Large Scale Constrained Trajectory Optimization Using Indirect Methods

Thomas Antony
Purdue University

LARGE SCALE CONSTRAINED TRAJECTORY OPTIMIZATION USING INDIRECT METHODS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Thomas Antony

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2018

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

To my parents, Antony and Susheela and my sister Asha, for all their love and support,

To Beth, for always being by my side

ACKNOWLEDGMENTS

This work would not be possible without the support and encouragement of many great people during my time at Purdue. First and foremost, I would like to thank my advisor, Prof. Michael Grant. I consider myself very lucky to have had the opportunity to work with you and learn from you. I really appreciate your openness to new ideas as well as all of your ideas that got me started in the beginning. I enjoyed all the discussions we had and valued your insights on my work. I would not be the engineer I am today without your constant guidance and support.

I would also like to acknowledge the Air Force Research Laboratory for funding a significant portion of this research and for inviting me to be a visiting researcher at the Mathematical Modeling and Optimization Institute back in Summer 2015. I would also like to thank Colin Hurd and Mark Barglof, for supporting this work through Smart-Ag LLC and giving me the opportunity to be a part of your great team.

I am also immensely lucky to have had great lab-mates at the Rapid Design of Systems Laboratory, who created a positive and stimulating work environment. I would particularly like to thank Harish Saranathan, Justin Mansell, Kshitij Mall and Michael Sparapany. I enjoyed working with all of you and learned a lot from all the great discussions we had at the office. I would also like to thank Shreyas Subramanian, Joseph Williams, and Sean Nolan.

Finally, I would like to thank my family for their love and constant encouragement. I would also like to thank Beth Kashon for always being there for me especially when I most needed it. The work and accomplishments presented in this document would not be possible without all of your support. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

SYMBOLS

A_{ref}	Reference area, m^2
C_D	Coefficient of drag
C_L	Coefficient of lift
H	Hamiltonian
h_s	Scale height, m
h	Altitude, m
J	Cost function or Jacobian matrix
L	Path cost
m	Mass, kg
N	Order of Chebyshev polynomial series
n	Number of vehicles in multi-vehicle problem
q	Dynamic pressure, N/m^2
$\dot{Q}$	Stagnation point heat-rate, W/m^2
$\dot{Q}_{max}$	Peak heat-rate at stagnation point, W/m^2
R_E	Radius of the Earth, m
r	Radial position, m
r_n	Nose radius, m
S	Path constraint function
$t_f,\ T$	Time of flight, s
V	Velocity, m/s
$\boldsymbol{u}$	Control vector
$\boldsymbol{x}$	State vector
α	Angle of attack, rad
β	Chebyshev coefficients of solution
γ	Flight path angle, rad

θ	Longitude, rad
	Regularization parameter
$\boldsymbol{\lambda}$	Costate vector
μ	Standard gravitational parameter, m^3/s^2
$\boldsymbol{\mu}$	Lagrange multiplier for adjoining path constraints
$\boldsymbol{\nu}_0$, $\boldsymbol{\nu}_f$	Lagrange multipliers for initial and terminal point constraints, respectively
$\boldsymbol{\pi}$	Lagrange multipliers for interior point constraints and tangency/-corner conditions
$\boldsymbol{\Psi}$	Boundary conditions vector
ρ	Density of the atmosphere, kg/m^3
ρ_0	Atmospheric density at sea-level, kg/m^3

ABBREVIATIONS

3DOF	Three Degrees of Freedom
API	Application Programming Interface
BVP	Boundary Value Problem
CGL	Chebyshev Gauss Lobatto
CPU	Central Processing Unit
DAE	Differential-Algebraic Equation
GPU	Graphics Processing Unit
ICRM	Integrated Control Regularization Method
IVP	Initial Value Problem
JIT	Just-In-Time
KKT	Karush-Kuhn-Tucker
L/D	Lift-to-Drag ratio
LLVM	Low-Level Virtual Machine
MCPI	Modified Chebyshev-Picard Iteration
MDO	Multidisciplinary Design Optimization
MPBVP	Multi-Point Boundary Value Problem
MPC	Model Predictive Control
MQA	Modified Quasi-linearization Algorithm
NLP	Nonlinear Programming Problem
OCP	Optimal Control Problem
ODE	Ordinary Differential Equation
OTIS	Optimal Trajectories by Implicit Simulation
PMP	Pontryagin's Minimum Principle
POST	Program to Optimize Simulated Trajectories
PSO	Particle Swarm Optimizer

QCPI	Quasi-linear Chebyshev-Picard Iteration
SNOPT	Sparse Nonlinear Optimizer
SQP	Sequential Quadratic Programming
STM	State Transition Matrix
TPBVP	Two-Point Boundary value Problem

ABSTRACT

Antony, Thomas PhD, Purdue University, May 2018. Large Scale Constrained Trajectory Optimization using Indirect Methods . Major Professor: Michael J. Grant.

State-of-the-art direct and indirect methods face significant challenges when solving large scale constrained trajectory optimization problems. Two main challenges when using indirect methods to solve such problems are difficulties in handling path inequality constraints, and the exponential increase in computation time as the number of states and constraints in problem increases. The latter challenge affects both direct and indirect methods.

A methodology called the Integrated Control Regularization Method (ICRM) is developed for incorporating path constraints into optimal control problems when using indirect methods. ICRM removes the need for multiple constrained and unconstrained arcs and converts constrained optimal control problems into two-point boundary value problems. Furthermore, it also addresses the issue of transcendental control law equations by re-formulating the problem so that it can be solved by existing numerical solvers for two-point boundary value problems (TPBVP). The capabilities of ICRM are demonstrated by using it to solve some representative constrained trajectory optimization problems as well as a five vehicle problem with path constraints. Regularizing path constraints using ICRM represents a first step towards obtaining high quality solutions for highly constrained trajectory optimization problems which would generally be considered practically impossible to solve using indirect *or* direct methods.

The Quasilinear Chebyshev Picard Iteration (QCPI) method builds on prior work and uses Chebyshev Polynomial series and the Picard Iteration combined with the Modified Quasi-linearization Algorithm. The method is developed specifically to utilize parallel computational resources for solving large TPBVPs. The capabilities of the numerical method are validated by solving some representative nonlinear optimal control problems. The performance of QCPI is benchmarked against single shooting and parallel shooting methods using a multi-vehicle optimal control problem. The results demonstrate that QCPI is capable of leveraging parallel computing architectures and can greatly benefit from implementation on highly parallel architectures such as GPUs.

The capabilities of ICRM and QCPI are explored further using a five-vehicle constrained optimal control problem. The scenario models a co-operative, simultaneous engagement of two targets by five vehicles. The problem involves 3DOF dynamic models, control constraints for each vehicle and a no-fly zone path constraint. Trade studies are conducted by varying different parameters in the problem to demonstrate smooth transition between constrained and unconstrained arcs. Such transitions would be highly impractical to study using existing indirect methods. The study serves as a demonstration of the capabilities of ICRM and QCPI for solving large-scale trajectory optimization methods.

An open source, indirect trajectory optimization framework is developed with the goal of being a viable contender to state-of-the-art direct solvers such as GPOPS and DIDO. The framework, named *beluga*, leverages ICRM and QCPI along with traditional indirect optimal control theory. In its current form, as illustrated by the various examples in this dissertation, it has made significant advances in automating the use of indirect methods for trajectory optimization. Following on the path of popular and widely used scientific software projects such as SciPy [1] and Numpy [2], *beluga* is released under the permissive MIT license [3]. Being an open source project allows the community to contribute freely to the framework, further expanding its capabilities and allow faster integration of new advances to the state-of-the-art.

1. MOTIVATION AND BACKGROUND

1.1 Motivation

Trajectory optimization plays a key role in conceptual design and control of complex dynamical systems. Knowledge of how a system behaves under different inputs can be used to design not just the ideal control inputs for a given scenario, but the systems themselves. Depending on the dynamics of the system being designed for, trajectory optimization can be a time-consuming and computationally intensive process. There are two main classes of methods that are used for solving trajectory optimization problems, namely, direct and indirect methods. Direct methods transcribe these problems into large parameter optimization problems to be solved using a non-linear programming method such as Sequential Quadratic Programming (SQP). This in contrast to indirect methods that use optimal control theory based on calculus of variations to convert the optimization problem into a nonlinear boundary value problem. These methods are described in detail in Sections 1.2.2 and 1.2.3.

Modern research in trajectory optimization and mission design has mostly trended towards pseudo-spectral and other direct methods [4–8]. They have many advantages over indirect methods including ease of use, larger region of convergence, and usually not requiring an accurate initial guess. Their disadvantages include being highly computationally intensive and not guaranteeing optimality. Indirect methods produce solutions that satisfy the necessary conditions of optimality, guaranteeing a local minimum. However, indirect methods are often cited as being too impractical [9] or difficult to apply to real-world problems owing to mainly three challenges [10]. These challenges, and strategies to overcome them, are examined in more detail in Section 1.2.3. One of these three main challenges involves the incorporation of path inequality constraints when using indirect methods. When using Pontryagin's Minimum Principle [11, 12] to incorporate path constraints, the trajectory is split into multiple constrained and unconstrained arcs, the sequence of which, has to be known a-priori before the problem can be numerically solved. Recent work has made considerable progress [13–15] in overcoming this issue, as discussed in more detail in Sections 1.3, 2.8.1 and 2.8.3. Even with these advances, the incorporation of path inequality constraints remains non-trivial, particularly when there are multiple constraints that may be active and inactive at different points along the trajectory. One of the goals of this dissertation is to overcome this challenge and create a systematic method for incorporating multiple constraints into optimal control problems when using indirect methods.

In engineering design and in control problems dealing with systems with highly nonlinear dynamics, fast numerical methods capable of solving them are extremely valuable. For example, in the design of hypersonic entry vehicles, a vehicle shape is usually chosen prior to trajectory design. Once the trajectory has been designed for the given vehicle, an iterative, time-consuming multidisciplinary design optimization (MDO) process is performed to alter the vehicle dimensions and the trajectory until it converges to a design that can accomplish the desired mission. In Ref. 16, these iterative steps are combined to enable simultaneous design of the vehicle shape and its trajectory for a specific mission. In order to achieve this, a new trajectory optimization methodology is constructed which uses analytic aerodynamic equations along with indirect methods and continuation. This method is shown to out-pace a multi-objective particle swarm optimizer by 10×-30× and also converge to more optimal results. However, the overall procedure was still very computationally intensive and the author suggests the use of more efficient numerical solvers for faster run-time performance.

Model predictive control (MPC), also known as finite-horizon or receding horizon control, is another field that can utilize faster numerical methods for trajectory optimization. In MPC, the optimal trajectory of a dynamic system for a finite interval of time is predicted, and the resulting control is used to actuate the system [17].

MPC has been successfully demonstrated for autonomous/semi-autonomous driving [18–20]. MPC is very useful because the control it commands is generated based on the nonlinear model of the system and obeys any state and input constraints. However, in MPC systems that are currently deployed, the dynamic models tend to be relatively simple. This is because as the model gets more complex both in terms of number of states and nonlinearity, the optimization problem becomes more difficult to solve and real-time performance becomes more difficult to achieve. For example, Ref. 21 uses a simple bicycle kinematic model for designing controls of an autonomous vehicle. Even in case of the Zero Propellant Maneuver [22] on the International Space Station, which made history as the first in-flight use of a pseudospectral direct solver, the trajectories were solved on ground-based systems and uploaded to the station.

