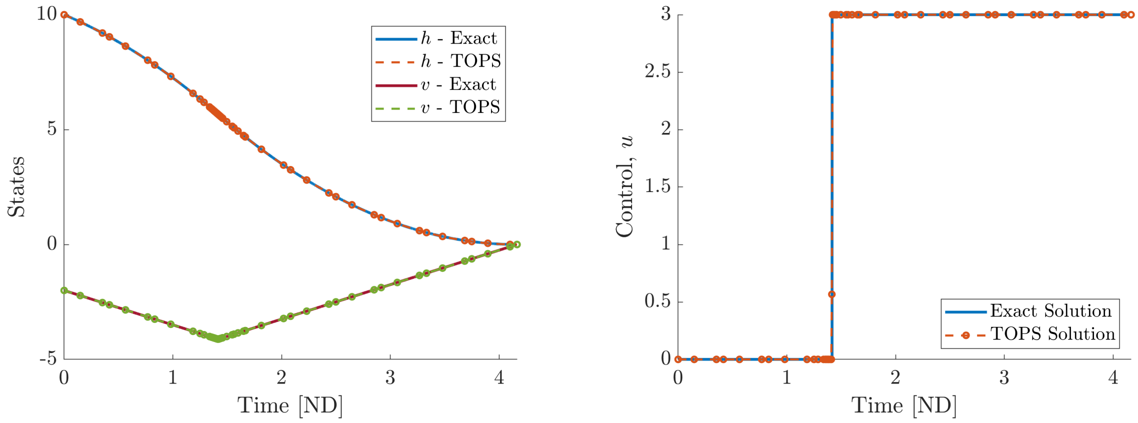
$$v^*(t) = \begin{cases} -\frac{3}{2}t + v_0, & t \leq s^*, \\ \frac{3}{2}t - 3s^* + v_0, & t \geq s^*, \end{cases} \tag{6.2}$$

$$u^*(t) = \begin{cases} 0, & t \leq s^*, \\ 3, & t \geq s^*, \end{cases} \tag{6.3}$$

where s^* is the single switch time, which is given by,

$$s^* = \frac{t_f^*}{2} + \frac{v_0}{3}, \qquad t_f^* = \frac{2}{3}v_0 + \frac{4}{3}\sqrt{\frac{1}{2}v_0^2 + \frac{3}{2}h_0}. \tag{6.4}$$

It can easily be calculated that for the boundary conditions given in Problem (ML), the problem has a switch and final time of $(s^*, t_f^*) = (1.4154, 4.1641)$. This is a "bang-bang"



(a) State Solution

(b) Control Solution

Figure 6.1: Exact and TOPS solutions to the moon-lander problem.

optimal control problem, in that the control is discontinuous and switches between its maximum and minimum values. This problem was solved in TOPS using the ph-adaptive mesh-refinement algorithm with $N_{\min} = 3$ and $N_{\max} = 12$ with 3 evenly spaced initial segments. The mesh accuracy requested was $\varepsilon = 10^{-8}$. The constraint violation and optimality tolerances were both set to 10^{-6}. Linear interpolation between the boundary conditions was used to provide an initial guess. The problem was solved using fmincon, SNOPT, and IPOPT all in quasi-Newton (first derivative) mode. This is because exact vectorized derivative calculations for free final time problems have not yet been implemented in TOPS. All first derivatives were provided by *ADiGator*. Table 6.1 contains statistics for each NLP solver that was used to solve this problem. The solution obtained using SNOPT is shown in Fig. 6.1. Note in Table 6.1 that IPOPT does not obtain a solution as quickly as fmincon or SNOPT, which are both using SQP Hessian approximation algorithms. This is expected. However, running IPOPT in non-vectorized second derivative mode for this problem actually increased the CPU time to approximately 4 seconds. This is due to the inefficiency of the non-vectorized AD derivatives.

Table 6.1: Moon-lander problem solution comparisons.

Solver	J^*	N_{pts}	N_{seg}	Mesh Iter.	ϵ_{sol}	CPU Time [sec]
Exact	8.2462	-	-	-	-	-
fmincon	8.2462	47	15	9	9.191×10^{-9}	1.557
SNOPT	8.2462	49	15	9	5.223×10^{-10}	0.551
IPOPT	8.2462	47	15	9	9.214×10^{-9}	2.239

All NLP solvers produced solutions with a similar number of collocation points, N_{pts}, and the same number of mesh segments, N_{seg}. The mesh-refinement algorithm converged in 9 iterations for each solver, and the requested mesh accuracy was satisfied for each case. Fig. 6.1 shows the state and control solutions produced by SNOPT. The state was interpolated using Lagrange polynomials while the control was linearly interpolated. Note that when interpolated, the TOPS state and control solutions lie *exactly* on top of the exact solution. Also note in Fig. 6.1 that the collocation points are densely clustered near the location of the discontinuity, effectively capturing the bang-bang control switch. The final time produced by TOPS was $t_f = 4.1641$, which is equal to the final time for the exact solution.

6.2 Orbit-Raising Problem

The orbit-raising problem is a classic introductory optimization problem Bryson and Ho [23]. Many students who have studied orbital mechanics and optimal control are intimately familiar with it. The orbit-raising problem is given as follows.

$$\boldsymbol{x}(t) = [r, \theta, u, v], \quad \boldsymbol{u}(t) = \phi,$$

$$\text{(OR)} \begin{cases} \text{Minimize} & J\left[\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot), t_f\right] = -r(t_f), \\ \text{Subject to} & \dot{r} = u, \\ & \dot{\theta} = \dfrac{v}{r}, \\ & \dot{u} = \dfrac{v^2}{r} - \dfrac{1}{r^2} + A(t)\sin(\theta), \\ & \dot{v} = -\dfrac{uv}{r} + A(t)\cos(\theta), \\ & A(t) = \dfrac{T}{(m_0 - |\dot{m}|t)}, \\ & (r_0, \theta_0, u_0, v_0) = (1, 0, 0, 1), \\ & (u_f, v_f, t_f) = \left(0, \sqrt{1/r_f}, 3.32\right), \end{cases}$$

Here, $m_0 = 1$, $\dot{m} = 0.0749$, and $T = 0.1405$. All units are non-dimensionalized. The goal of this problem is to make a circular-to-circular orbit transfer in a set amount of time that maximizes the final orbital radius. There is no closed-form solution to this problem. However, the author has solved this problem using an indirect shooting method in order to compare the direct and indirect solutions. When solving this problem, no mesh refinement was used. This was for two reasons. The first is that this problem will be used to illustrate the differences in the costate estimates produced by the LGR and LGL transcriptions. The second is that the solution is known to be *smooth* [150], so in general mesh refinement can be achieved by simply increasing the degree of the interpolating polynomial. This problem was solved for a single mesh interval containing 60 collocation points. It was solved using fmincon, SNOPT, and IPOPT for LGR points, and once using SNOPT for LGL points. All first derivatives were supplied using *ADiGator*. Since this is not a free final time problem, vectorized second derivatives were supplied to IPOPT using *ADiGator*. Note from Table 6.2 that when providing vectorized second derivatives, IPOPT outperforms SNOPT. An initial guess was provided by linearly interpolating between boundary conditions. A comparison of the computational statistics and objective value obtained using each NLP solver are given in Table 6.2. The control solution

Table 6.2: Orbit-raising problem solutions comparison.

Solver	$r(t_f)$	N_{pts}	N_{seg}	Mesh Iter.	ϵ_{sol}	CPU Time [sec]
Indirect	1.5253	-	-	-	-	-
fmincon	1.5253	60	1	1	-	4.67
SNOPT-LGR	1.5253	60	1	1	-	1.47
IPOPT	1.5253	60	1	1	-	0.51
SNOPT-LGL	1.5253	60	1	1	-	1.70

obtained using the indirect shooting scheme and TOPS is shown in Fig. 6.2. Note in Fig. 6.2

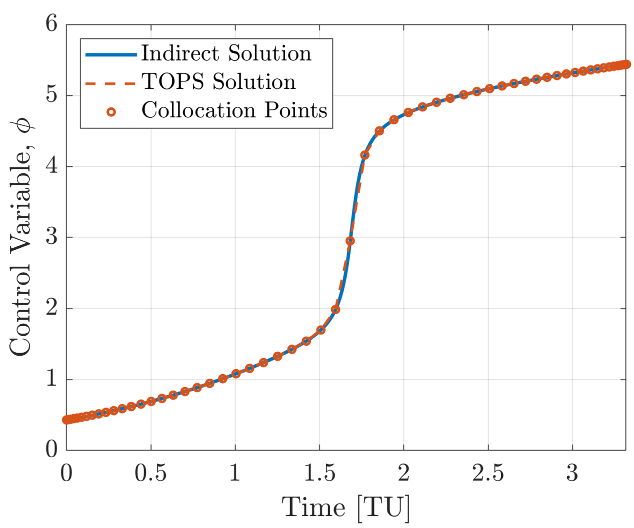
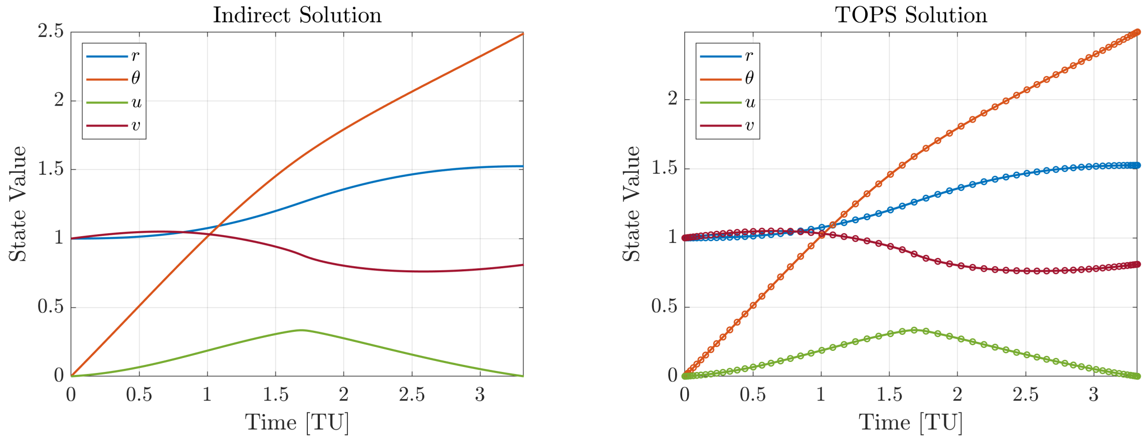


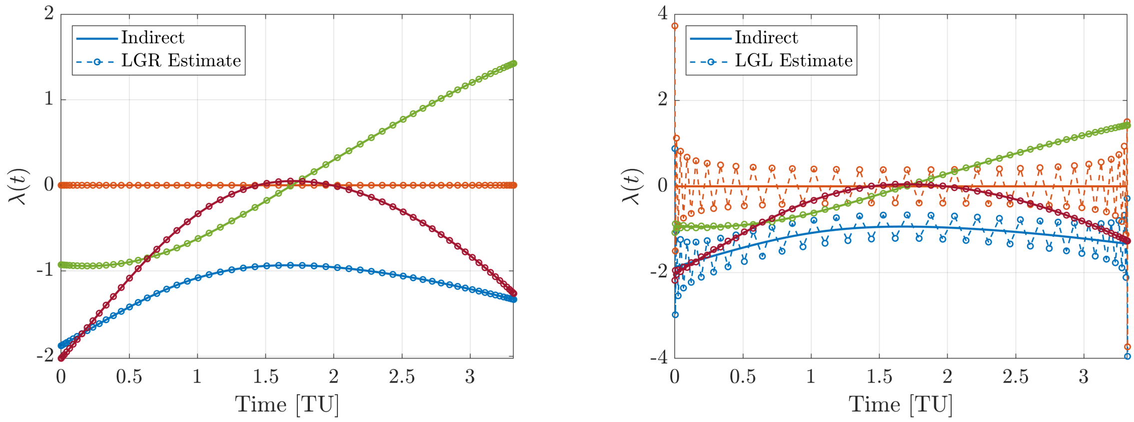
Figure 6.2: Control profile for indirect method and TOPS.

that the control solution obtained by TOPS lies nearly exactly on top of the indirect solution.



(a) Indirect State Solution (b) TOPS State Solution

Figure 6.3: Indirect and TOPS state solutions to the orbit-raising problem.



(a) LGL Costate Estimate (b) TOPS State Solution

Figure 6.4: Costate estimates obtained with the LGR and LGL PS methods.

In addition, Table 6.2 shows that the objective value computed by each solver was identical to the indirect objective value. Next, we examine the state profiles obtained. It is evident from Fig. 6.3 that the solutions obtained using the indirect method and TOPS are nearly identical. The state and control solutions obtained using the LGR and LGL transcriptions are identical. However, a significant difference arises when we examine the costate solutions obtained by the LGR and LGL transcriptions. Note that in Fig. 6.4a that the costate obtained by the LGR PS method lies exactly on the indirect solution. However, as soon as we employ the LGL method in Fig. 6.4b, the costates begin to exhibit oscillations or "chattering" about the optimal value. This is a well-known phenomenon, and the reason it occurs is discussed in Section 4.2.1.