In advanced aerospace applications, the current state-of-the-art methods are still not capable of real-time optimization. Particularly in the case of hypersonic systems [23], the dynamic models tend to be highly nonlinear which makes the problem more difficult. While some model predictive controller implementations using direct methods for hypersonic vehicles exist [24–26], they generally use some kind of linearization or simplification of the model to enable real-time computation. One of the goals of this dissertation is to advance the state-of-the-art in numerical methods to be closer to a real-time optimal solution capability for these kinds of problems using indirect methods. State-of-the-art direct and indirect methods are far from being capable of real-time trajectory optimization for real-world systems especially when run on flight hardware.

One of the key requirements for creating a fast numerical solver is to leverage modern computing architectures which are trending towards highly parallel systems in place of large monolithic processors [27–29]. Numerical methods designed with characteristics capable of exploiting these parallel processors will be faster than those which only utilize a single processor [30–32]. One of the goals of this dissertation is to design a numerical method for solving boundary value problems that can exploit these architectures, thereby enabling indirect methods to rapidly solve more complex, highly constrained trajectory optimization problems.

The two advancements described above will form key components of a larger generalized rapid trajectory optimization framework that can form a viable alternative to popular direct method solvers such as GPOPS [6] and DIDO [7]. All of the optimal control problems shown in this dissertation are solved using this framework.

1.2 Trajectory Optimization Overview

1.2.1 Introduction

Trajectory optimization of real-world systems is generally considered to be a difficult and computationally intensive task. It involves the calculation of the time-history of the control variable(s) associated with a system that optimizes a given performance index while satisfying problem-specific constraints at the initial point, terminal point, and interior points as well as path constraints. Hypersonic trajectory optimization refers to specific case of trajectories of vehicles flying at hypersonic velocities through an atmosphere. This is generally more complicated than solving pure spaceflight trajectories which, in some cases, may have closed-form analytical solutions [33].

A trajectory optimization or optimal control problem is generally expressed in the form given in Eq. (1.1). J is the cost functional or performance index to be optimized. This includes ϕ, the terminal cost, and ${}_{t_0}^{t_f} L(x, u, t)\,\mathrm{d}t$, the path cost integrated over the entire trajectory. There are also initial and terminal constraints, Ψ and Φ respectively, that are to be satisfied simultaneously. The problem may also include path inequality constraints such as S as well as control limits such as C that have to be satisfied for the solution to be feasible. C^- and C^+ are the upper and lower limits of the constraint respectively.

$$\text{Min } J = \phi(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} L(\boldsymbol{x}, \boldsymbol{u}, t)\,\mathrm{d}t \tag{1.1}$$

Subject to :

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t) \tag{1.2}$$

$$\boldsymbol{\Psi}(\boldsymbol{x}(t_0), t_0) = 0 \tag{1.3}$$

$$\boldsymbol{\Phi}(\boldsymbol{x}(t_f), t_f) = 0 \tag{1.4}$$

$$\boldsymbol{S}(\boldsymbol{x}, t) \leq 0 \tag{1.5}$$

$$C^{-} \leq C(\boldsymbol{u}, t) \leq C^{+} \tag{1.6}$$

$$t_0 = 0$$

As mentioned in the previous section, there are mainly two classes of methods used to solve such problems [10, 34]. Direct methods discretize the problem space into a large number of nodes, and transform the problem into a large parameter optimization problem, which is then solved using an optimization algorithm. Indirect methods use optimal control theory to convert the optimization problem into a boundary value problem, which is then solved using a numerical solver [12]. The use of indirect methods poses additional challenges such as the introduction of mathematical entities called "costates" which complicates the numerical solution process as described later in Section 1.2.3. A comparison of the advantages and disadvantages of direct and indirect methods is listed in Table. 1.1 [35]. Direct methods tend to be robust to initial guess and are applicable to a wide range of problems while indirect methods have features that make it suitable for use in applications that require rapid convergence to optimal solutions.

Table 1.1. Comparison of Direct and Indirect Methods

	Advantages	Disadvantages
Direct methods	Large region of convergence	Computationally intensive
	Easy to setup and use	Optimality not guaranteed
Indirect methods	Necessary conditions satisfied	Small region of convergence
	Fast convergence	Co-states introduced

1.2.2 Direct Methods

Introduction

Direct methods were developed as an alternative to the classical indirect methods for solving optimal control problems, and they became more feasible with the advent of more powerful computers. Instead of using optimal control theory to derive the necessary conditions of optimality, the continuous control history is discretized into a finite number of nodes [36, 37]. A general numerical optimization algorithm is then used to adjust these nodes until an optimal solution is found. One of the simplest ways to implement this is to simulate the entire trajectory for the given control history during every iteration. However, this can be very computationally intensive and is impractical for most real world scenarios.

More advanced direct methods discretize the control and state more intelligently to allow for the use of various quadrature schemes for implicit integration. The use of quadrature rules result in the discretized optimization problem being converted into a sparse nonlinear programming problem that can be efficiently solved using a solver such as SNOPT [38]. Depending on the type of discretization used, there are different types of direct methods that have been developed including Collocation [39–41], Differential Inclusion [42] and Pseudospectral methods [43, 44].

In some cases, such as in Ref. 45 and 46, optimal control theory is used to derive the necessary conditions for a hypersonic trajectory problem. However, the trajectory is solved for a simplified problem involving constant altitude flight with either a constant velocity or a predefined deceleration profile. This solution is then used to validate the results from a direct solver. In other examples, such as in Ref. 47, the solution from a direct solver is validated by checking a subset of the necessary conditions of optimality. When using direct methods, constraints are adjoined to the underlying parameter optimization problem using Lagrange multipliers. The covector mapping theorem [48–50] allows the computation of the necessary conditions of optimality from these Lagrange multipliers in certain direct solvers such as DIDO. All these cases point to the direct optimization being the preferred method in the design community for solving non-trivial optimization problems while certain indirect methods may be used for validating them.

Direct Shooting and Global Methods

Direct shooting is one of the simplest implementations of a direct method. The control history is discretized into a finite number of nodes, and the numerical optimizer searches the design space of these discrete control values to optimize the cost function. During each evaluation of the cost function, the dynamics of the system are repeatedly propagated using some numerical integration scheme. In direct shooting methods, the dynamic equations are propagated from different starting conditions, and a state transition matrix is used to perform updates. Ref. 51 implements a direct multiple shooting method that is completely derivative-free and is able to solve optimal control problems that include free end-time, free parameters as well as discontinuous multi-phase trajectories.

Global optimization methods such as genetic algorithms [52,53] and particle swarm optimizers (PSO) [54,55] are suited for integration based optimization as well. These methods do not require an accurate initial guess and can explore the design space in an automated manner without performing a full-factorial search. Their disadvantage is that the optimizers completely ignore the physics of the problems and there is no guarantee that the solution is locally or globally optimal. When using these global methods, path constraints may be implemented using penalty functions which may require further tuning by the designer before they are appropriately balanced with the objective function. Global optimizers can also be used as a starting point to generate coarse initial guesses to be used by more precise direct or indirect methods. Ref. 56 describes a hybrid method that uses a particle swarm optimizer to generate the initial guess for a pseudospectral direct solver.

Collocation and Differential Inclusion

Collocation methods discretize the entire solution space including states and controls, and use some implicit integration scheme for more efficient propagation of the dynamic system. Early implementations of collocation were developed by Hargraves and Paris [57–59] that used piecewise Chebyshev polynomials to represent the states and controls. Penalty functions were used to convert problems with path constraints into unconstrained parameter optimization problems and solved using a full second-order modified Newton algorithm. Ref. 39 describes a later work by the same authors, where the state and control histories are represented by piecewise cubic polynomials. An implicit integration scheme based on Hermite interpolation was used to convert the optimal control problem into a nonlinear programming problem. The advantage of this method was that it was easier to extend to general problems involving path constraints, control inequalities, and discontinuous states. One characteristic of the quadrature scheme used in these methods is that it can be shown that the converged solution is equivalent to the solution of an explicit fourth-order Runge-Kutta integration scheme [42].

While collocation can handle path constraints by including it directly in the nonlinear programming problem, it comes at the cost of higher dimensionality of design space as the number of nodes increases. One way to overcome this for certain classes of problems is to use differential inclusion [60]. Differential inclusion is the process of solving for the controls in terms of states and thereby eliminating them from the problem [61]. This can help considerably decrease the dimensionality of the problem while retaining the advantages of collocation. However, this is only possible for a subset of trajectory optimization problems in which the controls can be solved as a function of states. Ref. 42 compares the number of NLP parameters required to obtain the same accuracy with and without using differential inclusion. The author shows the differential inclusion method requires the use of quadrature rules with very low accuracy while collocation methods in general may use implicit integration rules of very high accuracy and are hence not limited in this manner.

Pseudospectral Methods

Direct methods have been used to solve optimal control problems in a wide range of applications [22, 62–65]. However, one of their drawbacks compared to indirect methods has always been the lack of a guarantee of optimality. Direct methods essentially discretize the problem first and then apply the optimality conditions to the discrete problem (Karush-Kuhn-Tucker or KKT conditions). In contrast, when using indirect methods the necessary conditions of optimality are first derived for the continuous problem and then the resulting boundary value problem is solved, typically using a numerical method, to obtain the solution. Relating the KKT multipliers in direct methods to the costates from indirect methods could help verify the optimality of the solution. This was proven difficult until the development of pseudospectral methods owing to discrepancies between the KKT multipliers and costates in certain problems even when the trajectories and control histories matched. This made it difficult to show that direct methods were in fact arriving at the same solution. It was discovered that by using specially constructed discretization, the KKT multipliers from pseudospectral methods can be mapped to the corresponding costates from indirect methods using the Covector Mapping Theorem [49, 66, 67]. This can be used to validate the results of pseudospectral methods using the necessary conditions of optimality from indirect methods. Some of the most widely used design software such as GPOPS [6] and DIDO [7] are based on pseudospectral algorithms.

1.2.3 Indirect Methods

Calculus of Variations

Calculus of variations is a branch of mathematics that has applications ranging from optics to quantum mechanics to aerospace engineering, and is the progenitor of modern optimal control theory. While early descriptions of relevant problems can be traced back as far as 300 A.D. [68], the most famous problem associated with calculus of variations is the Brachistochrone problem, posed by Johann Bernoulli in *Acta Eruditorum* in 1696 [69]. The word “brachistochrone” originates from the Greek words for “shortest” and “time”. Bernoulli’s original problem statement was,

> Given two points A and B in a vertical plane, what is the curve traced out by a point acted on only by gravity, which starts at A and reaches B in the shortest time?

This seemingly simple problem attracted the attention of such great minds as Newton, Lagrange, and Leibniz and eventually resulted in the rise of a field of mathematics known as calculus of variations. Lagrange approached the problem by considering sub-optimal trajectories close to the optimal path. Euler and Lagrange independently developed the differential equation that is now known as the Euler-Lagrange Equation [70]. By following Lagrange’s technique, a more generalized set of necessary conditions of optimality can be formulated, using the Euler-Lagrange theorem [71]. This theorem can be applied to solve optimal control problems in which the path and optimal control profile that optimizes a cost functional is computed. It forms the foundation of indirect trajectory optimization. The classical Brachistochrone problem and its variants are used in multiple parts of this dissertation for the validation of optimal control algorithms.

Necessary Conditions of Optimality

Indirect methods optimize the cost functional J from Eq. (1.1) by formulating a two-point or multi-point boundary value problem (BVP) that represents the necessary conditions of optimality. If these boundary conditions are satisfied, the solution will be locally optimal in the design space. In order to derive the necessary conditions, the dynamic equations of the system are augmented with a set of mathematical variables called costates. The BVP representing the necessary conditions of optimality is then formulated by applying the Euler-Lagrange equation [71].

The Hamiltonian is defined as shown in Eq. (1.7), where $\boldsymbol{\lambda}$ is the costate vector with its corresponding dynamic equations defined in Eq. (1.8). The optimal control law, $\boldsymbol{u}(t)$, is obtained as a function of the states and costates by solving Eq. (1.9). The initial and terminal boundary conditions on the costates are specified in Eqs. (1.10) and (1.11), where $\boldsymbol{\nu_0}$ and $\boldsymbol{\nu_f}$ are sets of undetermined parameters which are used to adjoin the state boundary conditions to the cost functional. If the total-time of the trajectory is not specified, it is determined by the free-final time condition in Eq. (1.12). The necessary conditions of optimality are defined by Eqs. (1.8–1.12), and they form a well-defined Two-Point Boundary Value Problem (TPBVP) that can be solved using the shooting method and other numerical solvers as described in Section 1.4.

$$H = \; L(x,u,t) + \boldsymbol{\lambda}^T(t)\boldsymbol{f}(\boldsymbol{x},\boldsymbol{u},t) \tag{1.7}$$

$$\dot{\boldsymbol{\lambda}} = -\frac{\partial H}{\partial \boldsymbol{x}} \tag{1.8}$$

$$\frac{\partial H}{\partial \boldsymbol{u}} = 0 \tag{1.9}$$

$$\boldsymbol{\lambda}(t_0) = -\boldsymbol{\nu}_{\boldsymbol{0}}^T \frac{\partial \boldsymbol{\Psi}}{\partial \boldsymbol{x}(t_0)} \tag{1.10}$$

$$\boldsymbol{\lambda}(t_f) = \frac{\partial \phi}{\partial \boldsymbol{x}(t_f)} + \boldsymbol{\nu}_{\boldsymbol{f}}^T \frac{\partial \boldsymbol{\Phi}}{\partial \boldsymbol{x}(t_f)} \tag{1.11}$$

$$\left. H + \frac{\partial \phi}{\partial t} + \boldsymbol{\nu}_{\boldsymbol{f}}^T \frac{\partial \boldsymbol{\Phi}}{\partial t} \right|_{t=t_f} = 0 \tag{1.12}$$

Path Constraints & Interior Point Constraints

The presence of path constraints and interior point constraints further complicates the boundary conditions by introducing corner conditions in certain costates and effectively splitting the trajectory into multiple arcs. This may also introduce corners in the control profile at the junction points of these arcs, where the derivative of the control is discontinuous. Path constraints are usually of the form shown in Eq. (1.13). In order to obtain the control law for the constrained arc, time derivatives of the path constraints are taken until the control variable appears explicitly. Assuming that this happens with the q^{th} derivative, the Hamiltonian is augmented as shown in Eq. (1.14), and the control law for the constraint boundary is obtained by solving $\boldsymbol{S}^{(q)} = 0$.

$$\boldsymbol{S}(\boldsymbol{x}, t) \leq 0 \tag{1.13}$$

$$H = L + \boldsymbol{\lambda}^T \boldsymbol{f} + \boldsymbol{\mu}^T \boldsymbol{S}^{(q)} \tag{1.14}$$

The addition of path constraints also modifies the dynamic equations of the costates along the constrained arcs as shown in Eq. (1.15), where the multipliers $\boldsymbol{\mu}$ are calculated by solving Eq. (1.16).

$$\dot{\boldsymbol{\lambda}} = -\frac{\partial H}{\partial \boldsymbol{x}} = -L_x - \boldsymbol{\lambda}^T \boldsymbol{f}_x - \boldsymbol{\mu}^T \boldsymbol{S}_{\boldsymbol{x}}^{(q)} \tag{1.15}$$

$$\frac{\partial H}{\partial \boldsymbol{u}} = L_u + \boldsymbol{\lambda}^T \boldsymbol{f}_u + \boldsymbol{\mu}^T \boldsymbol{S}_{\boldsymbol{u}}^{(q)} = 0 \tag{1.16}$$

The states are continuous at the entry (t_1) and exit (t_2) of the constrained arc as shown in Eq. (1.17). Corner conditions on costates are chosen such that the costates are continuous at the exit of the constrained arc as shown in Eq. (1.18). The tangency conditions described in Eq. (1.19) and corner conditions in Eq. (1.20) and Eq. (1.21) apply at the entry point of the constrained arc.

$$\begin{aligned} \boldsymbol{x}({t_1}^+) &= \boldsymbol{x}({t_1}^-) \\ \boldsymbol{x}({t_2}^+) &= \boldsymbol{x}({t_2}^-) \end{aligned} \tag{1.17}$$

$$\begin{aligned} \boldsymbol{\lambda}({t_2}^+) &= \boldsymbol{\lambda}({t_2}^-) \\ H({t_2}^+) &= H({t_2}^-) \end{aligned} \tag{1.18}$$

$$\boldsymbol{N}(\boldsymbol{x},t) = \begin{bmatrix} S(\boldsymbol{x},t) \\ S^{(1)}(\boldsymbol{x},t) \\ S^{(2)}(\boldsymbol{x},t) \\ . \\ . \\ . \\ S^{(q-1)}(\boldsymbol{x},t) \end{bmatrix} \tag{1.19}$$

$$\boldsymbol{\lambda}({t_1}^+) = \boldsymbol{\lambda}({t_1}^-) + \boldsymbol{\Pi}^T \boldsymbol{N}_{\boldsymbol{x}} \tag{1.20}$$

$$H({t_1}^+) = H({t_1}^-) + \boldsymbol{\Pi}^T \boldsymbol{N}_t \tag{1.21}$$

Interior point constraints are very similar to the tangency conditions in a path constraint, described in Eq. (1.19). It can be considered to be a case where the constrained arc is infinitesimally small. The states remain continuous across the "junction", but the costates and the Hamiltonian may have jump conditions imposed on them. An interior point constraint is defined as shown in Eq. (1.22) at some interior time, t_1. Introducing interior point constraints into the problem results in the continuity and corner conditions in Eq. (1.23), where $\mathbf{\Pi}$ is a vector of unknown parameters.

$$\boldsymbol{N}(\boldsymbol{x}(t_1), t_1) = 0 \tag{1.22}$$

$$\begin{aligned} \boldsymbol{x}({t_1}^-) &= \boldsymbol{x}({t_1}^+) \\ \boldsymbol{\lambda}({t_1}^-) &= \boldsymbol{\lambda}({t_1}^+) + \mathbf{\Pi}^T \boldsymbol{N}_{\boldsymbol{x}} \\ H({t_1}^-) &= H({t_1}^+) + \mathbf{\Pi}^T \boldsymbol{N}_t \end{aligned} \tag{1.23}$$

The Multi-Point Boundary Value Problem (MPBVP) resulting from applying the necessary conditions of optimality discussed above, can be solved using the multiple shooting method. The biggest challenge to use this approach for solving constrained problems is that the sequence of constrained and unconstrained arcs must be known a-priori in order to set up the multi-point boundary value problem. This is usually impractical for real-world problems, especially when there are multiple path inequality constraints which may be active or inactive at multiple points along the trajectory. One of the goals of this dissertation is to formulate an alternative method to incorporate path constraints into optimal control problems without requiring knowledge of the arc-sequence while still retaining the ability to use existing numerical BVP solver algorithms.

Recent Advances in Rapid Design using Indirect Methods

Indirect methods are often cited as impractical for use in complex problems owing to three main reasons [10].

- Use of indirect methods require knowledge of optimal control theory and derivation of necessary conditions of optimality
- It is difficult to ascertain the correct sequence of unconstrained and constrained arcs a-priori when the problem includes path inequality constraints in state and/or control variables.

- Providing a good initial guess to start the numerical solution process is very challenging.

Recent developments in this field has however helped overcome some of these challenges to a certain extent. The formulation of the necessary conditions mostly involves taking derivatives and some analytical root-solving in the case of deriving the optimal control law [12]. While these may be difficult, error-prone, and possibly impractical to do by hand [9], the advent of modern symbolic math engines has made it so these operations can be completely automated. Symbolic math engines such as SymPy [72], Mathematica [73] and the MATLAB Symbolic Math Toolbox [74] allow the automatic derivation of the necessary conditions of optimality without requiring any knowledge of optimal control theory on the part of the designer. The MATLAB symbolic toolbox and Mathematica have been used successfully to derive the necessary conditions for real-world nonlinear systems [14,75] including some involving elaborate analytical hypersonic aerodynamic equations [16]. Ref. 15 demonstrates how this method can be extended to incorporate higher fidelity aerodynamic and atmospheric models into optimal control problems while still using indirect methods. In contrast, direct methods, especially pseudospectral and collocation methods, are able to handle path inequality constraints in optimization problems as part of the NLP formulation as described in Section 1.2.2.

References 13 and 14 illustrate a homotopy continuation method for introducing path constraint arcs one at a time which allows the use of indirect methods without knowing the sequence of arcs a-priori. While this method can certainly be used to solve some non-trivial aerospace problems as demonstrated in the works cited above, the handling of path constraints using indirect methods remains a challenging task. The continuation methodology only works in those cases where the constrained arcs can be introduced one at a time into an unconstrained trajectory. The method becomes more difficult to use if the constraints are active at the beginning or end of the trajectory or if multiple constraints are active in the same region of the trajectory. Optimal control theory requires the trajectory to be split into phases at any point that a constraint becomes active or inactive. This results in what is called a multi-point boundary value problem (MPBVP), which has boundary conditions at multiple interior points along a trajectory. These MPBVPs can be challenging to solve numerically as they grow larger. The homotopy approach therefore fails to scale as the number of constraints increases, or if one or more constraints are active multiple times. This is one of the key motivations for seeking a new way of formulating path constraints in indirect methods. One of the goals of this research is to formulate a general method to simplify the handling of path constraints when using indirect methods. This is further examined in Chapter 2 which describes a new, systematic method for incorporating path constraints in optimal control problems – the Integrated Control Regularization Method (ICRM).

The requirement for an accurate initial guess is still a challenge when using indirect methods, especially for sensitive problems. The use of a homotopy continuation method can resolve this issue to a certain extent. This process is detailed in Section 1.3.

1.3 Continuation

One of the major drawbacks of indirect methods mentioned in Section 1.2.3 was the requirement for an accurate initial guess for solutions to reliably converge. In Ref. 76, a coarse approximation from a particle swarm optimizer is used to generate initial guess for costates to be used in an indirect solver. One strategy for generating an initial guess is by starting with analytic solutions of a related, but simpler problem. Continuation builds on this idea and bypasses the need for an accurate initial guess by first solving a trivial problem and then gradually changing it until it becomes the desired problem. The solution from the prior step is used as the initial guess for each subsequent step. This method relies on the inherent connectedness among families of optimal solutions. It is even useful for enhancing direct methods when solving certain difficult problems as noted in Ref. 77.

In Ref. 14, the maximum terminal energy trajectory for a hypersonic mission was calculated using indirect methods and continuation. The problem consisted of a hypothetical high-lift hypersonic vehicle with angle-of-attack and bank angle as the controls. The hypothetical mission had post-boost/impact geometry constraints as well a peak heat-rate path constraint. In order to seed the continuation process, it was necessary to supply it with a relatively simple trajectory as an initial guess. Since the objective was to maximize the velocity at impact, a simple trajectory that could be solved easily was one that flies nearly straight down from the assumed post-boost staging condition. The optimal trajectory for reaching a target almost directly underneath the staging location, with maximum velocity, would be a near-ballistic trajectory that minimizes the drag coefficient of the vehicle. Hence, the Allen and Eggers trajectory solution [78] for ballistic trajectories was used to construct a high quality initial guess to this optimization problem. The costates for this initial trajectory were constructed by reverse integration from the terminal point.