Certain filtering techniques can be used to eliminate these oscillations [48]. However, this chattering phenomenon is known to occur in certain situations for the solution for LGR and LGL state/control solutions, even though the costates may converge to the optimal solution [52]. However, this primarily occurs for problems that exhibit singular arcs, and methods have been developed to overcome this issue [123].

6.3 Earth-to-Dionysus (E2D) Problem

The next problem that will be considered is the Earth to asteroid 3671 Dionysus minimum-fuel rendezvous problem. 3671 Dionysus is a potentially hazardous asteroid due to its minimum orbit intersection distance with the earth is less than 0.5 AU and it has a diameter greater than 150 meters. As such, it is of interest for unmanned low-thrust exploratory missions. This problem is presented in [171] and is given as,

$$\boldsymbol{x} := [p, f, g, h, k, L, m] \in \mathbb{R}^7, \ \ \hat{\boldsymbol{\alpha}} := [\hat{\alpha}_r, \hat{\alpha}_t, \hat{\alpha}_n, \delta] \in \mathbb{R}^4, \ \ t_f,$$

$$(\text{E2D}) \begin{cases} \text{Minimize} & J[\boldsymbol{x}(\cdot), t_f] = -m(t_f), \\ \text{Subject to :} & \dot{\boldsymbol{x}} = \begin{bmatrix} \mathbf{A}\left(\dfrac{T_{\max}}{m}\delta\hat{\boldsymbol{\alpha}}\right) + \boldsymbol{b} \\ -\dfrac{T_{\max}}{c}\delta \end{bmatrix}, \\ & \|\hat{\boldsymbol{\alpha}}\| = 1, \\ & \delta \in [0, 1], \\ & (\boldsymbol{x}_{\text{MEE}}(t_0), m(t_0), t_0) = (\boldsymbol{x}^0_{\text{MEE}}, m^0, t^0), \\ & (\boldsymbol{x}_{\text{MEE}}(t_f), t_f) = (\boldsymbol{x}^f_{\text{MEE}}, t^f), \end{cases}$$

Here, $\boldsymbol{x}_{\text{MEE}} = [p, f, g, h, k, L]$ are the Modified Equinoctial Elements (MEE), a set of non-singular orbital elements that are well-behaved for long-duration low-thrust problems [171, 86].

The matrices $\boldsymbol{A}$ and $\boldsymbol{b}$ are given by,

$$\mathbf{A} = \begin{bmatrix} 0 & \frac{2p}{w}\sqrt{\frac{p}{\mu}} & 0 \\ \sqrt{\frac{p}{\mu}}\sin(L) & \sqrt{\frac{p}{\mu}}\frac{1}{w}\left[(w+1)\cos(L)+f\right] & -\sqrt{\frac{p}{\mu}}\frac{g}{w}\kappa \\ -\sqrt{\frac{p}{\mu}}\cos(L) & \sqrt{\frac{p}{\mu}}\frac{1}{w}\left[(w+1)\sin(L)+g\right] & \sqrt{\frac{p}{\mu}}\frac{f}{w}\kappa \\ 0 & 0 & \sqrt{\frac{p}{\mu}}\frac{s^2\cos(L)}{2w} \\ 0 & 0 & \sqrt{\frac{p}{\mu}}\frac{s^2\sin(L)}{2w} \\ 0 & 0 & \sqrt{\frac{p}{\mu}}\frac{1}{w}\kappa \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \sqrt{\mu p}\left(\frac{w}{p}\right)^2 \end{bmatrix}, \quad (6.5)$$

where $w = 1 + f\cos(L) + g\sin(L)$, $s^2 = 1 + h^2 + k^2$, $\kappa = h\sin(L) - k\cos(L)$, and μ is the gravitational parameter of the central body. For the purposes of this study, the central body is Sun and its gravitational parameter is denoted μ_s. The position, $\boldsymbol{r}$, and velocity, $\boldsymbol{v}$, of the spacecraft relative to a Sun-centered inertial (SCI) frame can be expressed in terms of the MEE set [17] as,

$$\boldsymbol{r} = \begin{bmatrix} \frac{p}{ws^2}\left[\cos(L) + \alpha\cos(L) + 2hk\sin(L)\right] \\ \frac{p}{ws^2}\left[\sin(L) - \alpha\sin(L) + 2hk\cos(L)\right] \\ \frac{2p}{ws^2}(h-k) \end{bmatrix}, \quad (6.6)$$

$$\boldsymbol{v} = \begin{bmatrix} -M\left[\sin(L) + \alpha\sin(L) - 2hk\cos(L) + g - 2fhk + \alpha g\right] \\ -M\left[-\cos(L) + \alpha\cos(L) + 2hk\sin(L) - f + 2ghk + \alpha f\right] \\ 2M\left[h\cos(L) + k\sin(L) + fh + gk\right] \end{bmatrix}, \quad (6.7)$$

where $\alpha = h^2 - k^2$ and $M = \frac{1}{s^2\sqrt{p}}$. The problem parameters are given in Table 6.3. This prob-

Table 6.3: Earth to Dionysus problem data.

Parameter	Value	Units
μ_s	132712440018	$[km^3/s^2]$
AU	149.6×10^6	$[km]$
g_0	9.8065	$[m/s^2]$
m_0	4000	$[kg]$
$T_{\max}$	0.32	$[N]$
I_{sp}	3000	$[sec]$
N_{rev}	5	$[-]$
t^f	3534	$[days]$

lem was manually scaled using canonical units, as the automatic scaling method destabilized the solution. The canonical units were selected as follows.

$$\text{DU} = 1 \text{ AU} \qquad \text{TU} = \sqrt{\frac{\text{DU}^3}{\mu_s}} = 5.0228 \times 10^6 \; [sec] \tag{6.8}$$

All EOMs, boundary conditions, and flight times were scaled using these units. The scaled boundary conditions are given by,

$$\boldsymbol{x}^0_{\text{MEE}} = \begin{bmatrix} 0.99969 \text{ AU} \\ -3.7668 \times 10^{-3} \\ 0.016287 \\ -7.70206 \times 10^{-6} \\ 6.18817 \times 10^{-7} \\ 14.16189 \end{bmatrix}, \qquad \boldsymbol{x}^f_{\text{MEE}} = \begin{bmatrix} 1.55370 \text{ AU} \\ 0.15303 \\ -0.51995 \\ 0.016183 \\ 0.11814 \\ 46.33024 \end{bmatrix}. \tag{6.9}$$

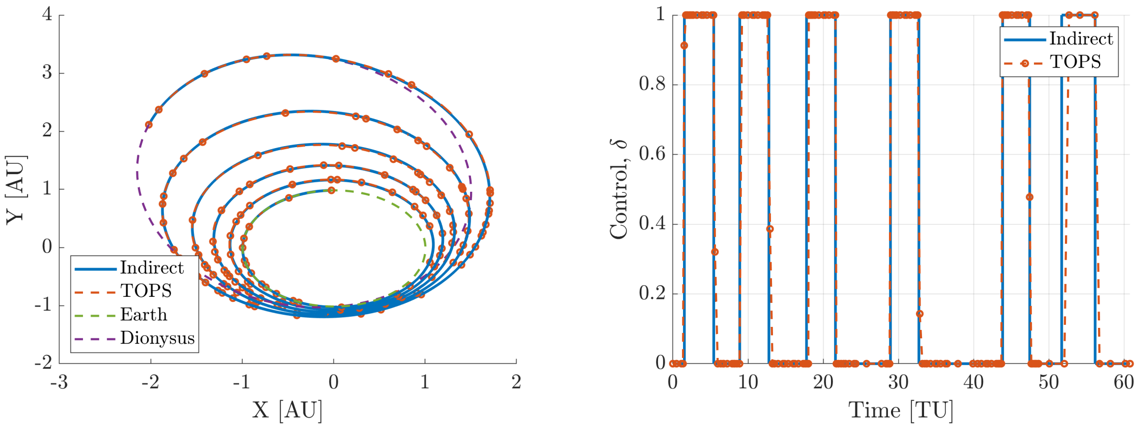
This is a benchmark problem and is relatively difficult to solve. As such, TOPS took much longer to reach a solution. In addition to this, the only solver that could obtain a solution in a reasonable amount of time was SNOPT. This will most likely change with the full implementation of vectorized second derivatives. This problem belongs to the class of low-thrust long-duration trajectory optimization problems, which are numerically difficult to solve due to the long transfer time and the existence of several orbital revolutions around the Sun [170].

In order to solve this problem, an initial mesh was selected using 15 mesh segments with 3 collocation points in each segment. The mesh parameters were $N_{\min} = 3$ and $N_{\max} = 6$. The mesh accuracy was selected to be $\varepsilon = 10^{-4}$. Linear interpolation between boundary conditions was used to provide an initial guess. Table 6.4 shows the solution statistics for the problem. Fig. 6.5 shows the E2D transfer trajectory and throttle history produced by the indirect method

Table 6.4: Earth to dionysus problem solution information.

Solver	$m(t_f)$ $[kg]$	N_{pts}	N_{seg}	Mesh Iter.	ϵ_{sol}	CPU Time [sec]
Indirect	2716.21	-	-	-	-	-
TOPS (SNOPT)	2718.33	151	45	5	9.83×10^{-5}	296.91

and TOPS. Note in Fig. 6.5a that the indirect and TOPS solutions are visually indistinguishable from one another. However, when we look at Fig. 6.5b, we see that the final thrust arc is

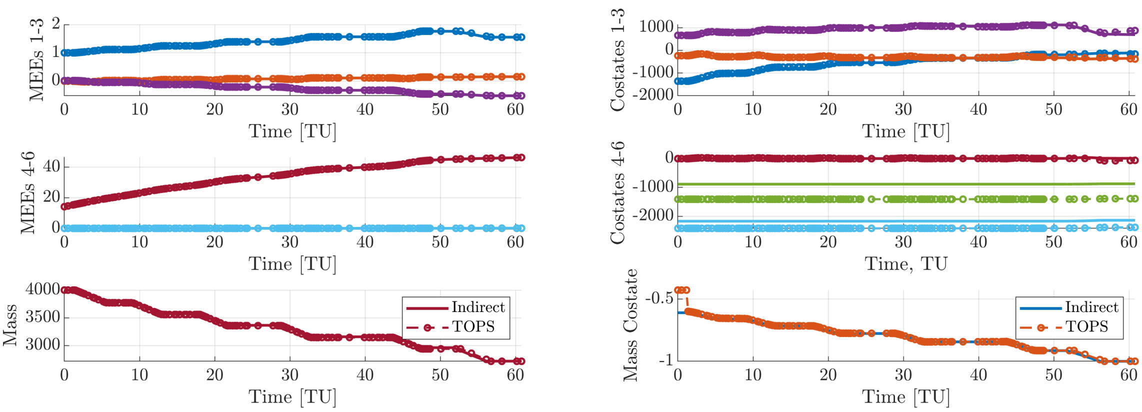


(a) E2D Trajectory Solutions.

(b) E2D Throttle Solutions.

Figure 6.5: Trajectory and Throttle Solutions for the E2D Problem.

not as effectively captured as the previous thrust arcs. This is an inherent shortcoming in the mesh refinement process. Reducing the mesh error tolerance too low can cause unnecessary points to be thrown and drastically increase the solution time, while too high of a mesh error tolerance can cause poor quality solutions. This solution struck a balance between quality and computational time, and the solution objective values given in Table 6.3 show a difference of approximately two kilograms. Fig. 6.6 shows the indirect and TOPS solutions for the state and costate time histories. Note in Fig. 6.6b that the costate time histories differ by a significant



(a) E2D MEE Time History Solutions.

(b) E2D Costate Solutions.

Figure 6.6: State and Costate Solutions for the E2D Problem.

margin for the costates associated with the h and k MEEs. This may be due to convergence to a local extrema or numerical difficulties associated with the Lagrangian Hessian approximation internally calculated by SNOPT. Regardless of this difference, the solution produced by TOPS is an excellent preliminary trajectory planning solution that can be further refined using indirect methods. This difficulty in solving a long-duration low-thrust problem is to be expected for pseudospectral methods, but the advent of Birkoff-based PS methods may alleviate this difficulty [144].

6.4 Satellite Constellation Formation Problem

The satellite constellation problem is an interesting problem that was solved by the author using an indirect method in [163]. This problem considers the case of two "deputy" or "chaser" satellites being deployed by a launch vehicle into a low Earth orbit (LEO) . The objective of the chaser satellites is to insert themselves into the orbit of a chief satellite that has already been deployed. The chaser satellites attempt to make a minimum-fuel maneuver such that the final orbit of the first chaser is the chief's orbit plus a phase angle of one degree (a leading orbit), while the second chaser assumes the chief's orbit mines a phase angle of one degree (a trailing orbit). The inspiration for this problem is the deployment of Starlink satellites that perform similar maneuvers to form satellite constellation "chains." The aim of these chains is to provide constant satellite coverage for a particular latitude.