This initial guess trajectory, being very close to the optimum, rapidly converges to a solution. Starting with this solution, the targeted location is moved until it matches the desired terminal conditions as shown in Figs. 1.1 and 1.2. At the end of this process, a maximum terminal velocity trajectory connecting the post-boost staging location and the targeted impact location is obtained.

A similar approach can also be used to introduce path constraints one at a time into an unconstrained solution. A new constrained arc is introduced at the point of maximum constraint violation and then the constraint limit is changed using continuation until the desired value is reached. This is a viable strategy for use with problems containing one or two constraints and is described in Refs. 13, 14, and 15. This strategy is also used later in Sections 2.8.1 and 2.8.3.

In all of the test cases in the later sections of this dissertation, a continuation strategy is used when solving complex optimal control problems. In most of these scenarios, the initial guess is created by propagating the dynamic equations of the system for a short period of time (typically 0.1 seconds), using a constant initial guess

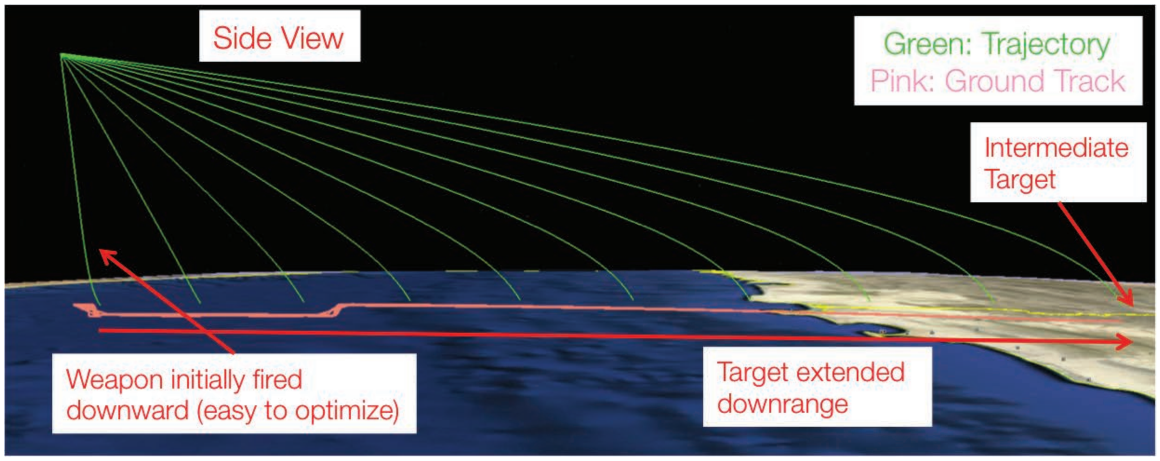

Side View
Green: Trajectory
Pink: Ground Track
Intermediate Target
Weapon initially fired downward (easy to optimize)
Target extended downrange

Figure 1.1. Side View of Trajectory during Continuation Process [79]

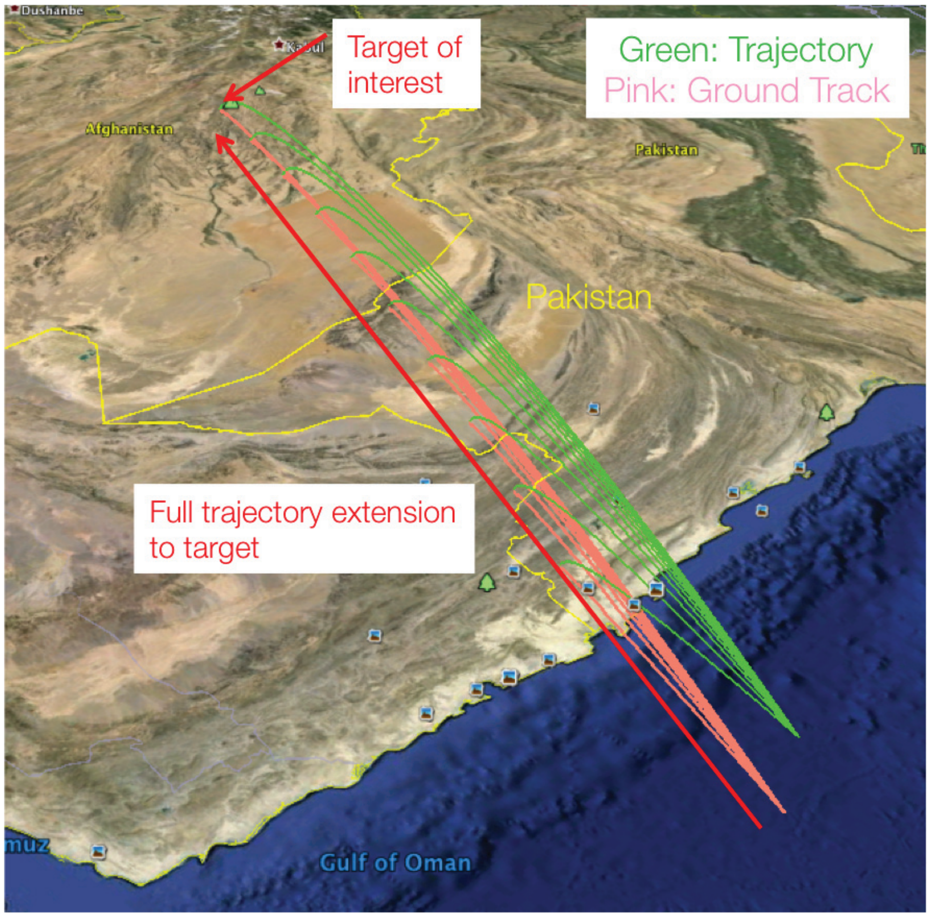


Figure 1.2. Full Extension of Trajectory to Final Target [79]

for the costates. The boundary conditions are then changed gradually in a series of steps until the desired solution is obtained. Ref. 80 shows a more advanced method of performing continuation by using a graph-search methodology. Such methods allow for further automation of continuation requiring even lesser intervention from the designer.

1.4 Overview of Numerical Methods for Boundary Value Problems

1.4.1 Introduction

Most real-world trajectory optimization problems involve nonlinear dynamic equations for which closed-form analytical solutions cannot be found. Both direct and indirect methods face challenges when the number of states and constraints in a problem increase beyond a certain limit. Direct pseudospectral methods rely on nonlinear programming solvers such as SNOPT [38]. Trajectory problems with a large number of states and constraints are very difficult to solve using direct methods because the underlying parameter optimization methods often experience a significant increase in computation time as the problem grows larger. When using direct methods, it is often required that certain workarounds be used to decrease the dimensionality of large scale optimization problems to a manageable level. One example of this can be seen in Ref. 81 where the covariance matrix is propagated separately during the evaluation of the cost function in order to make the problem computationally feasible.

In case of the indirect methods, the ability to rapidly and reliably solve nonlinear boundary value problems is very important. Though there is a large body of work in the literature about numerical methods for solving boundary value problems, there still exist challenges and a need for improvement of current methods. Some of these issues are:

- Significantly reducing the computational intensity of the algorithms

- Designing the algorithms to leverage emerging parallel computing architectures especially when solving large dimensional problems

Even if multiple path constraints can be addressed in indirect methods (Chapter 2), the ability to efficiently solve these large nonlinear boundary value problems in a reasonable amount of time is still very important. While a two-point boundary value problem is certainly easier to solve than a multi-point boundary value problem, they still pose a significant challenge once they become sufficiently large. Leveraging highly parallel computing resources such as Graphics Processing Units (GPUs) and multi-core processors could be crucial in rapidly solving these large boundary value problems.

The computational speeds of monolithic CPU processors have plateaued in the last decade. The technology is trending towards highly parallel architectures [82]. Hence it would be prudent for new computational methods to be designed to take advantage of highly parallel computational architectures such as GPUs and multi-core CPUs. The computational architecture of a GPU is significantly different from a conventional CPU processor. Adapting an algorithm to run on a GPU or on multiple CPUs requires careful design of operations and data structures to fully leverage the architecture. There are many factors to consider, the key among which, is the inherent parallelism of an algorithm. Parallel processors are best used for algorithms that involve large numbers of independent compute-intensive operations, without requiring much cross-communication. As such, the design of an inherently parallel numerical method for solving boundary value problems is detailed in Chapter 3.

The two main classes of numerical methods for solving boundary value problems are shooting methods and collocation methods.

1.4.2 Single Shooting Method

If $\boldsymbol{X}$ is the augmented state vector consisting of both the states and costates as shown in Eq. (1.24a), a TPBVP resulting from a trajectory optimization problem takes the form described by Eqs. 1.24b–1.24c.

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\lambda} \end{bmatrix} \tag{1.24a}$$

$$\boldsymbol{b}(\boldsymbol{X}_0, \boldsymbol{X}_f) = 0 \tag{1.24b}$$

$$\dot{\boldsymbol{X}}(t) = \boldsymbol{f}(\boldsymbol{X}, t) \tag{1.24c}$$

where $\boldsymbol{X}_0$ and $\boldsymbol{X}_f$ are the values of $\boldsymbol{X}$ at the times t_0 and t_f respectively.

The single shooting method root-solves for the values of $\boldsymbol{X}_0$ and $\boldsymbol{X}_f$ that satisfy these conditions. In order to do so, an initial guess for $\boldsymbol{X}_0$ is used to propagate the dynamic equations of the system ($\dot{\boldsymbol{X}}$). Along with the dynamic equations we also propagate equations describing the sensitivity of the system which form the state transition matrix (STM), Φ. $\Phi(t_f)$ and $\boldsymbol{X}_f$ are obtained by propagating Eqs. (1.24c) and (1.25) and are used to evaluate the residual error ( ) in the boundary conditions, i.e., the value of $\boldsymbol{b}(\boldsymbol{X}_0, \boldsymbol{X}_f)$.

$$\dot{\Phi} = F \cdot \Phi, \quad F = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{X}}, \quad \Phi(t_0) = I_N \tag{1.25}$$

This residual is used to compute a correction ($\Delta \boldsymbol{X}_0$) which is then used to update the initial guess, as shown in Eqs. (1.26a–1.26e). If the initial guess is sufficiently close to the solution, a series of such updates will drive the residual error to zero leading to a converged solution.

$$\boldsymbol{b}(\boldsymbol{X}_0, \boldsymbol{X}_f) = \tag{1.26a}$$

$$\boldsymbol{b}(\boldsymbol{X}_0 + \Delta\boldsymbol{X}_0, \boldsymbol{X}_f) = 0 \tag{1.26b}$$

$$\implies \boldsymbol{b}(\boldsymbol{X}_0 + \Delta\boldsymbol{X}_0, \boldsymbol{X}_f) - \boldsymbol{b}(\boldsymbol{X}_0, \boldsymbol{X}_f) = - \tag{1.26c}$$

Using a first-order expansion of the above expression,

$$\boldsymbol{b}(\boldsymbol{X}_0, \boldsymbol{X}_f) + \frac{\mathrm{d}\boldsymbol{b}}{\mathrm{d}\boldsymbol{X}_0}\Delta\boldsymbol{X}_0 - \boldsymbol{b}(\boldsymbol{X}_0, \boldsymbol{X}_f) = - \tag{1.26d}$$

$$\implies \quad \frac{\partial\boldsymbol{b}}{\partial\boldsymbol{X}_0} + \frac{\partial\boldsymbol{b}}{\partial\boldsymbol{X}_f}\frac{\partial\boldsymbol{X}_f}{\partial\boldsymbol{X}_0} \quad \Delta\boldsymbol{X}_0 = - \tag{1.26e}$$

The correction vector $\Delta\boldsymbol{X}_0$ is obtained by solving the linear system in Eq. (1.27).

$$(M + N\Phi)\,\Delta\boldsymbol{X}_0 = - \tag{1.27}$$

where M and N are the Jacobian matrices, obtained by taking partial derivatives of the boundary conditions $\boldsymbol{b}$ with respect to $\boldsymbol{X}_0$ and $\boldsymbol{X}_f$ respectively, and Φ is the sensitivity matrix of the system. This is called the single shooting method.