This constellation formation problem is simplified here to the case of one deputy assuming a leading orbit. However, two-deputy indirect solutions can be found in [163]. An interesting aspect of the problem is that it is solved in a Local-Vertical-Local-Horizontal (LVLH) frame of reference. This frame of reference is shown in Fig. 6.7 relative to an Earth-Centered-Inertial (ECI) frame. As can be seen from Fig. 6.7, this is a non-inertial frame of reference defined by the position and velocity of the chief in its orbit. The EOMs for this frame of reference are complex and non-intuitive, and are given in Problem (Sat). The problem statement is given

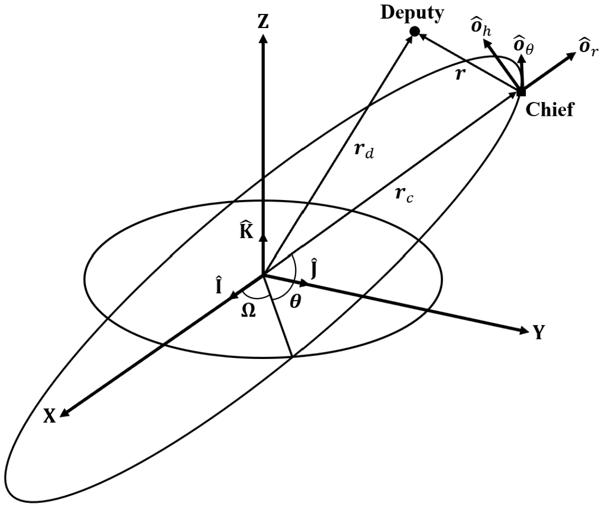


Figure 6.7: The ECI:$\{\hat{I}, \hat{J}, \hat{K}\}$ and co-moving LVLH:$\{\hat{\boldsymbol{o}}_r, \hat{\boldsymbol{o}}_\theta, \hat{\boldsymbol{o}}_h\}$ frames.

below.

$$\boldsymbol{x} := [x, y, z, v_x, v_y, v_z] \in \mathbb{R}^6, \;\; m \in \mathbb{R}, \;\; \boldsymbol{u} := [\hat{\boldsymbol{\alpha}}_r, \hat{\boldsymbol{\alpha}}_t, \hat{\boldsymbol{\alpha}}_n, \delta] \in \mathbb{R}^4, \;\; t_f,$$

$$(\text{Sat}) \begin{cases} \text{Minimize} & J\left[\boldsymbol{x}(\cdot), t_f\right] = -m(t_f), \\ \text{Subject to :} & \dot{\boldsymbol{x}} = \begin{bmatrix} v_x + y\dot{\theta}_c \\ v_y - x\dot{\theta}_c \\ v_z \\ v_y\dot{\theta}_c - \frac{\mu(r_c+x)}{\|\boldsymbol{r}_c+\boldsymbol{r}\|^3} + \frac{\mu}{r_c^2} \\ -v_x\dot{\theta}_c - \frac{\mu}{\|\boldsymbol{r}_c+\boldsymbol{r}\|^3}y \\ -\frac{\mu}{\|\boldsymbol{r}_c+\boldsymbol{r}\|^3}z \end{bmatrix} + \frac{T_{\max}}{m}\delta\boldsymbol{u} + \boldsymbol{a}_{J_2} = \boldsymbol{f}_{\boldsymbol{v}}, \\ & \|\boldsymbol{u}\| = 1, \\ & \delta \in [0, 1], \\ & (\boldsymbol{x}(t_0), m(t_0), t_0) = (\boldsymbol{x}^0, m^0, t^0), \\ & (\boldsymbol{x}(t_f), t_f) = (\boldsymbol{x}^f, t^f), \end{cases}$$

In Problem (Sat), $\dot{\theta}_c$ is the angular velocity of the chief satellite and $\boldsymbol{r}_c$ is the position vector of the chief relative to the Earth's center expressed in the LVLH frame. Here, the chaser's position vector $\boldsymbol{r}_d$, the position vector of the chief $\boldsymbol{r}_c$, the angular velocity vector of the chief $\boldsymbol{\omega}_c$, the relative position vector of the deputy $\boldsymbol{r} = \boldsymbol{r}_d - \boldsymbol{r}_c$, and the control $\boldsymbol{u}$ are given along the bases of the LVLH frame as

$$
\begin{aligned}
&\boldsymbol{r}_d = (r_c + x)\hat{\boldsymbol{o}}_r + y\hat{\boldsymbol{o}}_\theta + z\hat{\boldsymbol{o}}_h, && \boldsymbol{r}_c = r_c\hat{\boldsymbol{o}}_r, \\
&\boldsymbol{r} = x\hat{\boldsymbol{o}}_r + y\hat{\boldsymbol{o}}_\theta + z\hat{\boldsymbol{o}}_h, && \boldsymbol{\omega}_c = \dot{\theta}_c\hat{\boldsymbol{o}}_h, \\
&\boldsymbol{u} = u_x\hat{\boldsymbol{o}}_r + u_y\hat{\boldsymbol{o}}_\theta + u_z\hat{\boldsymbol{o}}_h.
\end{aligned}
\tag{6.10}
$$

We assume a circular, 550 kilometer equatorial chief orbit that is unperturbed, so $\dot{\theta}_c$ and r_c are constants. In addition, $\boldsymbol{a}_{J_2}$ is the acceleration due to the J_2 zonal harmonic, which is the dominant perturbation for LEO satellites [193, 32]. Expressions for the J_2 acceleration in the ECI frame can be found in Curtis [32], and the standard Euler 3-1-3 rotation sequence can be used to map the perturbations in the LVLH frame. However, it was found in [163] that the J_2 perturbation had a negligible effect on the solution for this problem, so it is ignored. The problem data is given in Table 6.5. The time of flight was selected to be 8 orbital periods of the chief. This problem was scaled similarly to Section 6.3 using canonical units with $\mathrm{DU} = R_e$.

Table 6.5: Constellation problem data.

Parameter	Value	Units
μ	398600	$[km^3/s^2]$
R_e	6371	$[km]$
r_c	550	$[km]$
$\dot{\theta}_c$	0.0011	$[1/sec]$
g_0	9.8065	$[m/s^2]$
m_0	260	$[kg]$
$T_{\max}$	0.5	$[N]$
I_{sp}	2000	$[sec]$
t^f	8	[Chief Periods]

A phase angle of $\phi = 1°$ was selected. The *unscaled* initial conditions in the LVLH frame are given by,

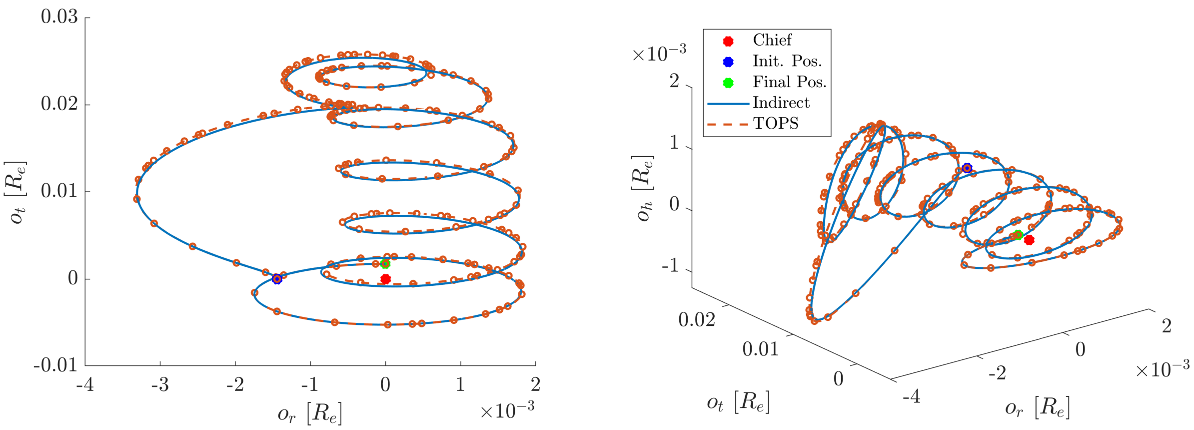
$$
\boldsymbol{x}_0 = \begin{bmatrix} -10\,km \\ 0\,km \\ 10\,km \\ 0\,km/s \\ 0\,km/s \\ 0\,km/s \end{bmatrix}, \qquad \boldsymbol{x}_f \begin{bmatrix} r_c\cos(\phi) - r_c \\ r_c\sin(\phi) \\ 0\,km \\ 0\,km/s \\ 0\,km/s \\ 0\,km/s \end{bmatrix}.
\tag{6.11}
$$

To solve this problem, an initial mesh was selected with 15 evenly distributed segments with $N_{\min} = 3$ and $N_{\max} = 9$. The requested mesh accuracy was $\varepsilon = 10^{-6}$. First derivatives were supplied using AD and linear interpolation between the boundary conditions was used as an initial guess. Note that this was an intentionally poor guess for a Cartesian space but the solver still converged to the correct solution. This problem is a better example of the capabilities of TOPS for shorter time-horizon trajectory optimization problems. The solution statistics are shown in Table 6.6. From Table 6.5, it is evident that there is difference in objective values

Table 6.6: Constellation minimum-fuel problem solution information.

Solver	$m(t_f)$ $[kg]$	N_{pts}	N_{seg}	Mesh Iter.	ϵ_{sol}	CPU Time [sec]
Indirect	259.7221	-	-	-	-	-
TOPS (SNOPT)	259.7217	173	36	8	9.567×10^{-7}	82.67

obtained by an indirect shooting method and TOPS are negligible. Fig. 6.8 shows the 2D and 3D minimum-fuel trajectories obtained using both methods. Note in Fig. 6.8b that the



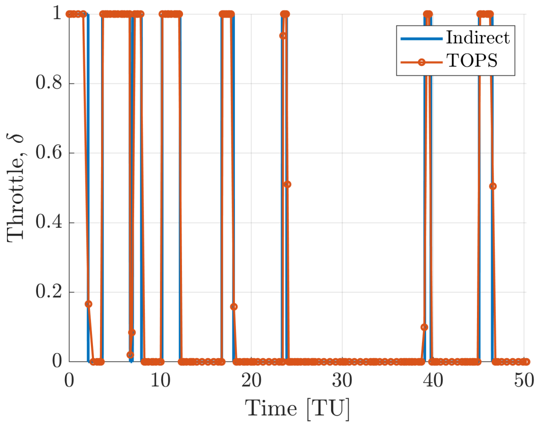
(a) 2D (Overhead) Trajectory.

(b) 3D Trajectory.

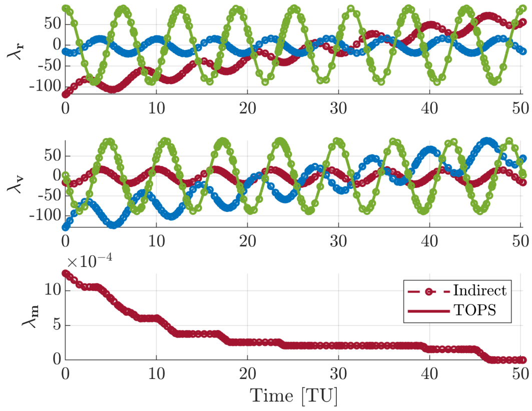
Figure 6.8: LVLH Trajectory Solutions.

trajectories lie nearly exactly on top of one another. The initial and final position labels in the legend of Fig. 6.8 are the initial and final positions of the deputy. The deputy is deployed in a lower orbit than the chief. Since the deputy is in a lower-altitude orbit (and hence, on a faster orbit than the chief), it needs to overtake the chief. Thus, the deputy immediately drops its orbit. This increases the relative velocity of the deputy and begins to increase the phase angle between the chief and the chaser. The deputy then increases its orbital radius and uses the relative orbital motion to "loop" back to the chief and assume its final position in the leading orbit.

Next, we examine the throttle and costate solutions obtained using each method. These are given in Fig. 6.9. Note from Fig. 6.9a that TOPS captured all thrust-coast arcs, including a



(a) Throttle Solution, δ.



(b) Costate Solutions.

Figure 6.9: Constellation Problem Control and Costate Solutions.

very short-duration coast arc around 8 [TU]. These short-duration arcs usually very difficult to capture when using CSC methods and caused some difficulty when solving this problem using the indirect shooting method. However, since the state values are arbitrary for a PS method, absolute discontinuities can sometimes be more easily captured on the "initial pass" of the direct method. Fig. 6.9b shows the costate solutions obtained by the indirect method and by TOPS. For this problem, the costates are captured *exactly* by the solver. This problem is an excellent demonstration of the capabilities of TOPS.

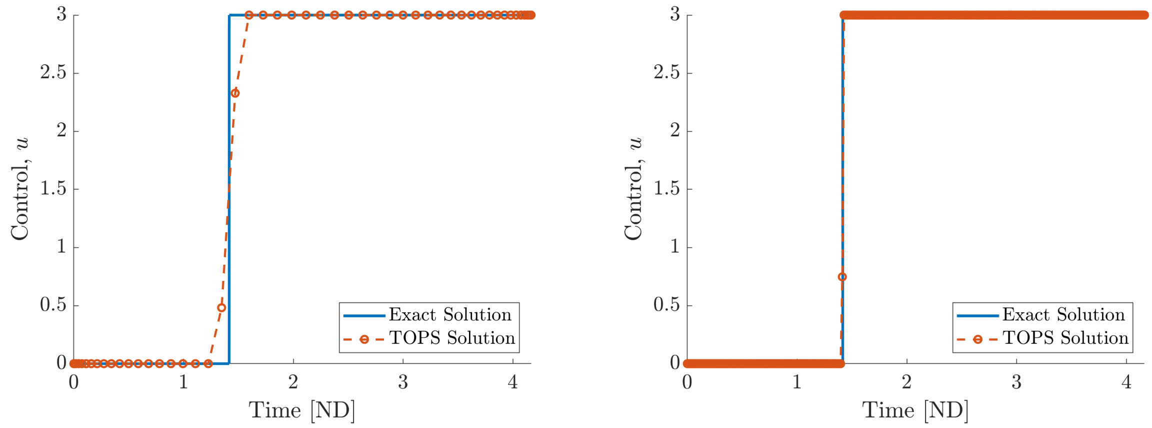
Chapter 7

Further Work

Over the past two decades, PS methods have become extremely mature and have become widely used throughout the engineering world due to their high-accuracy [51], sparse computational implementation [129], and both theoretical and in-practice guarantees of convergence [68, 149, 59]. In various aerospace applications, very accurate solutions [197, 70, 151] have been generated by PS methods using polynomial degrees of $N \leq 100$. The same is true for problems typically solved in TOPS. Due to their growth in popularity, PS methods have been used to solve increasingly difficult problems. Oftentimes, in order to achieve the desired levels of accuracy for these problems, polynomial degrees of $N \gg 100$ are required. This issue is present even for simple problems with discontinuous control solution. Consider the Moon-Lander problem given in Section 6.1. In order to obtain an accurate solution for this simple two-state problem using a single segment, a polynomial degree of $N \approx 400$ is required (see Fig. 7.1). However, this approach of strictly increasing the polynomial degree poses an issue for Lagrange-interpolant based PS methods since the condition number associated with the dynamics approximation,

$$\boldsymbol{D}\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{t}), \tag{7.1}$$

becomes very large as N increases. This is due primarily to the large condition number associated with the differentiation matrix $\boldsymbol{D}$, which grows according to $\mathcal{O}(N^2)$ for the Legendre PS methods [92]. For example, the condition number for the LGR differentiation matrix is approximately 10^3 for $N = 100$ (or $\mathcal{O}(N^2)$). To alleviate this issue, the problem must be split into multiple intervals and the polynomial degree decreased in each interval to reduce the condition



(a) $N = 50$ (b) $N = 400$

Figure 7.1: Single-Interval Solutions to the Moon Lander Problem

number of the differentiation matrix and obtain a solution [41, 131]. However, this process significantly reduces the rate of convergence of the problem, and the spectral convergence rate associated with PS methods can vanish entirely [92]. This greatly increases the computational load associated with a PS method. However, in recent years, a new breed of "well-conditioned" PS methods have been developed and introduced [122, 187, 38]. These new PS methods have transitioned from using Lagrange interpolants to using Birkoff interpolants, which are a generalization of Lagrange and Hermite interpolants [158]. **This new form of PS methods offers a reduction in the growth of the condition number of the dynamics approximation from $\mathcal{O}(N^2)$ to $\mathcal{O}(1)$** [92] for problems with known initial conditions. This seems to imply that the only form of mesh refinement needed when using a Birkhoff PS method is to *simply increase the number of collocation points*. In the general case, the condition number is $\mathcal{O}(\sqrt{N})$ for OCPs whose initial conditions are unknown. Much work has been done by Ross and Proulx [144], who have developed practical methods to compute the elements of a Birkoff-based PS method. In addition, they have developed a generalized framework for a Birkhoff PS method that is structurally very similar to the Lagrange PS methods. As such, they should be quite easy to implement for anyone familiar with PS methods. In fact, Ross and Proulx [144] show that Birkhoff-based methods exhibit identical behavior **regardless of the choice of Lagrange or Chebyshev transcription points**. This means that there is no difference between LGL, LGR, LG, and CGL solutions when using a Birkoff PS method. The reason for this is currently unknown, and is an open area of research. See Ross and Proulx [144] and Ross [146] for further information on the topic of Birkhoff PS methods. The authors of [144] have already implemented this Birkhoff PS method into the DIDO optimization toolbox [146] with promising results. In addition, [201] has used the Birkhoff PS method in combination with SCP methods to successfully solve a LEO rendezvous problem. **The primary focus of further work on TOPS should be to transition from the currently implemented Lagrange PS methods to Birkhoff PS methods.**

Another practical topic of research is the development of a novel automatic scaling and mesh-refinement algorithm that incorporates the "scaling and balancing" concepts introduced by Ross et al. [151]. The idea presented in this study is that scaling parameters should be chosen so that the magnitudes of the states and costate approximations obtained using the covector mapping theorem should be of approximately the same order. The paper introduces this as a process that must be performed by hand in order to increase the quality of the solution. The author of this study proposes that this process may be automated by deriving an approximate relationship between the scaling coefficients and the difference in magnitude of the state and costates. This approximate relationship could (in theory) be used to *automatically* select new values of scaling coefficients that will automatically re-scale the problem between mesh refinement iterations. This process may potentially be treated as a feedback control system, and appropriate systems control theory can possibly be applied to solve this problem.

An specific area where TOPS needs improvement is its mesh-refinement algorithm. The primary mesh-refinement algorithm used by TOPS is the *ph*-adaptive mesh-refinement algorithm [131] given in Section 5.2. This algorithm is not a very efficient algorithm and throws many unnecessary collocation points, although it is quite general. As such, a more efficient mesh-refinement algorithm would greatly benefit TOPS, specifically for solving bang-bang problems with many control switches. A mesh-refinement algorithm that exhibits excellent potential was introduced by Agamawi et al. [4]. This mesh-refinement algorithm is cutting-edge and produces efficient, accurate solutions for discontinuous problems. Additional mesh-refinement algorithm suggestions can be found throughout Section 5.2. In addition, TOPS currently employs vectorized AD second derivative calculations only for *fixed* final time problems. In the near future, the author plans to implement correct vectorized second derivative calculations for *free* final time problems.

A final area of improvement for TOPS is the implementation of "warm-starts" for the NLP solvers used by the software. This may be accomplished by providing the costate values obtained from a particular previous solution to a particular field of the NLP solver using a prolongation operator [67]. This may apparently increase the performance of the solver, but has not yet been implemented in TOPS.

References

[1] M. Abramowitz and I. A. Stegun. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, volume 55. US Government printing office, 1964.

[2] B. Acikmese and S. R. Ploen. Convex programming approach to powered descent guidance for mars landing. *Journal of Guidance, Control, and Dynamics*, 30(5):1353–1366, 2007.

[3] Y. M. Agamawi and A. V. Rao. Comparison of derivative estimation methods in optimal control using direct collocation. *AIAA Journal*, 58(1):341–354, 2020.

[4] Y. M. Agamawi, W. W. Hager, and A. V. Rao. Mesh refinement method for solving bang-bang optimal control problems using direct collocation. In *AIAA Scitech 2020 Forum*, page 0378, 2020.

[5] A. M. Aguirre-Mesa, M. J. Garcia, and H. Millwater. Multiz: a library for computation of high-order derivatives using multicomplex or multidual numbers. *ACM Transactions on Mathematical Software (TOMS)*, 46(3):1–30, 2020.

[6] E. L. Allgower and K. Georg. *Numerical continuation methods: an introduction*, volume 13. Springer Science & Business Media, 2012.

[7] V. Arya, E. Taheri, and J. Junkins. Electric thruster mode-pruning strategies for trajectory-propulsion co-optimization. *Aerospace Science and Technology*, 116:106828, Sept. 2021. ISSN 12709638. doi: 10.1016/j.ast.2021.106828. URL `https://linkinghub.elsevier.com/retrieve/pii/S1270963821003382`.

[8] V. Arya, E. Taheri, and J. L. Junkins. A composite framework for co-optimization of spacecraft trajectory and propulsion system. *Acta Astronautica*, 178:773–782, 2021.

[9] J. Aziz, D. Scheeres, J. Parker, and J. Englander. A smoothed eclipse model for solar electric propulsion trajectory optimization. *Transactions of the Japan Society for Aeronautical and Space Sciences, Aerospace Technology Japan*, pages 17–181, 2019.

[10] I. Babuška and M. Suri. The p-and hp versions of the finite element method, an overview. *Computer methods in applied mechanics and engineering*, 80(1-3):5–26, 1990.

[11] N. Bedrossian, S. Bhatt, M. Lammers, and L. Nguyen. Zero-propellant maneuver [tm] flight results for 180 deg iss rotation. In *Proceedings of the 20th International Symposium on Space Flight Dynamics*, 2007.

[12] 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*, 29(6):1435–1440, 2006.

[13] E. Bertolazzi. mexipopt: a re-write of the matlab ipopt interface for recent matlab releases. `https://github.com/ebertolazzi/mexIPOPT`. Accessed: 2022-11-02.

[14] R. Bertrand and R. Epenoy. New smoothing techniques for solving bang-bang optimal control problems - numerical results and statistical interpretation. *Optimal Control Applications and Methods*, 23(4):171–197, July 2002. ISSN 0143-2087, 1099-1514. doi: 10.1002/oca.709. URL `https://onlinelibrary.wiley.com/doi/10.1002/oca.709`.

[15] J. T. Betts. Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 21(2):193–207, 1998.

[16] J. T. Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2 edition, 2010.

[17] J. T. Betts. Optimal low–thrust orbit transfers with eclipsing. *Optimal Control Applications and Methods*, 36(2):218–240, 2015.

[18] J. T. Betts and W. P. Huffman. Mesh refinement in direct transcription methods for optimal control. *Optimal Control Applications and Methods*, 19(1):1–21, 1998.

[19] L. T. Biegler and V. M. Zavala. Large-scale nonlinear programming using ipopt: An integrating framework for enterprise-wide dynamic optimization. *Computers & Chemical Engineering*, 33(3):575–582, 2009.

[20] L. Blackmore. Autonomous precision landing of space rockets. In *Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2016 Symposium*, volume 46, pages 15–20. The Bridge Washington, DC, 2016.

[21] L. Blackmore, B. Açikmeşe, and D. P. Scharf. Minimum-landing-error powered-descent guidance for mars landing using convex optimization. *Journal of guidance, control, and dynamics*, 33(4):1161–1171, 2010.

[22] S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[23] A. E. Bryson and Y.-C. Ho. *Applied optimal control: optimization, estimation, and control*. Routledge, 2018.

[24] R. H. Byrd, J. Nocedal, and R. A. Waltz. K nitro: An integrated package for nonlinear optimization. In *Large-scale nonlinear optimization*, pages 35–59. Springer, 2006.

[25] J.-B. Caillau, B. Daoud, and J. Gergaud. Minimum fuel control of the planar circular restricted three-body problem. *Celestial Mechanics and Dynamical Astronomy*, 114(1): 137–150, 2012. doi: 10.1007/s10569-012-9443-x.

[26] M. D. Canon, C. D. Cullum Jr, and E. Polak. Theory of optimal control and mathematical programming. 1970.

[27] C. Canuto, M. Y. Hussaini, A. Quarteroni, A. Thomas Jr, et al. *Spectral methods in fluid dynamics*. Springer Science & Business Media, 2012.

[28] L. Casalino and G. Colasurdo. Optimization of variable-specific-impulse interplanetary trajectories. *Journal of guidance, control, and dynamics*, 27(4):678–684, 2004.

[29] M. Cerf. Fast solution of minimum-time low-thrust transfer with eclipses. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 233(7):2699–2714, June 2019. ISSN 0954-4100, 2041-3025. doi: 10.1177/0954410018785971. URL `http://journals.sagepub.com/doi/10.1177/0954410018785971`.

[30] G. Colasurdo and L. Casalino. Tentative Solutions for Indirect Optimization of Spacecraft Trajectories. In G. Fasano and J. D. Pintér, editors, *Space Engineering*, volume 114, pages 87–102. Springer International Publishing, Cham, 2016. ISBN 978-3-319-41506-2 978-3-319-41508-6. doi: 10.1007/978-3-319-41508-6_3. URL `http://link.springer.com/10.1007/978-3-319-41508-6_3`. Series Title: Springer Optimization and Its Applications.

[31] B. A. Conway. A survey of methods available for the numerical optimization of continuous dynamic systems. *Journal of Optimization Theory and Applications*, 152(2): 271–306, 2012.

[32] H. Curtis. *Orbital Mechanics for Engineering Students: Revised Reprint*, chapter 10, pages 487–492. Butterworth-Heinemann, 2020.

[33] C. L. Darby, D. Garg, and A. V. Rao. Costate estimation using multiple-interval pseudospectral methods. *Journal of spacecraft and rockets*, 48(5):856–866, 2011.

[34] 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*, 32(4): 476–502, 2011.

[35] P. J. Davis and P. Rabinowitz. *Methods of numerical integration*. Courier Corporation, 2007.

[36] E. Dickmanns, K. H. Well, et al. Approximate solution of optimal control problems using third order hermite polynomial functions. In *Optimization Techniques IFIP Technical Conference*, pages 158–166. Springer, 1975.

[37] Q. T. Dinh and M. Diehl. Local convergence of sequential convex programming for nonconvex optimization. In *Recent Advances in Optimization and its Applications in Engineering*, pages 93–102. Springer, 2010.