1.4.3 Multiple Shooting Method

When a trajectory optimization problem contains path inequality or interior point constraints, the trajectory is split into multiple arcs, with the possibility of discontinuities at the junctions. These boundary value problems are called multi-point boundary value problems (MPBVP). The problem may also contain scalar parameters such as the Lagrange multipliers, $\boldsymbol{\nu}_0$, $\boldsymbol{\nu}_f$, and $\boldsymbol{\pi}$, that have to be solved along with the state and costate trajectories. In such cases, the correction vector is computed using an extension of the single shooting method, called the multiple shooting method. The general form of a MPBVP is as follows:

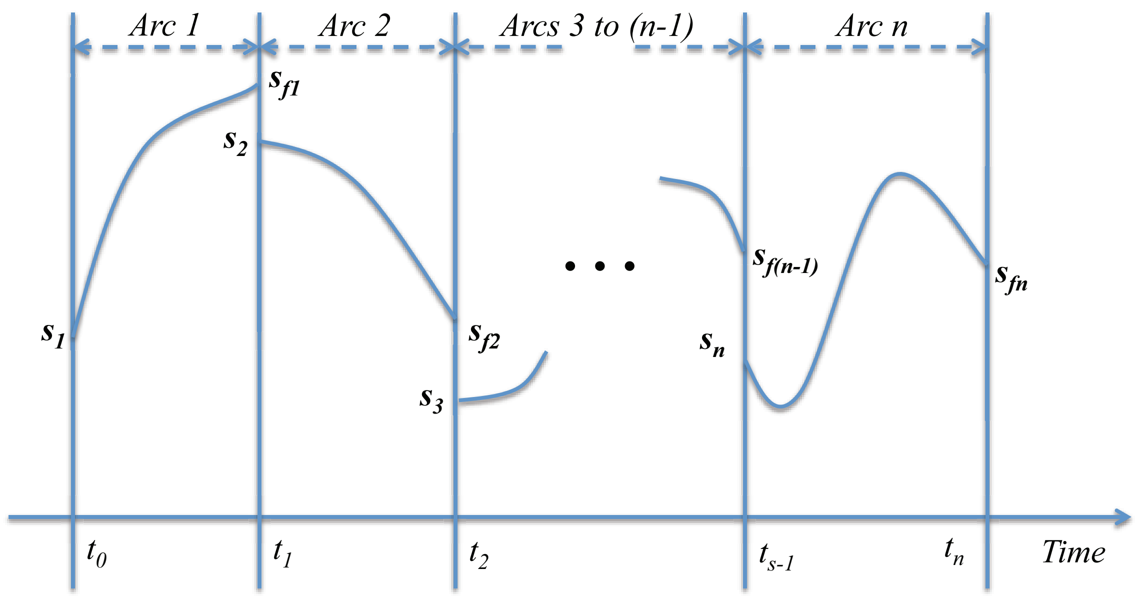


Figure 1.3. A Multi-Point Boundary Value Problem.

$$\boldsymbol{b}(\boldsymbol{s}_1, \boldsymbol{s}_{f1}, \boldsymbol{s}_2, \boldsymbol{s}_{f2} \ldots \boldsymbol{s}_n, \boldsymbol{s}_{fn}, \boldsymbol{p}) = 0$$

$$\dot{\boldsymbol{s}} = \begin{cases} \boldsymbol{f}_1(t, \boldsymbol{s}) & \text{if } t_0 \leq t \leq t_1 \\ \boldsymbol{f}_2(t, \boldsymbol{s}) & \text{if } t_1 \leq t \leq t_2 \\ & \vdots \\ \boldsymbol{f}_n(t, \boldsymbol{s}) & \text{if } t_{n-1} \leq t \leq t_n \end{cases}$$

where $\boldsymbol{s}_1$, $\boldsymbol{s}_2$, …, $\boldsymbol{s}_n$ are the values at the left endpoints of the arcs, $\boldsymbol{s}_{f1}$, $\boldsymbol{s}_{f2}$, …, $\boldsymbol{s}_{fn}$ are the values at the right endpoints of the arcs as shown in Fig. 1.3, and $\boldsymbol{p}$ is the set of scalar parameters.

To solve this problem, we first compute a Jacobian matrix J, which is of the form:

$$J = \begin{matrix} M_1 + N_1\Phi_1 & M_2 + N_2\Phi_2 & \cdots & M_n + N_n\Phi_n & J_p \end{matrix} \tag{1.28}$$

$$\begin{aligned}
M_1 &= \frac{\partial \boldsymbol{b}}{\partial \boldsymbol{s}_1}, M_2 = \frac{\partial \boldsymbol{b}}{\partial \boldsymbol{s}_2}, \cdots \ M_n = \frac{\partial \boldsymbol{b}}{\partial \boldsymbol{s}_n}, \\
N_1 &= \frac{\partial \boldsymbol{b}}{\partial \boldsymbol{s}_{f1}}, N_2 = \frac{\partial \boldsymbol{b}}{\partial \boldsymbol{s}_{f2}} \ \cdots \ N_n = \frac{\partial \boldsymbol{b}}{\partial \boldsymbol{s}_{fn}}, \\
J_p &= \frac{\partial \boldsymbol{b}}{\partial \boldsymbol{p}}
\end{aligned} \tag{1.29}$$

The sensitivity matrices for each arc is computed separately as Φ_1,Φ_2, $\ldots \Phi_n$. The correction vector $\Delta\boldsymbol{s}$ will consist of corrections to $\boldsymbol{s_1}$, $\boldsymbol{s_2}$, $\ldots$, $\boldsymbol{s_n}$ and $\boldsymbol{p}$, and it is computed by solving the following linear system.

$$J\Delta\boldsymbol{s} = - \tag{1.30}$$

Prior work has shown that it is possible to accelerate the multiple shooting method by leveraging GPUs [30] or multi-core processors [32]. However, the fundamentally sequential nature of the shooting method limits the degree to which it can be parallelized. This motivates the need for the development of better numerical algorithms which are implicitly parallel in nature and can hence better leverage parallel processors.

1.4.4 Collocation

Collocation methods that use piecewise polynomials, similar to the ones used for direct optimization, can also used for solving boundary value problems [83–86]. One of the most widely used versions of collocation is implemented in MATLABś *bvp4c* and *bvp5c* solvers, based on work by Shampine [87–89].

The collocation method implemented in *bvp4c* was originally designed to solve two-point boundary value problems of the form:

$$\mathbf{y} = \mathbf{f}(x, \mathbf{y}, \mathbf{p}), \qquad a \leq x \leq b \tag{1.31a}$$

$$\boldsymbol{b}(\mathbf{y}(a), \mathbf{y}(b), \mathbf{p}) = 0 \tag{1.31b}$$

where x is the independent variable, $\mathbf{y}$ the vector of dynamic states, and $\mathbf{p}$ a vector of unknown parameters. The method was later expanded to also solve multi-point boundary value problems.

The approximate solution $\mathbf{S}(x)$ is expressed as a continuous function consisting of piecewise polynomials defined on a discrete mesh. It satisfies the boundary conditions, $\boldsymbol{b}(\mathbf{S}(a), \mathbf{S}(b), \mathbf{p}) = 0$, and satisfies the differential equations of the system at both ends and mid point of each sub-interval. These conditions are defined mathematically as:

$$\mathbf{S}\,(x_n) = \mathbf{f}(x_n, \mathbf{S}(x_n)) \tag{1.32a}$$

$$\mathbf{S}\,((x_n + x_{n+1})\,/2) = \mathbf{f}(((x_n + x_{n+1})\,/2, \mathbf{S}((x_n + x_{n+1})\,/2)) \tag{1.32b}$$

$$\mathbf{S}\,(x_{n+1}) = \mathbf{f}(x_{n+1}, \mathbf{S}(x_{n+1})) \tag{1.32c}$$

Eqs. (1.32) define a set of nonlinear algebraic equations that can be solved to obtain the coefficients of $\mathbf{S}(x)$. The residual error in the differential equations is then defined as

$$\mathbf{r}(x) = \mathbf{S}\,(x) - \mathbf{f}\,(x, \mathbf{S}(x), \mathbf{p}) \tag{1.33}$$

and the residual error in boundary conditions as $\boldsymbol{b}\,(\mathbf{S}(a), \mathbf{S}(b), \mathbf{p})$. The size of the residual error is used to control the mesh size depending on the error tolerance specified by the user. Ref. 90 describes an adaptive mesh selection strategy that be used to control the true error, $\mathbf{e}(x) = \mathbf{S}(x) - \mathbf{y}(x)$, allowing the use for tighter tolerances on coarser grids. This strategy is used by MATLAB's *bvp5c* solver for mesh refinement.

Unlike shooting methods, when using collocation the solution is approximated over the whole interval $[a, b]$. The nonlinear algebraic equations are solved iteratively after linearization. In certain problems, this results in more numerical stability for solving sensitive problems. However, *bvp4c* and associated methods still suffer the same drawbacks as direct collocation when solving large-dimensional problems with highly nonlinear dynamics. The collocation method as it currently exists is not inherently parallel and therefore is not able to effectively leverage parallel processors. Some prior work has shown that a GPU implementation of multiple shooting offers a speed-up of 2x-4x over *bvp4c* for relatively small optimization problems [30].

Another class of boundary value solvers exist that are very similar to collocation methods in terms of how they discretize the solution space. These are the Chebyshev-Picard Iteration algorithms [31,91,92] further described in Chapter 3. These methods approximate the solution using Chebyshev polynomials and use the Picard-Lindelöf Theorem [93] to perform solution updates that do not require partial derivative information. The numerical operations in these algorithms mainly consist of matrix-vector operations which are highly parallelizable. The new parallel numerical method developed in Chapter 3 is based on this class of methods.

1.5 Trajectory Optimization Frameworks

Many trajectory optimization tools exist that use direct methods such as GPOPS [6], DIDO [7], the Optimal Trajectories by Implicit Simulation Tool (OTIS) [94], and the Program to Optimize Simulated Trajectories (POST) [95]. DIDO uses a Legendre-Gauss-Lobatto mesh for discretization while GPOPS uses a Legendre-Gauss mesh. Both methods are implemented in MATLAB [96] and have been used to solve a wide range of trajectory optimization problems. DIDO has been used to solve trajectory and maneuver problems for both aircraft [97] and spacecraft [63, 98, 99], reusable launch vehicles [62], and many other real-world problems [64]. It also made history from a mathematical perspective as it was the first ever use in-flight of Pseudospectral optimal control theory when the Zero Propellant Maneuver (ZPM) experiment was successfully deployed on the International Space Station (ISS) in 2006 [22]. However, it is to be noted that the trajectory for the ZPM was computed on the ground and uploaded to the ISS where it was tracked using traditional controllers. The pseudospectral method was not adopted for real-time computation on flight hardware. Similarly, GPOPS has been applied to a wide variety of optimal control problems [100–

102] as well. There are also open-source tools such as PSOPT [103] which implements pseudospectral methods, sparse nonlinear programming, automatic differentiation, and incorporates automatic scaling and mesh refinement facilities.

There are no similar generalized trajectory optimization frameworks that are based on indirect methods, primarily due the challenges in using indirect methods. In Ref. 104, a prototype indirect trajectory optimization framework was developed in MATLAB that used MATLAB's Symbolic Computing Toolbox for deriving the necessary conditions of optimality and *bvp4c* as the numerical BVP solver. This framework was further expanded in Ref. 30 to include a GPU-accelerated multiple shooting solver. These works form the starting point for the development of a generalized rapid trajectory optimization framework that leverages indirect methods. The fourth contribution in this dissertation is the development of an open source, expandable trajectory optimization framework with the goal of making indirect methods as accessible and easy-to-use as direct methods. This is discussed further in Chapter 5.