[38] K. Du. On well-conditioned spectral collocation and spectral methods by the integral reformulation. *SIAM Journal on Scientific Computing*, 38(5):A3247–A3263, 2016.

[39] I. Dunning, J. Huchette, and M. Lubin. Jump: A modeling language for mathematical optimization. *SIAM review*, 59(2):295–320, 2017.

[40] T. N. Edelbaum. Optimum power-limited orbit transfer in strong gravity fields. *AIAA Journal*, 3(5):921–925, 1965.

[41] J. D. Eide, W. W. Hager, and A. V. Rao. Modified legendre–gauss–radau collocation method for optimal control problems with nonsmooth solutions. *Journal of Optimization Theory and Applications*, 191(2):600–633, 2021.

[42] A. Einstein. *The Collected Papers of Albert Einstein, Volume 15 (Translation Supplement): The Berlin Years: Writings & Correspondence, June 1925–May 1927*, volume 15. Princeton University Press, 1987.

[43] D. H. Ellison, B. A. Conway, J. A. Englander, and M. T. Ozimek. Analytic gradient computation for bounded-impulse trajectory models using two-sided shooting. *Journal of Guidance, Control, and Dynamics*, 41(7):1449–1462, 2018.

[44] D. H. Ellison, B. A. Conway, J. A. Englander, and M. T. Ozimek. Application and analysis of bounded-impulse trajectory models with analytic gradients. *Journal of Guidance, Control, and Dynamics*, 41(8):1700–1714, 2018.

[45] G. Elnagar, M. A. Kazemi, and M. Razzaghi. The pseudospectral legendre method for discretizing optimal control problems. *IEEE transactions on Automatic Control*, 40(10): 1793–1796, 1995.

[46] J. A. Englander, B. A. Conway, and T. Williams. Automated mission planning via evolutionary algorithms. *Journal of Guidance, Control, and Dynamics*, 35(6):1878–1887, 2012.

[47] F. Fahroo and I. Ross. Computational optimal control by spectral collocation with differential inclusion. In *NASA Conference Publication*, pages 185–200. NASA, 1999.

[48] F. Fahroo and I. Ross. Trajectory optimization by indirect spectral collocation methods. In *Astrodynamics specialist conference*, page 4028, 2000.

[49] F. Fahroo and I. M. Ross. Costate estimation by a legendre pseudospectral method. *Journal of Guidance, Control, and Dynamics*, 24(2):270–277, 2001.

[50] F. Fahroo and I. M. Ross. Direct trajectory optimization by a chebyshev pseudospectral method. *Journal of Guidance, Control, and Dynamics*, 25(1):160–166, 2002.

[51] F. Fahroo and I. M. Ross. Advances in pseudospectral methods for optimal control. In *AIAA guidance, navigation and control conference and exhibit*, page 7309, 2008.

[52] F. Fahroo and I. M. Ross. Convergence of the costates does not imply convergence of the control. *Journal of guidance, control, and dynamics*, 31(5):1492–1497, 2008.

[53] J. Fike and J. Alonso. The development of hyper-dual numbers for exact second-derivative calculations. In *49th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition*, page 886, 2011.

[54] B. Fornberg. *A practical guide to pseudospectral methods*. Number 1. Cambridge university press, 1998.

[55] A. Forsgren, P. E. Gill, and M. H. Wright. Interior methods for nonlinear optimization. *SIAM review*, 44(4):525–597, 2002.

[56] C. C. Françolin, H. Hou, W. W. Hager, and A. V. Rao. Costate estimation of state-inequality path constrained optimal control problems using collocation at legendre-gauss-radau points. In *52nd IEEE Conference on Decision and Control*, pages 6469–6474. IEEE, 2013.

[57] Y. Gao and C. Kluever. Analytic orbital averaging technique for computing tangential-thrust trajectories. *Journal of guidance, control, and dynamics*, 28(6):1320–1323, 2005.

[58] D. Garg, M. Patterson, C. Darby, C. Francolin, G. Huntington, W. Hager, and A. Rao. Direct trajectory optimization and costate estimation of general optimal control problems using a radau pseudospectral method. In *AIAA guidance, navigation, and control conference*, page 5989, 2009.

[59] 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*, 46(11):1843–1851, 2010.

[60] D. Garg, M. A. Patterson, C. Francolin, C. L. Darby, G. T. Huntington, W. W. Hager, and A. V. Rao. Direct trajectory optimization and costate estimation of finite-horizon and infinite-horizon optimal control problems using a radau pseudospectral method. *Computational Optimization and Applications*, 49(2):335–358, 2011.

[61] 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. 2017.

[62] I. M. Gelfand, R. A. Silverman, et al. *Calculus of variations*. Courier Corporation, 2000.

[63] J. Gergaud and T. Haberkorn. Orbital transfer: Some links between the low-thrust and the impulse cases. *Acta Astronautica*, 60(8-9):649–657, Apr. 2007. ISSN 00945765. doi: 10.1016/j.actaastro.2006.10.009. URL `https://linkinghub.elsevier.com/retrieve/pii/S0094576506003857`.

[64] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. User's guide for npsol (version 4.0): A fortran package for nonlinear programming. Technical report, Stanford Univ CA Systems Optimization Lab, 1986.

[65] P. E. Gill, W. Murray, and M. A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005.

[66] P. E. Gill, W. Murray, and M. H. Wright. *Practical optimization*. SIAM, 2019.

[67] Q. Gong, F. Fahroo, and I. M. Ross. Spectral algorithm for pseudospectral methods in optimal control. *Journal of Guidance, Control, and Dynamics*, 31(3):460–471, 2008.

[68] Q. Gong, I. M. Ross, W. Kang, and F. Fahroo. Connections between the covector mapping theorem and convergence of pseudospectral methods for optimal control. *Computational Optimization and Applications*, 41(3):307–335, 2008.

[69] Q. Gong, I. M. Ross, and F. Fahroo. Costate computation by a chebyshev pseudospectral method. *Journal of Guidance, Control, and Dynamics*, 33(2):623–628, 2010.

[70] K. F. Graham and A. V. Rao. Minimum-time trajectory optimization of low-thrust earth-orbit transfers with eclipsing. *Journal of Spacecraft and Rockets*, 53(2):289–303, 2016.

[71] A. Griewank. Who invented the reverse mode of differentiation. *Documenta Mathematica, Extra Volume ISMP*, pages 389–400, 2012.

[72] A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[73] T. Guo, F. Jiang, and J. Li. Homotopic approach and pseudospectral method applied jointly to low thrust trajectory optimization. *Acta Astronautica*, 71:38–50, 2012.

[74] T. Guo, F. Jiang, and J. Li. Homotopic approach and pseudospectral method applied jointly to low thrust trajectory optimization. *Acta Astronautica*, 71:38–50, Feb. 2012. ISSN 00945765. doi: 10.1016/j.actaastro.2011.08.008. URL `https://linkinghub.elsevier.com/retrieve/pii/S0094576511002694`.

[75] C. M. Haissig, K. D. Mease, and N. X. Vinh. Minimum-fuel, power-limited transfers between coplanar elliptical orbits. *Acta Astronautica*, 29(1):1–15, 1993.

[76] C. R. Hargraves and S. W. Paris. Direct trajectory optimization using nonlinear programming and collocation. *Journal of guidance, control, and dynamics*, 10(4):338–342, 1987.

[77] H. C. Henninger and J. D. Biggs. Near time-minimal earth to l 1 transfers for low-thrust spacecraft. *Journal of Guidance, Control, and Dynamics*, 40(11):2999–3004, 2017.

[78] D. R. Herber. Basic implementation of multiple-interval pseudospectral methods to solve optimal control problems. Technical report, 2015.

[79] S. Hernandez and M. R. Akella. Energy preserving low-thrust guidance for orbit transfers in ks variables. *Celestial Mechanics and Dynamical Astronomy*, 125(1):107–132, 2016.

[80] C. Hofmann and F. Topputo. Rapid low-thrust trajectory optimization in deep space based on convex programming. *Journal of Guidance, Control, and Dynamics*, 44(7): 1379–1388, 2021.

[81] B. V. Holden, S. He, and A. V. Rao. Minimum-time earth-to-mars interplanetary orbit transfer using adaptive gaussian quadrature collocation. *Journal of Spacecraft and Rockets*, 58(6):1819–1832, 2021.

[82] H. Holt, R. Armellin, N. Baresi, Y. Hashida, A. Turconi, A. Scorsoglio, and R. Furfaro. Optimal q-laws via reinforcement learning with guaranteed stability. *Acta Astronautica*, 187:511–528, 2021.

[83] S. Jain and P. Tsiotras. Trajectory optimization using multiresolution techniques. *Journal of Guidance, Control, and Dynamics*, 31(5):1424–1436, 2008.

[84] P. Jawaharlal Ayyanathan and E. Taheri. Mapped adjoint control transformation method for low-thrust trajectory design. *Acta Astronautica*, 193:418–431, Apr. 2022. ISSN

00945765. doi: 10.1016/j.actaastro.2021.12.019. URL `https://linkinghub.elsevier.com/retrieve/pii/S0094576521006548`.

[85] J.-W. Jo and J. E. Prussing. Procedure for applying second-order conditions in optimal control problems. *Journal of Guidance, Control, and Dynamics*, 23(2):241–250, 2000. doi: 10.2514/2.4546.

[86] J. L. Junkins and E. Taheri. Exploration of alternative state vector choices for low-thrust trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 42(1):47–64, 2019.

[87] S. Kameswaran and L. T. Biegler. Convergence rates for direct transcription of optimal control problems using collocation at radau points. *Computational Optimization and Applications*, 41(1):81–126, 2008.

[88] R. Kamyar and E. Taheri. Aircraft optimal terrain/threat-based trajectory planning and control. *Journal of Guidance, Control, and Dynamics*, 37(2):466–483, 2014.

[89] M. Karpenko, S. Bhatt, N. Bedrossian, A. Fleming, and I. Ross. First flight results on time-optimal spacecraft slews. *Journal of Guidance, Control, and Dynamics*, 35(2):367–376, 2012.

[90] M. Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4):849–904, 2017.

[91] D. E. Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2004.

[92] N. Koeppen, I. M. Ross, L. C. Wilcox, and R. J. Proulx. Fast mesh refinement in pseudospectral optimal control. *Journal of Guidance, Control, and Dynamics*, 42(4):711–722, 2019.

[93] G. Lantoine and R. P. Russell. A hybrid differential dynamic programming algorithm for constrained optimal control problems. part 1: Theory. *Journal of Optimization Theory and Applications*, 154(2):382–417, 2012.

[94] G. Lantoine, R. P. Russell, and T. Dargent. Using multicomplex variables for automatic computation of high-order derivatives. *ACM Transactions on Mathematical Software (TOMS)*, 38(3):1–21, 2012.

[95] S. Lee, P. von Ailmen, W. Fink, A. Petropoulos, and R. J. Terrile. Design and optimization of low-thrust orbit transfers. In *2005 IEEE Aerospace Conference*, pages 855–869. IEEE, 2005.

[96] D. Lev, R. M. Myers, K. M. Lemmer, J. Kolbeck, H. Koizumi, and K. Polzin. The technological and commercial expansion of electric propulsion. *Acta Astronautica*, 159: 213–227, 2019.

[97] S. Li and X. Jiang. Review and prospect of guidance and control for mars atmospheric entry. *Progress in Aerospace Sciences*, 69:40–57, 2014.

[98] T. Li, Z. Wang, and Y. Zhang. Double-homotopy technique for fuel optimization of power-limited interplanetary trajectories. *Astrophysics and Space Science*, 364(9):144, Sept. 2019. ISSN 0004-640X, 1572-946X. doi: 10.1007/s10509-019-3637-6. URL `http://link.springer.com/10.1007/s10509-019-3637-6`.

[99] Y. Liao, H. Li, and W. Bao. Indirect radau pseudospectral method for the receding horizon control problem. *Chinese Journal of Aeronautics*, 29(1):215–227, 2016.

[100] F. Liu, W. W. Hager, and A. V. Rao. An hp mesh refinement method for optimal control using discontinuity detection and mesh size reduction. In *53rd IEEE Conference on Decision and Control*, pages 5868–5873. IEEE, 2014.

[101] F. Liu, W. W. Hager, and A. V. Rao. Adaptive mesh refinement method for optimal control using decay rates of legendre polynomial coefficients. *IEEE Transactions on Control Systems Technology*, 26(4):1475–1483, 2017.

[102] K. Mall and M. J. Grant. Epsilon-Trig Regularization Method for Bang-Bang Optimal Control Problems. *Journal of Optimization Theory and Applications*, 174(2):500–517,

Aug. 2017. ISSN 0022-3239, 1573-2878. doi: 10.1007/s10957-017-1129-9. URL http://link.springer.com/10.1007/s10957-017-1129-9.

[103] K. Mall and E. Taheri. Entry trajectory optimization for mars science laboratory class missions using indirect uniform trigonometrization method. In *2020 American Control Conference (ACC)*, pages 4182–4187. IEEE, 2020.

[104] K. Mall and E. Taheri. Optimal control of wave energy converters using epsilon-trig regularization method. In *2020 American Control Conference (ACC)*, pages 1773–1778. IEEE, 2020.

[105] K. Mall and E. Taheri. Unified trigonometrization method for solving optimal control problems in atmospheric flight mechanics. In *AIAA Scitech 2020 Forum*, page 0022, 2020.

[106] K. Mall and E. Taheri. Three-degree-of-freedom hypersonic reentry trajectory optimization using an advanced indirect method. *Journal of Spacecraft and Rockets*, 59(5):1463–1474, 2022.

[107] K. Mall, M. J. Grant, and E. Taheri. Uniform trigonometrization method for optimal control problems with control and state constraints. *Journal of Spacecraft and Rockets*, 57(5):995–1007, 2020.

[108] K. Mall, M. J. Grant, and E. Taheri. Uniform Trigonometrization Method for Optimal Control Problems with Control and State Constraints. *Journal of Spacecraft and Rockets*, 57(5):995–1007, Sept. 2020. ISSN 0022-4650, 1533-6794. doi: 10.2514/1.A34624. URL https://arc.aiaa.org/doi/10.2514/1.A34624.

[109] K. Mall, E. Taheri, and P. Prabhu. Solving singular control problems using uniform trigonometrization method. *AIChE Journal*, 67(6):e17209, 2021.

[110] D. Malyuta, T. P. Reynolds, M. Szmuk, T. Lew, R. Bonalli, M. Pavone, and B. Acikmese. Convex optimization for trajectory generation. *arXiv preprint arXiv:2106.09125*, 2021.

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

[112] J. R. Martins and J. T. Hwang. Review and unification of methods for computing derivatives of multidisciplinary computational models. *AIAA journal*, 51(11):2582–2599, 2013.

[113] C. Mavriplis. Adaptive mesh strategies for the spectral element method. *Computer methods in applied mechanics and engineering*, 116(1-4):77–86, 1994.

[114] J. Meditch. On the problem of optimal thrust programming for a lunar soft landing. *IEEE Transactions on Automatic Control*, 9(4):477–484, 1964.

[115] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on optimization*, 2(4):575–601, 1992.

[116] A. T. Miller and A. V. Rao. Rapid ascent-entry vehicle mission optimization using hp-adaptive gaussian quadrature collocation. In *AIAA Atmospheric Flight Mechanics Conference*, page 0249, 2017.

[117] K. Mohan, M. Patterson, and A. Rao. Optimal trajectory and control generation for landing of multiple aircraft in the presence of obstacles. In *AIAA Guidance, Navigation, and Control Conference*, page 4826, 2012.

[118] B. J. Naasz, M. Strube, J. Van Eepoel, B. W. Barbee, and K. M. Getzandanner. Satellite servicing's autonomous rendezvous and docking testbed on the international space station. In *Annual AAS Rocky Mountain Section Guidance and Control Conference*, number LEGNEW-OLDGSFC-GSFC-LN-1040, 2011.

[119] R. D. Neidinger. Introduction to automatic differentiation and matlab object-oriented programming. *SIAM review*, 52(3):545–563, 2010.

[120] M. Neuenhofen. Review of theory and implementation of hyper-dual numbers for first and second order automatic differentiation. *arXiv preprint arXiv:1801.03614*, 2018.

[121] N. P. Nurre and E. Taheri. Comparison of indirect and convex-based methods for low-thrust minimum-fuel trajectory optimization. In *2022 AAS/AIAA Astrodynamics Specialist Conference, AAS 22-782, Charlotte, NC, USA, August 7-11*, 2022.

[122] S. Olver and A. Townsend. A fast and well-conditioned spectral method. *siam REVIEW*, 55(3):462–489, 2013.

[123] E. R. Pager and A. V. Rao. Method for solving bang-bang and singular optimal control problems using adaptive radau collocation. *Computational Optimization and Applications*, 81(3):857–887, 2022.

[124] B. Pan, P. Lu, X. Pan, and Y. Ma. Double-Homotopy Method for Solving Optimal Control Problems. *Journal of Guidance, Control, and Dynamics*, 39(8):1706–1720, Aug. 2016. ISSN 0731-5090, 1533-3884. doi: 10.2514/1.G001553. URL `https://arc.aiaa.org/doi/10.2514/1.G001553`.

[125] B. Pan, X. Pan, and S. Zhang. A new probability-one homotopy method for solving minimum-time low-thrust orbital transfer problems. *Astrophysics and Space Science*, 363(9):1–12, 2018.

[126] B. Pan, Y. Ma, and Y. Ni. A new fractional homotopy method for solving nonlinear optimal control problems. *Acta Astronautica*, 161:12–23, Aug. 2019. ISSN 00945765. doi: 10.1016/j.actaastro.2019.05.005. URL `https://linkinghub.elsevier.com/retrieve/pii/S0094576518317314`.

[127] B. Pan, X. Pan, and Y. Ma. A quadratic homotopy method for fuel-optimal low-thrust trajectory design. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 233(5):1741–1757, Apr. 2019. ISSN 0954-4100, 2041-3025. doi: 10.1177/0954410018761965. URL `http://journals.sagepub.com/doi/10.1177/0954410018761965`.

[128] X. Pan and B. Pan. Practical Homotopy Methods for Finding the Best Minimum-Fuel Transfer in the Circular Restricted Three-Body Problem. *IEEE Access*, 8:47845–47862, 2020. ISSN 2169-3536. doi: 10.1109/ACCESS.2020.2978246. URL `https://ieeexplore.ieee.org/document/9023950/`.

[129] M. A. Patterson and A. V. Rao. Exploiting sparsity in direct collocation pseudospectral methods for solving optimal control problems. *Journal of Spacecraft and Rockets*, 49 (2):354–377, 2012.

[130] 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 (TOMS)*, 41(1):1–37, 2014.

[131] M. A. Patterson, W. W. Hager, and A. V. Rao. A ph mesh refinement method for optimal control. *Optimal Control Applications and Methods*, 36(4):398–421, 2015.

[132] E. Pellegrini and R. P. Russell. On the computation and accuracy of trajectory state transition matrices. *Journal of Guidance, Control, and Dynamics*, 39(11):2485–2499, 2016.

[133] J. Peng, C. Roos, and T. Terlaky. Primal-dual interior-point methods for second-order conic optimization based on self-regular proximities. *SIAM Journal on Optimization*, 13 (1):179–203, 2002.

[134] J. T. Peterson, S. K. Singh, J. L. Junkins, and E. Taheri. Lyapunov guidance in orbit element space for low-thrust cislunar trajectories. *Advances in the Astronautical Sciences AAS Guidance, Navigation, and Control 2020*, 172, AAS 20-115, 2020.

[135] A. E. Petropoulos, Z. B. Tarzi, G. Lantoine, T. Dargent, and R. Epenoy. Techniques for designing many-revolution electric-propulsion trajectories. *Advances in the Astronautical Sciences*, 152(3):2367–2386, 2014.

[136] V. Petukhov and W. S. Wook. Joint Optimization of the Trajectory and the Main Parameters of an Electric Propulsion System. *Procedia Engineering*, 185:312–318, 2017. ISSN 18777058. doi: 10.1016/j.proeng.2017.03.309. URL https://linkinghub.elsevier.com/retrieve/pii/S1877705817314637.

[137] V. Petukhov and S. W. Yoon. Optimization of perturbed spacecraft trajectories using complex dual numbers. part 1: Theory and method. *Cosmic Research*, 59(5):401–413, 2021.

[138] V. Petukhov and S. W. Yoon. Optimization of perturbed spacecraft trajectories using complex dual numbers. part 2: Numerical results. *Cosmic Research*, 59(6):517–528, 2021.

[139] V. G. Petukhov. Method of continuation for optimization of interplanetary low-thrust trajectories. *Cosmic Research*, 50(3):249–261, May 2012. ISSN 0010-9525, 1608-3075. doi: 10.1134/S0010952512030069. URL http://link.springer.com/10.1134/S0010952512030069.

[140] L. S. Pontryagin. *Mathematical theory of optimal processes*. CRC press, 1987.

[141] 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 (TOMS)*, 37(2):1–39, 2010.

[142] E. A. Robertson. Synopsis of precision landing and hazard avoidance (PL&HA) capabilities for space exploration. In *AIAA Guidance, Navigation, and Control Conference*, page 1897, 2017.

[143] R. T. Rockafellar. *Convex analysis*, volume 18. Princeton university press, 1970.

[144] I. Ross and R. Proulx. Further results on fast birkhoff pseudospectral optimal control programming. *Journal of Guidance, Control, and Dynamics*, 42(9):2086–2092, 2019.

[145] I. M. Ross. *A primer on Pontryagin's principle in optimal control*. Collegiate publishers, 2 edition, 2015.

[146] I. M. Ross. Enhancements to the dido optimal control toolbox. *arXiv preprint arXiv:2004.13112*, 2020.

[147] I. M. Ross and F. Fahroo. Legendre pseudospectral approximations of optimal control problems. In *New trends in nonlinear dynamics and control and their applications*, pages 327–342. Springer, 2003.

[148] I. M. Ross and F. Fahroo. Pseudospectral knotting methods for solving nonsmooth optimal control problems. *Journal of Guidance, Control, and Dynamics*, 27(3):397–405, 2004.

[149] I. M. Ross and M. Karpenko. A review of pseudospectral optimal control: From theory to flight. *Annual Reviews in Control*, 36(2):182–197, 2012.

[150] I. M. Ross, Q. Gong, and P. Sekhavat. Low-thrust, high-accuracy trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 30(4):921–933, 2007.

[151] I. M. Ross, Q. Gong, M. Karpenko, and R. Proulx. Scaling and balancing for high-performance computation of optimal controls. *Journal of Guidance, Control, and Dynamics*, 41(10):2086–2097, 2018.

[152] J. L. Russell. Kepler's laws of planetary motion: 1609–1666. *The British journal for the history of science*, 2(1):1–24, 1964.

[153] R. P. Russell. Primer vector theory applied to global low-thrust trade studies. *Journal of Guidance, Control, and Dynamics*, 30(2):460–472, 2007.

[154] M. Sagliano. Performance analysis of linear and nonlinear techniques for automatic scaling of discretized control problems. *Operations Research Letters*, 42(3):213–216, 2014.

[155] M. Sagliano. Pseudospectral convex optimization for powered descent and landing. *Journal of guidance, control, and dynamics*, 41(2):320–334, 2018.

[156] M. Sagliano, S. Theil, M. Bergsma, V. D'Onofrio, L. Whittle, and G. Viavattene. On the radau pseudospectral method: theoretical and implementation advances. *CEAS Space Journal*, 9(3):313–331, 2017.

[157] H. Saranathan and M. J. Grant. Relaxed autonomously switched hybrid system approach to indirect multiphase aerospace trajectory optimization. *Journal of Spacecraft and Rockets*, 55(3):611–621, 2018.

[158] I. J. Schoenberg. On hermite-birkhoff interpolation. *Journal of Mathematical Analysis and Applications*, 16(3):538–543, 1966.

[159] J. Shen, T. Tang, and L.-L. Wang. *Spectral methods: algorithms, analysis and applications*, volume 41. Springer Science & Business Media, 2mod011.

[160] A. Shirazi, J. Ceberio, and J. A. Lozano. Spacecraft trajectory optimization: A review of models, objectives, approaches and solutions. *Progress in Aerospace Sciences*, 102: 76–98, 2018.

[161] S. Singh, J. Junkins, B. Anderson, and E. Taheri. Eclipse-Conscious Transfer to Lunar Gateway Using Ephemeris-Driven Terminal Coast Arcs. *Journal of Guidance, Control, and Dynamics*, 44(11):1972–1988, Nov. 2021. ISSN 1533-3884. doi: 10.2514/1.G005920. URL `https://arc.aiaa.org/doi/10.2514/1.G005920`.

[162] S. K. Singh, B. D. Anderson, E. Taheri, and J. L. Junkins. Exploiting manifolds of L 1 halo orbits for end-to-end Earth–Moon low-thrust trajectory design. *Acta Astronautica*, 183:255–272, June 2021. ISSN 00945765. doi: 10.1016/j.actaastro.2021.03.017. URL `https://linkinghub.elsevier.com/retrieve/pii/S0094576521001302`.

[163] S. Sowell and E. Taheri. Minimum-time and minimum-fuel low-thrust trajectory design for satellite formation in low-earth orbits. In *2022 American Control Conference (ACC)*, pages 2938–2943. IEEE, 2022.

[164] H. J. Sussmann and J. C. Willems. 300 years of optimal control: from the brachystochrone to the maximum principle. *IEEE Control Systems Magazine*, 17(3):32–44, 1997.

[165] E. Taheri. Composite Smooth Control Method for Low-Thrust Trajectory Design: Variable Specific Impulse Engine. In *AIAA Scitech 2020 Forum*, Orlando, FL, Jan. 2020. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-595-1. doi: 10.2514/6.2020-2184. URL `https://arc.aiaa.org/doi/10.2514/6.2020-2184`.

[166] E. Taheri. Optimization of many-revolution minimum-time low-thrust trajectories using sundman transformation. In *AIAA Scitech 2021 Forum*, page 1343, 2021.

[167] E. Taheri. Optimization of Many-Revolution Minimum-Time Low-Thrust Trajectories Using Sundman Transformation. In *AIAA Scitech 2021 Forum*, VIRTUAL EVENT, Jan. 2021. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-609-5. doi: 10.2514/6.2021-1343. URL `https://arc.aiaa.org/doi/10.2514/6.2021-1343`.