1.6 Outline

Chapter 2 explains the development of the Integrated Control Regularization Method (ICRM) for incorporating path constraints in optimal control problems when using indirect methods. This method is based on the regularization of path inequality constraints using saturation functions to convert them into equality constraints. The results of this method are validated using known optimal control problems by comparing them against the results obtained using either the Euler-Lagrange equations [12] or GPOPS [6]. ICRM is then used to perform some trade studies and analysis of a two-vehicle co-operative engagement scenario.

Chapter 3 details a new numerical method for solving two-point boundary value problems, called the Quasilinear Chebyshev Picard Iteration (QCPI). The algorithm is inherently parallel and is hence capable of leveraging parallel computing resources such as multi-core CPUs. The algorithm is implemented in the Python [105] programming language and accelerated using the *Numba* [106] Just-In-Time compiler to automatically parallelize the code and enable it to utilize multiple-core processors. The results are validated using some known optimal control problems. The performance of the solver is also benchmarked against the multiple shooting algorithm for a multi-vehicle trajectory optimization problem.

Chapter 4 uses the methods developed in the previous chapters to solve a representative large problem consisting of multiple vehicles and constraints. The scenario is based on the relatively simple kinematic model used in Ref. [107] and showcases some of the complex interactions and cross-coupling effects that appear in optimal trajectories of multiple vehicles. This chapter also demonstrates the capabilities of QCPI in solving large-dimensional nonlinear boundary value problems.

Chapter 5 describes the design and usage of an open source rapid trajectory optimization framework which uses indirect methods. The framework includes multiple numerical solvers and optimal control strategies and also has the potential for expansion by providing a flexible API. Initially implemented with a single shooting based solver and conventional optimal control theory, the framework is further expanded to include support for ICRM and QCPI detailed in the preceding chapters. All of the examples described in this dissertation are implemented using this framework.

Chapter 6 summarizes the contributions of this dissertation and details future work for improving QCPI, ICRM, and possible additions to the indirect trajectory optimization framework.

1.7 Contributions of Thesis

The goal of this dissertation is to advance the state-of-the-art in rapid trajectory optimization methods for the purpose of addressing large-scale optimal control problems using indirect methods. This section lists out the specific contributions made in this work.

A new method of handling path constraints in indirect methods using saturation functions and posing the BVP as a semi-explicit DAE is developed. The addition of path constraints to optimal control problems is considered a challenging task when using indirect methods. When deriving the necessary conditions of optimality, path constraints cause the trajectory to be split into a sequence of constrained and unconstrained. The order of these arcs have to be known a-priori for numerical solution of these multi-point boundary value problems. While using a continuation method to introduce constrained arcs one at a time is possible, it does not scale if the constraint is active multiple times or if there are multiple constraints. The Integrated Control Regularization Method (ICRM) allows the addition of one or more path constraints to an optimal control problem while maintaining the solution as a single arc. The method is designed so that the resulting BVP can be solved using existing numerical BVP solvers. The capabilities of ICRM are demonstrated by using it to analyze a constrained, two-vehicle optimal control problem.

A new numerical method for solving two-point boundary value problems is developed that is inherently parallel and is capable of leveraging parallel computing architectures. The new method, called the Quasilinear Chebyshev Picard Iteration (QCPI), uses Chebyshev polynomials for global approximation of the solution and can be applied to a wide range of nonlinear two-point boundary value problems. This is an expansion of the Modified Chebyshev Picard Iteration [31] algorithm which was only capable of addressing a limited class of BVPs. By using the Picard Iteration and the Modified Quasi-linearization Algorithm [108], the explicit computation of the sensitivity matrix is avoided. Only the partial derivatives of the boundary condition functions are computed while that of the dynamic equations are not required. The algorithm consists mostly of matrix-vector multiplication operations which can be parallelized very effectively. The performance of the algorithm is benchmarked against existing numerical methods using a multi-vehicle trajectory optimization problem.

The two methods developed in this dissertation is applied to a **large constrained multi-vehicle problem**. Trade studies are performed to analyze the effects of changing various problem parameters and constraints to illustrate the cross-coupled dynamics and non-intuitive trajectory changes that occur in such problems.

A generalized open source, rapid trajectory optimization framework using indirect methods is the third contribution of this dissertation. The optimization framework is capable of automatically deriving the necessary conditions of optimality without requiring any knowledge of optimal control theory on the part of the designer. The framework is also designed with a rich API that allows expansion with new optimal control algorithms and numerical methods and even allows implementation of direct methods, if required. The various features and design elements of this framework are discussed in detail.

2. INTEGRATED CONTROL REGULARIZATION METHOD (ICRM)

2.1 Overview

Handling of path-inequality constraints via optimal control theory is, in general, non-trivial. This is mitigated to a certain extent by using continuation and introducing constraints into the unconstrained problem one at a time [13]. Ref. 109 shows a method of regularizing bang-bang control problems with singular arcs into a two-point boundary value problem. Ref. 110 expands this concept further to apply to general optimal control problems with bounded control variables. However, this method based on trigonometry is not able to address path inequality constraints containing state variables. A more generalized method is presented in Ref. 111 by which state and control inequality constraints can be systematically incorporated into an optimal control problem while using indirect methods. Unlike in conventional optimal control theory, this method retains the trajectory as a single arc in the presence of multiple path constraints. This original formulation of regularized path constraints resulted in a semi-explicit differential algebraic equation formulation of the boundary value problem. This DAE-BVP required the use of a custom numerical solver due to extra algebraic equations adjoined to the BVP that do not have closed form solutions. In Ref. 111 a numerical solver based on collocation was developed in order to address this problem. This solver, as it is based on collocation, may not be suitable for parallelization and likely suffers from scalability issues as the problem size grows.

To overcome these challenges, a new way of formulating optimal control problems is proposed. In this method, the optimal control law is obtained, not by analytically solving algebraic equations in the traditional manner, but by adjoining the algebraic equations to the BVP to form an extended BVP with extra differential equations representing the controls. This allows the control variables to be numerically integrated rather than directly obtained from closed form control laws. This is also called index-reduction of differential algebraic equations [112,113] and it comes with its own set of challenges such as some numerical difficulties as described in Section 2.3. However, as illustrated in the later parts of this chapter, the method is still capable of solving non-trivial constrained optimal control problems including some hypersonic trajectory problems. While differentiating an existing control equation and numerically integrating it may seem counter-intuitive, it greatly simplifies the process of solving complicated optimal control problems as described in later sections.

The use of saturation functions to regularize path constraints and numerically integrating the control greatly changes and simplifies the general approach to solving large-scale trajectory optimization problems with path constraints. This method is termed the Integrated Control Regularization Method (ICRM).

2.2 Regularization of Path Inequality and Control Constraints

In the original work, it was shown that path constraints can be regularized by representing them as smooth saturation functions. While this method is closely related to how a path constraint can be implemented using a penalty function, Ref. 111 shows that by using saturation functions it is possible to ensure that the constraints are never violated during the numerical solution process. This intrinsic property of the saturation function method guarantees that the numerical solver is never working with an infeasible solution. As an example, consider a general optimal control problem with a path constraint as follows:

$$\begin{aligned}\text{Min } J = \phi(\boldsymbol{x}(T), T) + \int_0^T L(t, \boldsymbol{x}, \boldsymbol{u})\, dt \\ \dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}, \mathbf{u}), \qquad \Phi(\boldsymbol{x}(0), \boldsymbol{x}(T)) = 0 \\ S_i(\boldsymbol{x}) \in [S_i^-, S_i^+], \qquad i = 1 \dots p\end{aligned} \tag{2.1}$$

where $S_i(\boldsymbol{x})$ represents the path constraints in the problem. The constraint is replaced by a suitable saturation function, ψ:

$$S_i(\boldsymbol{x}) = \psi(\xi_{i,1}), \qquad i = 1 \dots p \tag{2.2}$$

Eq. (2.2) is successively differentiated w.r.t time, and new coordinates, $\dot{\xi}_{i,j} = \xi_{i,j+1}$, are introduced, until a control variable appears. For example, if $S_i(\boldsymbol{x})$ is of order 2:

$$\begin{aligned} S_i^{(1)}(\boldsymbol{x}) &= \psi\,\dot{\xi}_{i,1} && := h_1, \quad \dot{\xi}_{i,1} = \xi_{i,2} \\ S_i^{(2)}(\boldsymbol{x}) &= \psi\;\;\xi_{i,2}^2 + \psi\,\dot{\xi}_{i,2} && := h_2, \quad \dot{\xi}_{i,2} = u_{ei} \end{aligned} \tag{2.3}$$

Finally, an equality constraint, $S_i^{(q)}(\boldsymbol{x}) - h_q = 0$ is also added to the problem along with an added term ${}_0^T\, \epsilon u_{ei}^2 dt$ to the path cost, where u_{ei} is a new control variable, $\xi_{i,j}$ are new state variables, and q is the order of the constraint.

The resulting extended optimal control problem (OCP) which incorporates the path constraints is stated as:

$$\text{Min } J = \Phi(\boldsymbol{x}(T), T) + {}_0^T\; L(t, \boldsymbol{x}, \boldsymbol{u}) \;\; + \epsilon u_{ei}^2 dt \tag{2.4a}$$

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}, \mathbf{u}) \tag{2.4b}$$

$$\dot{\xi}_{i,1} = \xi_{i,2} \tag{2.4c}$$

$$\dot{\xi}_{i,2} = u_{ei} \tag{2.4d}$$

$$h_2(t) - S_i^{(2)}(\boldsymbol{x}(t)) = 0 \tag{2.4e}$$

$$\Phi(\boldsymbol{x}(0), \boldsymbol{x}(T)) = 0 \tag{2.4f}$$

$$S_i\left(\mathbf{0}, \mathbf{x}(\mathbf{0})\right) - \psi(\xi(0)) = 0 \tag{2.4g}$$

$$S_i^{(1)}\left(\mathbf{0}, \mathbf{x}(\mathbf{0})\right) - h_1(0) = 0 \tag{2.4h}$$

Applying optimal control theory to the extended OCP in Eq. (2.4), the Hamiltonian is defined as:

$$H = \boldsymbol{\lambda}^T \boldsymbol{f} + L + \epsilon u_{ei}^2 + \eta_{i,1}\dot{\xi}_{i,1} + \eta_{i,2}\dot{\xi}_{i,2} + \mu_{i,1}\;\; h_2 - S_i^{(2)}(\boldsymbol{x}) \tag{2.5}$$

where $\boldsymbol{\lambda}$ is the vector of costates corresponding to the states $\boldsymbol{x}$, $\eta_{i,1}$ and $\eta_{i,2}$ are the costates corresponding to $\xi_{i,1}$ and $\xi_{i,2}$, and $\mu_{i,1}$ is the Lagrange multiplier used to adjoin the equality constraint to the Hamiltonian. The extended boundary value problem is then formulated, consisting of the differential equations in Eq. (2.6), the boundary conditions in Eq. (2.7), and the algebraic conditions in Eq. (2.8). Combined, these three sets of equations constitute a differential algebraic equation boundary value problem (DAE-BVP).