[168] E. Taheri and J. L. Junkins. Generic smoothing for optimal bang-off-bang spacecraft maneuvers. *Journal of Guidance, Control, and Dynamics*, 41(11):2470–2475, 2018.

[169] E. Taheri and J. L. Junkins. Generic Smoothing for Optimal Bang-Off-Bang Spacecraft Maneuvers. *Journal of Guidance, Control, and Dynamics*, 41(11):2470–2475, Nov. 2018. ISSN 0731-5090, 1533-3884. doi: 10.2514/1.G003604. URL `https://arc.aiaa.org/doi/10.2514/1.G003604`.

[170] E. Taheri and J. L. Junkins. How Many Impulses Redux. *The Journal of the Astronautical Sciences*, 67(2):257–334, June 2020. ISSN 0021-9142, 2195-0571. doi:

10.1007/s40295-019-00203-1. URL http://link.springer.com/10.1007/s40295-019-00203-1.

[171] E. Taheri, I. Kolmanovsky, and E. Atkins. Enhanced smoothing technique for indirect optimization of minimum-fuel low-thrust trajectories. *Journal of Guidance, Control, and Dynamics*, 39(11):2500–2511, 2016.

[172] E. Taheri, I. Kolmanovsky, and E. Atkins. Enhanced Smoothing Technique for Indirect Optimization of Minimum-Fuel Low-Thrust Trajectories. *Journal of Guidance, Control, and Dynamics*, 39(11):2500–2511, Nov. 2016. ISSN 0731-5090, 1533-3884. doi: 10.2514/1.G000379. URL https://arc.aiaa.org/doi/10.2514/1.G000379.

[173] E. Taheri, N. I. Li, and I. Kolmanovsky. Co-state initialization for the minimum-time low-thrust trajectory optimization. *Advances in Space Research*, 59(9):2360–2373, 2017.

[174] E. Taheri, N. I. Li, and I. Kolmanovsky. Co-state initialization for the minimum-time low-thrust trajectory optimization. *Advances in Space Research*, 59(9):2360–2373, May 2017. ISSN 02731177. doi: 10.1016/j.asr.2017.02.010. URL https://linkinghub.elsevier.com/retrieve/pii/S0273117717301072.

[175] E. Taheri, J. L. Junkins, I. Kolmanovsky, and A. Girard. A novel approach for optimal trajectory design with multiple operation modes of propulsion system, part 1. *Acta Astronautica*, 172:151–165, 2020.

[176] E. Taheri, J. L. Junkins, I. Kolmanovsky, and A. Girard. A novel approach for optimal trajectory design with multiple operation modes of propulsion system, part 2. *Acta Astronautica*, 172:166–179, 2020.

[177] E. Taheri, J. L. Junkins, I. Kolmanovsky, and A. Girard. A novel approach for optimal trajectory design with multiple operation modes of propulsion system, part 1. *Acta Astronautica*, 172:151–165, July 2020. ISSN 00945765. doi: 10.1016/j.actaastro.

2020.02.042. URL https://linkinghub.elsevier.com/retrieve/pii/S0094576520301156.

[178] L. N. Trefethen. Finite difference and spectral methods for ordinary and partial differential equations. 1996.

[179] L. N. Trefethen. Is gauss quadrature better than clenshaw–curtis? *SIAM review*, 50(1): 67–87, 2008.

[180] E. Trélat. Optimal control and applications to aerospace: some results and challenges. *Journal of Optimization Theory and Applications*, 154(3):713–758, 2012.

[181] E. Verlinde. On the origin of gravity and the laws of newton. *Journal of High Energy Physics*, 2011(4):1–27, 2011.

[182] G. von Winckel. Legende-gauss nodes and weights, matlab central file exchange. https://www.mathworks.com/matlabcentral/fileexchange/4540-legendre-gauss-quadrature-weights-and-nodes, 2022.

[183] G. von Winckel. Legende-gauss-lobatto nodes and weights, matlab central file exchange. https://www.mathworks.com/matlabcentral/fileexchange/4775-legende-gauss-lobatto-nodes-and-weights, 2022.

[184] G. von Winckel. Legende-gauss-radau nodes and weights, matlab central file exchange. https://www.mathworks.com/matlabcentral/fileexchange/4850-legende-gauss-radau-nodes-and-weights, 2022.

[185] G. von Winckel. Pseudospectral differentiation on an arbitrary grid, July 2004. URL https://www.mathworks.com/matlabcentral/fileexchange/5515-pseudospectral-differentiation-on-an-arbitrary-grid.

[186] H. Wang and S. Xiang. On the convergence rates of legendre approximation. *Mathematics of computation*, 81(278):861–877, 2012.

[187] L.-L. Wang, M. D. Samson, and X. Zhao. A well-conditioned collocation method using a pseudospectral integration matrix. *SIAM Journal on Scientific Computing*, 36(3):A907–A929, 2014.

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

[189] Z. Wang and M. J. Grant. Optimization of minimum-time low-thrust transfers using convex programming. *Journal of Spacecraft and Rockets*, 55(3):586–598, 2018.

[190] M. J. Weinstein and A. V. Rao. Algorithm 984: Adigator, a toolbox for the algorithmic differentiation of mathematical functions in matlab using source transformation via operator overloading. *ACM Transactions on Mathematical Software (TOMS)*, 44(2):21, 2017.

[191] G. J. Whiffen and J. A. Sims. Application of a novel optimal control algorithm to low-thrust trajectory optimization. 2001.

[192] D. C. Woffinden and D. K. Geller. Navigating the road to autonomous orbital rendezvous. *Journal of Spacecraft and Rockets*, 44(4):898–909, 2007.

[193] R. Woollands, E. Taheri, and J. L. Junkins. Efficient computation of optimal low thrust gravity perturbed orbit transfers. *The Journal of the Astronautical Sciences*, 67(2):458–484, 2020.

[194] M. Wright. The interior-point revolution in optimization: history, recent developments, and lasting consequences. *Bulletin of the American mathematical society*, 42(1):39–56, 2005.

[195] S. Wright, J. Nocedal, et al. Numerical optimization. *Springer Science*, 35(67-68):7, 1999.

[196] Z. Xie, C. K. Liu, and K. Hauser. Differential dynamic programming with nonlinear constraints. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 695–702. IEEE, 2017.

[197] H. Yan, Q. Gong, C. D. Park, I. M. Ross, and C. N. D'Souza. High-accuracy trajectory optimization for a trans-earth lunar mission. *Journal of guidance, control, and dynamics*, 34(4):1219–1227, 2011.

[198] L. C. Young. *Lectures on the calculus of variations and optimal control theory*, volume 304. American Mathematical Soc., 2000.

[199] C. Zhang, F. Topputo, F. Bernelli-Zazzera, and Y.-S. Zhao. Low-Thrust Minimum-Fuel Optimization in the Circular Restricted Three-Body Problem. *Journal of Guidance, Control, and Dynamics*, 38(8):1501–1510, Aug. 2015. ISSN 0731-5090, 1533-3884. doi: 10.2514/1.G001080. URL `https://arc.aiaa.org/doi/10.2514/1.G001080`.

[200] P. Zhang, J. Li, H. Baoyin, and G. Tang. A low-thrust transfer between the Earth–Moon and Sun–Earth systems based on invariant manifolds. *Acta Astronautica*, 91:77–88, Oct. 2013. ISSN 00945765. doi: 10.1016/j.actaastro.2013.05.005. URL `https://linkinghub.elsevier.com/retrieve/pii/S0094576513001550`.

[201] Z. Zhang, D. Zhao, X. Li, C. Kong, and M. Su. Convex optimization for rendezvous and proximity operation via birkhoff pseudospectral method. *Aerospace*, 9(9):505, 2022.

[202] Z. Zhu, Q. Gan, X. Yang, and Y. Gao. Solving fuel-optimal low-thrust orbital transfers with bang-bang control using a novel continuation technique. *Acta Astronautica*, 137:98–113, Aug. 2017. ISSN 00945765. doi: 10.1016/j.actaastro.2017.03.032. URL `https://linkinghub.elsevier.com/retrieve/pii/S0094576516306889`.

Appendices

Appendix A

How to Use TOPS

A.1 Example Problem Setup

This appendix section aims to provide a "tutorial" of sorts on how to solve an optimal control problem using the Tiger Optimization Software. In order to keep this section as simple as possible, we will provide pseudoscode functions that demonstrate how to set up the Orbit-Raising problem presented in Section 6.2. When you first setup TOPS, you must run the file "INSTALL_TOPS.m". This file will download the required files to perform Lagrange interpolation, add TOPS to the MATLAB path, and look for installations of SNOPT, IPOPT, and ADiGator. To use AD, you need to download a copy of ADiGator from Matthew Weinstein's Github page [190]. A simple Google search for "Matthew Weinstein ADiGator" will produce the ADiGator Github link from which the user can download the required source code. Download the .zip file and unzip the contents to the location 'TOPS/Automatic Differentiation/', and the AD capabilities of TOPS will be good to go. To set up SNOPT and IPOPT, place the source folders for these solvers (e.g., 'snopt7' or 'ipoptmatlab') in the folder 'TOPS/NLP Solvers/'. For a free MATLAB copy of IPOPT, simply Google search "IPOPT MATLAB Bertolazzi" and a link to the Github page with instructions to install the mexed IPOPT solver should be one of the first results. Note that this version of IPOPT is also available as a MATLAB toolbox, so if the user wants to install the toolbox rather than downloading the source code, they may so. TOPS should be able to use it just fine. Remember to run "INSTALL_TOPS.m" when starting MATLAB or after installing ADiGator, SNOPT, or IPOPT.

After this, navigate to the "TOPS/problems" folder and create a new folder for your problem. The folder for this problem is already included under the problems folder as "orbit-raising." We create a main file called "OR_main.m" that will define the problem and call TOPS to solve it. An example main file is provided below.

```
% ==================================
% Orbit Raising Problem Main File
% OR_main.m
% ==================================
clear;
clc;

% Store user function handles in "p.func" for all the functions you need.
p.func.dynamics   = @OR_dynamics; % A function that calculates "f".
p.func.mayer      = @OR_mayer;    % Mayer cost function.
```

```
% p.func.lagrange  = @OR_lagrange; % Lagrange cost function
p.func.event     = @OR_event;    % Event defects function.
p.func.initial   = @OR_initial; % Initial guess function.
% p.func.path_eq   = @OR_path_eq; % Additional path equality function.
% p.func.path_ineq = @OR_path_ineq; % Path inequality function.

% Tell the solver how many states and controls we have.
p.ns = 4; % x = [r, theta, u, v]
p.nu = 1; % u = phi

% Also, we tell it that this is NOT a free final time problem.
p.varTF = 0; % 0 = no, 1 = yes.

% Next, we store our problem specific constants in 'p.prob'.
p.prob.m0   = 1;      % The initial mass.
p.prob.mDot = 0.0749; % The mass flow rate.
p.prob.T    = 0.1405; % The maximum thrust.
p.prob.mu   = 1;      % Scaled gravitational parameter

% After this, we can set our initial and final states.
p.prob.S0 = [1, 0, 0, 1];
p.prob.SF = 0; % Here, r(tf) is free, theta(tf) is free, u(tf)
               % should be 0 (circle-to-circle), and v(tf) is
               % constrained to equal sqrt(mu/r(tf)). The
               % only one to put here is uf.

% Define problem time horizon. Always include these arguments,
% even for a free-final-time problem, and use tf as a guess.
p.prob.t0 = 0;
p.prob.tf = 3.32;

% === Variable Bounds === %
% Here, we construct some basic state bounds. Make sure these are stored
% in 'p.XU', 'p.XL', etc.
p.XU = [4, 4, 2, 2]; % [r, theta, u, v]
p.XL = [0, 0, 0, 0];

% If the variable is unbounded, just set the corresponding bound to
% plus or minus inf. For example,
% p.XU = [inf, inf, inf inf];
% p.XL = -[inf, inf, inf, inf];

% Next, we set our control bounds. We include these to
% restrict the search space and improve convergence. Always
% store these as shown below.
p.UU = 2*pi;  % Phi upper bound.
p.UL = -2*pi; % Phi lower bound.

% If we have time bounds, store them in these structures. We don't need
% them since this is a fixed final time problem.
% p.tU = 2*p.prob.tf;
% p.tL = 0;

% === Mesh Setup === %
% Next, we should define our initial mesh. If defined, p.Narray and
% p.Tarray will overwrite the numSegInit option.
p.Narray = [20,20];   % Number of points in each segment.
```

```
p.Tarray = [0,0.5,1]; % Relative locations of mesh points (boundaries).

% Next, we set up some mesh options. If you include these, you don't
% have to define p.Narray or p.Tarray.
p.mesh.eps = 1e-4; % Here, we ask for a mesh accuracy of 10^-4;
p.mesh.Nmin = 10;  % Here, we set our Nmin for the ph scheme.
p.mesh.Nmax = 20;  % Set Nmax for the ph scheme.

% This option let us set the initial number of mesh segments.
% We don't need it, since we already set up our mesh in p.Narray.
p.mesh.numSegInit = 2;

% Since we aren't using mesh refinement, we don't need them.
% Next, we set the maximum number of mesh refinement iterations.
p.mesh.Mmax = 1; % Just 1 for no mesh refinement.

% Finally, set the mesh algorithm.
p.mesh.algorithm = 'ph'; % Ph is your best bet. You can choose
                         % 'hp-legendre' or 'hp-u', but they may not work.

% === Solver Setup === %
% First, select our solver.
p.opts.solver = 'fmincon';

% Next, we set some options.
p.opts.max_iter   = 20000; % Max solver iterations.
p.opts.constr_tol = 1e-6;  % Set our solver constraint tolerance.
p.opts.optim_tol  = 1e-6;  % Set our solver optimality tolerance.

% === Scaling Options === %
% This option lets us choose to use the automatic affine scaling
% method introduced earlier. We don't need it now.
p.opts.scale = 0; % 0 = no, 1 = yes.

% === PS Transcription settings === %
% Let's select our transcription. We can pick between LGR, LGL, and CGL.
p.opts.transcription = 'LGR';

% === Derivative Options === %
% These options let us choose to use AD derivatives or not.
% We can pick 0, 1, or 2 for the corresponding derivative level.
p.opts.derivative_level = 1; % Just use AD first derivatives.

% Next, select our derivative type. We can pick between AD or complex step.
% 'ad-vect' is the fastest and most efficient for most problems.
p.opts.derivative_type = 'ad-vect'; % 'ad', 'ad-vect', or 'cs' (first
                                    % derivatives only)

% === Solve the Problem === %
% Finally,  we solve the problem by calling TOPS with p as the argument.
[t,X,U,f,p,L] = TOPS(p);

% For output, we get our time vector t, the states X, the controls U,
% the objective value f, and the costates L and an output structure p.

% === Plot Solution === %
U = wrapTo2Pi(U); % wrap control
```

```

% Plot state solution.
figure()
hold on;
plot(t, X, '-o', 'LineWidth', 2, 'MarkerSize', 4);
xlabel('Time [TU]')
ylabel('State Solution')
legend('r', '\theta', 'u', 'v', 'location', 'northwest')
xlim([p.prob.t0 p.prob.tf])

% Plot control solution.
figure()
hold on;
plot(t, U, '-o', 'LineWidth', 2, 'MarkerSize', 4);
xlabel('Time [TU]')
ylabel('Control Solution')
xlim([p.prob.t0 p.prob.tf])

% Plot costate solution.
figure()
hold on;
plot(t, L, '-o', 'LineWidth', 2, 'MarkerSize', 4);
xlabel('Time [TU]')
ylabel('Costate Solution')
legend('L_r', 'L_\theta', 'L_u', 'L_v', 'location', 'northwest')
xlim([p.prob.t0 p.prob.tf])

% And we're done!
```

And that is it for the main file! Since we do not have any non-dynamic path equalities, inequalities, or a Lagrange cost for this problem, we do not need to define those functions. However, we will still include them to show how one should write these functions. Let us first look at the dynamics function.

```
% ==================================
% Orbit raising problem dynamics file
% OR_dynamics.m
% ==================================
% Inputs:
% X: The state matrix.
% U: The control matrix.
% t: A time vector.
% p: Problem data structure.
% Outputs:
% f: The state derivative.
% ==================================
function f = OR_dynamics(X,U,t,p)
% First, we extract any information that we need for our dynamics file
% from 'p'.
T    = p.prob.T;
m0   = p.prob.m0;
mDot = p.prob.mDot;

% Next, we extract our states and controls from X and U.
r     = X(:,1);
theta = X(:,2);
```

```
u      = X(:,3);
v      = X(:,4);

phi = U(:,1); % Only control we have.

% Next, we do an intermediate calculation to get A.
A = T./(m0 - mDot.*t); % Time-varying thrust.

% Finally, we calculate our state derivatives.
rDot      = u;     % Radial velocity.
thetaDot = v./r; % Angular velocity.
uDot = (v.^2)./r - 1./(r.^2) + A.*sin(phi); % Radial acceleration.
vDot = -(u.*v)./r + A.*cos(phi); % Angular acceleration.

% Last, we concatenate the column-vector derivatives row-wise.
f = [rDot, thetaDot, uDot, vDot];
end
```

It is important to note that the entire dynamics function should be vectorized, which means that it should be able to operate on inputs whose row dimension is an arbitrary positive number. Practically, this means that matrix operations are prohibited and element-wise multiplication and division should be used. Additionally, the output matrix `f` should be preallocated at the start of the file or created at the end, as is shown in the above dynamics file. Now that we have our dynamics function, we can take a look at our Mayer or endpoint cost function.

```
% =====================================
% Orbit Raising Problem Mayer cost.
% OR_mayer.m
% =====================================
% Inputs:
% x0: The initial state (1 by Nx).
% xf: The final state (1 by Nx).
% tf: The final time.
% p: Problem data.
% Outputs:
% E: The Mayer cost.
% =====================================
function E = OR_mayer(x0,xf,tf,p)
% This function is pretty straightforward. Our cost is just the
% negative final radius.
E = -xf(1); % phi = -r(t_f)
end
```

Well that was easy! Let's look at the Lagrange cost function, even though we don't have a Lagrange cost for this problem.

```
% =====================================
% Orbit Raising Problem Lagrange cost.
% OR_Lagrange.m
% =====================================
% Inputs:
% X: The state matrix.
% U: The control matrix.
% t: The time vector.
% p: Problem data.
```

```
% Outputs:
% F: Lagrangian of the cost.
% =====================================
function F = OR_lagrange(X,U,t,p)
% EXAMPLE ONLY. Do not include this in the actual problem.

% If we did have a Lagrange cost, we would just place the Lagrangian
% in this function. It might look like this.
F = (T./m).*U;
end
```

The Lagrange cost function is very simple too. This function should also be vectorized. Let's look at the events function next.

```
% =====================================
% Orbit Raising Problem event constraints.
% OR_event.m
% =====================================
% Inputs:
% x0: The initial state (1 by Nx).
% xf: The final state (1 by Nx).
% tf: The final time.
% p: Problem data.
% Outputs:
% e: The event constraints.
% =====================================
function e = OR_event(x0,xf,t,p)
% We stored our known BCs earlier in p.prob.SF. This function
% should calculate the event constraints and output them as a vector.
e1 = x0(1:4) - p.prob.S0;   % Here, we enforce initial conditions.
e2 = xf(3);  % u(tf) = 0.
e3 = xf(4) - sqrt(1/xf(1)); % v(tf) - sqrt(mu/r(tf)) = 0.

% Finally, we stack them as an output.
e = [e1; e2; e3];
end
```

Next, we can take a look at the initial guess function. This function constructs an initial guess for z.

```
% =====================================
% Orbit raising problem initial guess constructor.
% OR_initial.m
% =====================================
% Inputs:
% tx: Time points associated with the states (LGR and discretization ...
    point).
% tu: Time points associated with the controls (only the LGR points).
% p: Problem data.
% Outputs:
% z0: Initial guess.
% =====================================
function z0 = OR_initial(tx,tu,p)
% First, we make a function that will linearly interpolate between
% our initial and final conditions at the points given in tx
% and tu.
```

```
interp_fun = @(t,z0,zf) (zf - z0)/(t(end) - t(1)).*t + z0;

% Next, we intialize our state and control guesses.
S0 = []; U0 = []; % initialize

% Make an initial guess for our final states. We assume a final radius
% of 3 AU and a final theta of pi.
SFg = [3; pi; 0; sqrt(p.prob.mu/3)]; % rf = 3, thetaf = pi as our guess.

% Next, we iterate over the states to get our guess for each state/control.
% The 'tx' variable are the LGR support points plus the non-collocated
% point at the final time. Your initial guess should provide values for the
% states at these points ONLY, so 'tx' is provided for your convenience.
for i = 1:p.ns
    S0 = [S0; interp_fun(tx, p.prob.S0(i), SFg(i))];
end

% Next, we interpolate between -pi and pi for our intial control guess.
% The parameter 'tu' is similar to 'tx' but it omits the non-discretized
% point. Your control guess should be obtained at all points in 'tu', so
% it is provided for your convenience.
U0 = [U0; interp_fun(tu, -pi, pi)];

% Combine the initial guess such that z0 = [X0(:); U0(:); tf0]. TOPS won't
% solve an initial final time problem right now because they are rare. It
% may solve them in the future.
z0 = [S0(:); U0(:)];
end
```

Again, this process is fairly simple. This just requires a bit of typing. We have finished everything we need for the orbit-raising problem. However, we will look at what the path equality and inequality functions should look like.

```
% ======================================
% Example path equality function.
% OR_path_eq.m
% ======================================
% Inputs:
% X: The state matrix.
% U: The control matrix.
% t: The time vector.
% p: Problem data.
% Outputs:
% e: A matrix of path equalities.
% ======================================
function e = OR_path_eq(X,U,t,p)
% EXAMPLE ONLY. Do not include this in the actual problem.

% This function should be used to define path equalities that are
% not part of the dynamics. Take a three-dimensional low thrust problem.
% We want to constrain a thrust aiming vector to have a magnitude of 1,
% given by norm(u_aim) = 1, where u_aim = [ur, ut, uh]. We can
% re-write this as u_aim^2 = 1. This is constructed as follows.
e = U(:,1).^2 + U(:,2).^2 + U(:,3).^2 - 1;
end
```

Note that the path equality function should also be vectorized. All equality constraints should be written in a defect form, such that the defect is equal to zero. Finally, we will look at the path inequality function. This is almost identical to the path equality function.

```
% ======================================
% Example path inequality function.
% OR_path_ineq.m
% ======================================
% Inputs:
% X: The state matrix.
% U: The control matrix.
% t: The time vector.
% p: Problem data.
% Outputs:
% h: A matrix of path inequalities.
% ======================================
function h = OR_path_ineq(X,U,t,p)
% EXAMPLE ONLY. Do not include in actual problem.

% This function should be used to define path inequalities. This is
% fairly straightforward. They should look like this.
h1 = a - b; % a ≤ b
h2 = c - d; % d ≥ c

% Combine them as follows.
h = [h1, h2];
end
```

This concludes the tutorial on how to set up a problem in TOPS! The solutions for the orbit raising problem that were found using this code can be seen in Section 6.2. To see this code in the TOPS solver, look under ‘TOPS/problems/orbit-raising’.

A.2 TOPS Options

This section contains a list of default TOPS options.

Solver Settings

Setting	Default Value	Options	Description
p.opts.solver	'fmincon'	'fmincon', ... 'snopt', ... 'ipopt'	NLP solver choice. A string or char vector. Only use SNOPT or IPOPT if on MATLAB path.
p.opts.max_iter	10000	[-]	Maximum number of solver/major iterations.
p.opts.constr_tol	1e-6	[-]	Constraint violation tolerance.
p.opts.optim_tol	1e-6	[-]	Optimality tolerance.
p.opts.fmincon	[]	All fmincon options.	Store non-default fmincon options. Will iteration and tolerance options unless overwritten.
p.opts.snopt	[]	All SNOPT options.	User SNOPT options.
p.opts.ipopt	[]	All IPOPT options.	User IPOPT options.

Derivative Settings

Setting	Default Value	Options	Description
p.opts.derivative_level	[-]	0, 1, 2	Derivative level. Will throw error if this field is not set.
p.opts.derivative_type	'ad-vect'	'ad', 'ad-vect', 'cs'.	Derivative method. If 'cs', will set derivative level to 1.
p.opts.check_derivatives	0	0, 1	NLP check on AD derivatives.

Scaling Settings

Setting	Default Value	Options	Description
p.opts.scale	0	0, 1	Use automatic scaling.
p.opts.scale_type	'affine'	'affine', 'pjrn'.	Scaling method to use. Capable of using the PJRN method, but it is not advised.

Mesh Options

Setting	Default Value	Options	Description
p.Narray	`[]`	Any array of positive integers.	Number of collocation points in each mesh interval. Should have one less element that p.Tarray. Automatically calculated if empty.
p.Tarray	`[]`	Any strictly increasing sequence. Ex: `[0, 0.5, 1]` is a two interval mesh with the knot at half the final time.	Locations of the mesh points. Should include the initial and final mesh points. Automatically calculated if empty.
p.mesh.algorithm	`'ph'`	`'ph'` `'hp-legendre'`, `'hph'`	Mesh refinement algorithm.
p.mesh.Mmax	`1`	Any integer $\geq$ `1`.	Number of mesh iterations.
p.mesh.Nmin	`5`	Any integer $\geq$ `3`.	Minimum collocation points in a segment.
p.mesh.Nmax	`15`	Any integer $\leq$ `Inf`.	Max collocation points in a segment. `Inf` results in a pure *p* method and Nmax = Nmin results in a true *h* method.
p.mesh.numSegInit	`1`	Any integer $\geq$ `1`.	Initial number of evenly spaced mesh segments.
p.mesh.eps	`1e-4`	Problem-dependent value. Should generally be $\leq$ `1e-3`.	Mesh accuracy tolerance.
p.mesh.deltafobjtol	`1e-4`	[-]	If change in objective between mesh iterations is less than this, algorithm terminates.
p.mesh.uDotMax	`1`	In [0,1].	For *hph* algorithm.
p.mesh.sigma_bar	`1`	In [0,1].	For *hp*-Legendre algorithm.