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}, \mathbf{u}) \tag{2.6a}$$

$$\dot{\xi}_{i,1} = \xi_{i,2} \tag{2.6b}$$

$$\dot{\xi}_{i,2} = u_{ei} \tag{2.6c}$$

$$\dot{\boldsymbol{\lambda}} = -\frac{\partial H}{\partial \boldsymbol{x}} \tag{2.6d}$$

$$\dot{\eta}_{i,1} = -\frac{\partial H}{\partial \xi_{i,1}} \tag{2.6e}$$

$$\dot{\eta}_{i,2} = -\frac{\partial H}{\partial \xi_{i,2}} \tag{2.6f}$$

$$\Phi(\boldsymbol{x}(0), \boldsymbol{x}(T)) = 0 \tag{2.7a}$$

$$\boldsymbol{\lambda}(T) = \frac{\partial \Phi}{\partial \boldsymbol{x}} \tag{2.7b}$$

$$\eta_{i,1}(T) = 0 \tag{2.7c}$$

$$\eta_{i,2}(T) = 0 \tag{2.7d}$$

$$S_i\left(\boldsymbol{x}(0)\right) - \psi(\xi(0)) = 0 \tag{2.7e}$$

$$S_i^{(1)}(\boldsymbol{x}(0)) - h_1 = 0 \tag{2.7f}$$

$$\frac{\partial H}{\partial \boldsymbol{u}} = 0, \quad \frac{\partial H}{\partial u_{ei}} = 0, \quad \frac{\partial H}{\partial \mu_{i,2}} = 0 \tag{2.8}$$

The process described above is repeated and extra states, controls, and Lagrange multipliers are added for each constraint $S_i(\boldsymbol{x})$. It is to be noted that the process of taking derivatives of the constraint until the control appears (Eq. (2.3)) closely mirrors the procedure in optimal control theory as detailed in Section 1.2.3. However, instead of inserting a new arc with boundary conditions for every instance of a constraint, the BVP is extended with extra states and controls.

In conventional optimal control theory, especially for unconstrained problems, the algebraic control equations are usually invertible and can be solved for the control variables, $\boldsymbol{u}$, thereby eliminating them from the problem. There are cases where this is not possible in which case a computationally intensive numerical root-solving method may be attempted [15]. The resulting ODE-BVP is then solved using numerical methods such as shooting or collocation. In the case of ICRM, the presence of saturation functions renders some of these algebraic equations transcendental making a closed-form solution impossible. In Ref. 111, a special numerical solver based on collocation was designed to solve these kinds of DAE-BVPs. In this dissertation, a method for converting these DAE-BVPs into ODE-BVPs is pursued in order to be able to solve these problems using existing numerical methods. This is especially important for scaling as it will then be possible to leverage existing parallel numerical solvers for this purpose.

2.3 Differential Algebraic Equations

Differential algebraic equations are equations of the following form:

$$\boldsymbol{f}(\boldsymbol{x}\ ,\boldsymbol{x},t)=0 \tag{2.9}$$

If $\partial F/\partial \boldsymbol{x}\ $ is non-singular, the DAE can be converted into an explicit ODE function that can be solved using ODE solvers. In this case, Eq (2.9) is defined as having an index of zero and can be called an implicit ODE. If $\partial F/\partial \boldsymbol{x}\ $ is singular, F can be differentiated w.r.t t until it is possible to find a solution for $\boldsymbol{x}\ $ as a function of $\boldsymbol{x}$ and t. The number of derivatives required to obtain such a solution is called the index of the DAE. Solving DAEs of high index is a non-trivial task and has been extensively examined in the literature [114–119].

In optimal control problems, a special form of DAEs are encountered:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t) \tag{2.10a}$$

$$0 = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}, t) \tag{2.10b}$$

where $\boldsymbol{x} \in R^{n1}$ and $\boldsymbol{u} \in R^{n2}$ and are called the **differential variable** and the **algebraic variable** of the DAE respectively. This is called the semi-explicit form of DAEs [120]. Semi-explicit DAEs show up in many engineering models with applications ranging from process engineering and mechanical engineering to electrical engineering. $\boldsymbol{f}$ may denote the differential equations for the dynamics of the system while $\boldsymbol{g}$ may denote system invariants such as conservation of energy, charge, etc. In the case of optimal control problems, $\boldsymbol{f}$ will consist of the dynamic equations for the states and costates while $\boldsymbol{g}$ represents the control laws and equality constraints.

References 121, 122, and 123 discuss numerical methods for solving semi-explicit DAEs. In some cases, $\boldsymbol{g}$ can be analytically inverted in order to eliminate $\boldsymbol{u}$ from the system (DAE of index 0). However, when there are custom numerical functions involved in the problem definition or if $\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}, t) = 0$ contains transcendental equations, a closed form solution is no longer possible. This means that the DAE then has an index greater than zero. In fact, this usually happens when ICRM is applied to an optimal control problem with path constraints.

The index of the DAE in Eq. (2.10) can be reduced by one by differentiating it w.r.t t [120]. This is a method that is often used for solving problems of this form.

$$\boldsymbol{g}_{\boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{u}, t)\dot{\boldsymbol{x}} + \boldsymbol{g}_{\boldsymbol{u}}(\boldsymbol{x}, \boldsymbol{u}, t)\dot{\boldsymbol{u}} + \boldsymbol{g}_t = 0 \tag{2.11}$$

If $\boldsymbol{g}_{\boldsymbol{u}}$ is non-singular, it is possible to obtain a closed form solution for $\dot{\boldsymbol{u}}$, and the DAE system is converted into an explicit ODE system. The caveat here is that this system requires consistent initial values for u for the numerical solution process to be stable. Ref. 117 details a method for maintaining the consistency of the solution while using a Backward Differentiation Formula (BDF) method to solve the DAE. Ref. 112 examines the numerical difficulties that may be encountered when this strategy is used for solving DAEs. Some of the main challenges cited in Ref. 112 are when the problem is of nearly higher index, i.e, when $\boldsymbol{g}$ is sometimes singular and when there are fast-changing transients in $\boldsymbol{g}$. Some strategies for overcoming these challenges for certain types of DAEs are detailed in Refs 117, 124–126. These strategies involve using known system invariants as a way to stabilize the numerical method and may not be applicable as a universal answer to these issues.

From the point of view of the numerical solution, it is desirable for the DAE to have an index which is as small as possible. As we have seen, a reduction of the index can be achieved by differentiating the constraints. As such, in this dissertation we use index-reduction by differentiation as described by Gear [120] to convert the DAEs into explicit ODEs which can be solved using existing numerical solvers. As shown in the later sections in this chapter, even with the possible numerical difficulties, this method is capable of solving complex optimal control problems.

2.4 Numerically Integrated Optimal Control Law

As described in the previous sections, when using indirect methods, an optimal control problem is first converted into a DAE-BVP. This boundary value problem has differential and algebraic conditions of the form in Eq. (2.12) where the control law equations, $\frac{\partial H}{\partial \boldsymbol{u}} = 0$ form the $\boldsymbol{g}$ function.

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t) \tag{2.12a}$$

$$0 = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}, t) \tag{2.12b}$$

$$0 = \boldsymbol{b}(\boldsymbol{x}(0), \boldsymbol{x}(T)) \tag{2.12c}$$

A closed-form solution for the control variables, $\boldsymbol{u}$, is then obtained, if possible, by solving this equation, thereby eliminating them from the overall problem to obtain a BVP of the form shown in Eq. (2.13). However, the control law equation may have multiple solutions, requiring further effort to determine which option is the optimal one. In such a scenario, Pontryagin's Minimum Principle [11] can be used to select the solution that minimizes the Hamiltonian.

$$\frac{dx}{dt} = \boldsymbol{f}(t, \boldsymbol{x}), \ \boldsymbol{b}(\boldsymbol{x}(0), \boldsymbol{x}(T)) = 0 \tag{2.13}$$

The application of saturation functions for regularization of path constraints results in a control law consisting of transcendental equations that generally have no closed form solution. The index-reduction strategy described in Eq. (2.11) is then used to formulate a new BVP in which dynamic equations are derived for the control variables. This process is detailed in Eq. (2.14).

$$\begin{aligned} 0 &= \partial_x \boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{u})\dot{\boldsymbol{x}} + \partial_u \boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{u})\dot{\boldsymbol{u}} + \partial_t \boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{u}) \\ \implies \partial_u \boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{u})\dot{\boldsymbol{u}} &= -\partial_x \boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{u})\dot{\boldsymbol{x}} - \partial_t \boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{u}) \end{aligned} \tag{2.14}$$

The original algebraic constraint in Eq. (2.12b) is then added as a boundary condition at either the initial or terminal point to obtain a well-formed two-point boundary value problem in Eq. (2.15). It is very important when using this method that the initial values chosen for the control variables are consistent and satisfy the original control law. Starting with inconsistent values for the control variables can result in significant numerical instabilities when solving the BVP.

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u}) \tag{2.15a}$$

$$\partial_u \boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{u})\dot{\boldsymbol{u}} = -\partial_x \boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{u})\dot{\boldsymbol{x}} - \partial_t \boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{u}) \tag{2.15b}$$

$$\boldsymbol{g}(t_0, \boldsymbol{x}(t_0), \boldsymbol{u}(t_0)) = 0 \tag{2.15c}$$

$$\boldsymbol{b}(\boldsymbol{x}(0), \boldsymbol{x}(T)) = 0 \tag{2.15d}$$

It is assumed that $\partial_{\boldsymbol{u}} \boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{u})$ is always non-singular for the entirety of the optimal trajectory. ICRM in its current form can only be applied for problems in which this condition holds. If $\partial_{\boldsymbol{u}} \boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{u})$ is singular, the DAE is of a higher index and requires more sophisticated numerical methods for obtaining a solution.

2.5 Candidate Saturation Functions

There are many functions that can be used as saturation functions for path constraint regularization. Some of the common examples are the sigmoid, arc-tangent, and hyperbolic tangent functions. Ref. 111 suggests the following saturation functions as candidates for path constraint regularization depending on the type of constraint.

For one-sided constraints with an upper bound ($S(\boldsymbol{x}) \leq S^+$) or a lower bound ($S(\boldsymbol{x}) \geq S^-$), Eqs. (2.16) or (2.17) respectively may be used as saturation functions. These one-sided saturation functions are illustrated in Figure 2.1.

$$\psi(\xi) = S^+ - \exp(-\xi) \tag{2.16}$$

$$\psi(\xi) = S^- + \exp(\xi) \tag{2.17}$$

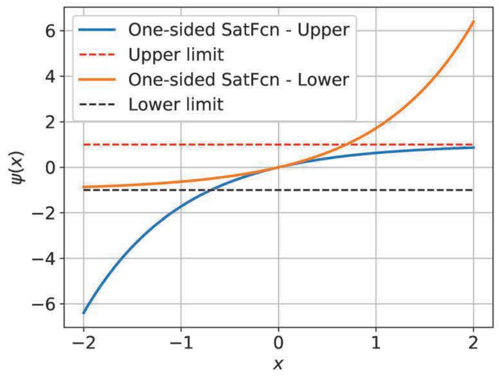


Figure 2.1. One-sided Saturation Functions

For two-sided constraint, such as $S^- \leq S(x) \leq S^+$, the following function based on the sigmoid function is suggested by the authors of Ref. 111.

$$\psi(\xi) = S^+ - \frac{S^+ - S^-}{1 + \exp(m\xi)}, \qquad m = \frac{4}{S^+ - S^-} \tag{2.18}$$

Eq. (2.18) is formulated so that the resulting function has a slope of one at $\xi = 0$. The two-sided saturation function is illustrated in Figure 2.2. From further analysis, it was found that these saturation functions are well behaved only if the constraint functions are normalized to have limits of ± 1 as discussed in the next section.

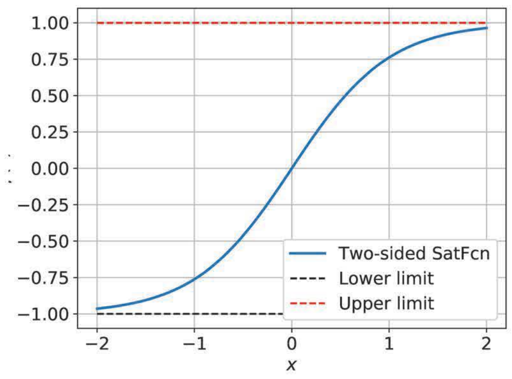


Figure 2.2. Two-sided Saturation Functions

2.6 Constraint Normalization

The nature of the saturation functions used requires that the constraint functions be normalized to have limits of ± 1 before they can be regularized. If the constraints are not normalized, the saturation functions have to be modified on a case-by-case basis to change their rate of "switching" to achieve consistent results during regularization.

For one-sided constraints and two-sided constraints with symmetric limits, normalization can be easily accomplished by dividing the constraint expression by the constraint limit as shown in Eqs. (2.19). Two-sided constraints with asymmetric limits will have to be split into separate one-sided constraints and normalized before regularization can be applied.

$$S(x) \leq S^{+} \Rightarrow \frac{S(x)}{S^{+}} \leq 1 \tag{2.19a}$$

$$S(x) \geq S^{-} \Rightarrow \frac{S(x)}{S^{-}} \geq 1 \tag{2.19b}$$

$$-S \leq S(x) \leq S \Rightarrow -1 \leq \frac{S(x)}{S} \leq 1 \tag{2.19c}$$

2.7 Regularization Parameter, , and the Push-Off Factor

As discussed before in Section 2.2, the regularization parameter, , is used to append an extra term to the path-cost for each path constraint added to the problem. This term, $\epsilon u_e^2 \, dt$, converts the original problem into a multi-objective optimal control problem. forms the weighting factor in the objective functional for the magnitude of the extra control variable, u_e, added during constraint regularization. Depending on the order of the constraint, this control variable either directly influences the value of the saturation function representing the constraint or affects a derivative of the saturation function. For larger values of , reduction of u_e will have a higher weightage in the path-cost, and therefore the optimal solution will have a smaller value for u_e. This causes peak-value of the saturation function to be pushed further from its asymptotes, and therefore the effective constraint limit becomes smaller than its desired value. This creates a push-off factor between the optimal trajectory and the constraint. The relationship between the magnitude of this push-off factor and is highly problem dependent based on the relative magnitudes of the path constraint and the original cost functional. It is to be noted that this "push-off factor" is *not* a tunable parameter, but rather, an outcome of using ICRM.

As is made smaller, the optimal solution generates larger values for the augmented control, u_e, which results in the saturation function getting closer to its asymptotic limits. If is reduced to zero, u_e tends to infinity which corresponds to the multi-arc solution from conventional optimal control theory. While decreasing

u_e allows the solution to very closely approach the constraint, this comes with the additional challenge of vanishing gradients as the saturation function approaches its limit. This also makes the problem numerically sensitive at the points where the constraint is active leading to difficulties when solving the BVP. The addition of extra nodes at these locations may significantly increase the computation time even in the case of BVP solvers capable of handling such numerically sensitive regions.

In many of the examples that follow in this dissertation, a constrained optimal control problem is initially solved with a relatively large value of , followed by a reduction in   using a continuation methodology. The push-off factor causes the constraints to be more restrictive initially and decreasing   results in a "relaxation" of these constraints. For example, in the case of a control constraint, a larger value of   restricts the effective control authority available resulting in sub-optimal results. Decreasing   releases this artificial restriction allowing the solution to more closely approach the desired constraints.

When a problem has multiple constraints, changing the value of   for one constraint may affect the push-off factors for the other constraints simultaneously. This is because the relative magnitudes of the terms of the cost functional changes when one   is modified, thereby changing the optimal value of u_e for every constraint. An example of this phenomenon can be seen in Section 2.8.4.

Choosing a feasible value for   for each constraint is required in order to use ICRM. If the chosen value of   is too large, the effective "cushion" around the constraint limit might make the problem infeasible. Conversely, if the initial value of   is too small, the problem may become too numerically sensitive and difficult to solve. The value of   is currently chosen by trial and error because its effect on the solution is highly problem-specific. A possible strategy for automatic selection of   is discussed in Section 6.2.1.

2.8 Validation

Some representative constrained trajectory optimization problems were solved using ICRM, and the results were validated by solving the same using conventional optimal control theory or using GPOPS. The necessary conditions of optimality for a modified version of the Brachistochrone problem with a path constraint are derived for both ICRM as well as conventional optimal control theory to demonstrate the simplification of the process. The control histories and trajectories are compared to validate that the ICRM solution is very close to the optimal solution. The same is done for some other representative problems, namely the one-dimensional free-flight problem with a control constraint and a hypersonic trajectory problem with a heat-rate and control constraint.

2.8.1 Constrained Brachistochrone Problem

The first validation problem is a modified version of the classical Brachistochrone problem with a linear path constraint. The constraint is positioned such that the optimal trajectory is restricted from going as far down the y direction as it does in the unconstrained case. The optimal constrained trajectory follows the constraint where required and then goes back to a time-optimal path. The results are validated by solving the same problem using conventional optimal control theory formulating it as a multi-point boundary value problem and comparing the solutions.

Problem Statement:

$$\text{Min } T \tag{2.20a}$$

Subject to :

$$\dot{x} = v\cos\theta \tag{2.20b}$$

$$\dot{y} = v\sin\theta \tag{2.20c}$$

$$\dot{v} = g\sin\theta \tag{2.20d}$$

$$x(0) = y(0) = 0, x(T) = -y(T) = 10 \tag{2.20e}$$

$$g = -9.81 \tag{2.20f}$$

$$x + y \geq -1.0 \tag{2.20g}$$

where θ is the control.

Necessary Conditions of Optimality - MPBVP

Applying conventional optimal control theory (Section 1.2.3), the constrained optimal control problem is posed as a multi-point boundary value problem (MPBVP). The necessary conditions of optimality have to be derived for two cases – when the constraint is inactive ($x+y > -1.0$) and when the constraint is active ($x+y = -1.0$). When the constraint is inactive, the Hamiltonian is defined as:

$$H = \lambda_x v\cos\theta + \lambda_y v\sin\theta + \lambda_v g\sin\theta + 1.0 \tag{2.21}$$

where λ_x, λ_y and λ_v are costates. The dynamic equations for the costates are defined as

$$\dot{\lambda_x} = -\frac{\partial H}{\partial x} = 0 \tag{2.22a}$$
$$\dot{\lambda_y} = -\frac{\partial H}{\partial x} = 0 \tag{2.22b}$$
$$\dot{\lambda_v} = -\frac{\partial H}{\partial x} = -\lambda_x \cos\theta - \lambda_y \sin\theta \tag{2.22c}$$
$$\tag{2.22d}$$

The control law is found by solving the following equation:

$$\frac{\partial H}{\partial \theta} = g\lambda_v \cos\theta - \lambda_x v \sin\theta + \lambda_y v \cos\theta = 0 \tag{2.23a}$$
$$\implies \theta = -2\,\mathrm{atan} \quad \frac{1}{g\lambda_v + \lambda_y v} \quad \lambda_x v \pm \quad \overline{g^2\lambda_v^2 + 2g\lambda_v\lambda_y v + \lambda_x^2 v^2 + \lambda_y^2 v^2} \tag{2.23b}$$

Since there are two control options, the optimal value is found by evaluating the Hamiltonian and choosing the one which gives the minimum value (Pontryagin's Minimum Principle). The only added boundary conditions are $\lambda_v(T) = 0$, since $v(T)$ is unconstrained and $H(T) = 0$ since the T is unconstrained.

When the constraint is active, the Hamiltonian is different, and, consequently, the costates have different dynamics and the control law is different. In order to derive these conditions, first, the path constraint is differentiated w.r.t time until the control variable θ appears.

$$\frac{d \;\, x + y - 1.0)}{dt} = \frac{dx}{dt} + \frac{dy}{dt} = v\cos\theta + v\sin\theta \tag{2.24}$$

In the case of Eq. (2.20g), the control appears when the expression is differentiated once, making it a path constraint of order 1. The Hamiltonian is augmented with this differentiated constraint using a Lagrange multiplier, μ_1, to obtain the Hamiltonian for the constrained arc, H_c.

$$H_c = \lambda_x v \cos\theta + \lambda_y v \sin\theta + \lambda_v g \sin\theta + 1.0 + \mu_1 \left(v\cos\theta + v\sin\theta\right) \tag{2.25}$$

The costate equations for the constrained arc are obtained as:

$$\dot{\lambda}_x = 0 \tag{2.26a}$$

$$\dot{\lambda}_y = 0 \tag{2.26b}$$

$$\dot{\lambda}_v = -\lambda_x \cos\theta - \lambda_y \sin\theta - \mu_1 (\sin\theta + \cos\theta) \tag{2.26c}$$

$$\tag{2.26d}$$

The constrained control law includes expressions for both θ and the Lagrange multiplier, μ_1. The constrained control law is found to have two solutions as well. The optimal value of θ is found to be equivalent to the slope of the constraint function.

$$\theta = \frac{\pi}{4} \pm \frac{\pi}{2} \tag{2.27a}$$

$$\mu_1 = -\frac{1}{2v}\left(g\lambda_v + v\left(\lambda_x + \lambda_y\right)\right) \tag{2.27b}$$

The entry and exit conditions for the constrained arc referenced in Eqs (1.17)-(1.21) are calculated as shown below. t_1 refers to the junction at the end of the constrained arc. π_0 is a Lagrange multiplier used to adjoin the costate corner conditions as discussed in the previous chapter in Eq. (1.23). It becomes an extra free parameter that is to be solved for along with the BVP.

$$x(t_1^-) = x(t_1^+), \quad y(t_1^-) = y(t_1^+), \quad v(t_1^-) = v(t_1^+) \tag{2.28a}$$

$$x(t_1^+) + y(t_1^+) + 1.0 = 0 \tag{2.28b}$$

$$\lambda_x(t_1^-) = \lambda_x(t_1^+) + \pi_0, \quad \lambda_y(t_1^-) = \lambda_y(t_1^+) + \pi_0, \quad \lambda_v(t_1^-) = \lambda_v(t_1^+) \tag{2.28c}$$

$$H(t_1^-) = H_c(t_1^+) \tag{2.28d}$$

$$x(t_2^-) = x(t_2^+), \quad y(t_2^-) = y(t_2^+), \quad v(t_2^-) = v(t_2^+) \tag{2.28e}$$

$$\lambda_x(t_2^-) = \lambda_x(t_2^+), \quad \lambda_y(t_2^-) = \lambda_y(t_2^+), \quad \lambda_v(t_2^-) = \lambda_v(t_2^+) \tag{2.28f}$$

$$H_c(t_2^-) = H(t_2^+) \tag{2.28g}$$

This Multi-Point Boundary Value Problem (MPBVP) is solved using the multiple shooting method. The result is compared to that obtained using ICRM in the upcoming sections. It is to be noted that in this particular problem, the constraint is only active once, leading to a single constrained arc bookended by two unconstrained arcs. However, in more general problems, the constraint may be active more than once, leading to the conditions such as the ones in Eq. (2.28) being repeated at every entry/exit junction to each constrained arc and making the MPBVP larger and more difficult to solve.

Necessary Conditions of Optimality - ICRM

By applying ICRM, the path constraint was incorporated into the problem without the complexities mentioned in Section 1.2.3. Following the steps in Section 2.2, the optimal control problem is augmented with an extra state, ξ_1 and one extra control, u_{e1}. The dynamics of this new state is defined as:

$$\dot{\xi}_1 = u_{e1} \tag{2.29}$$

Using the one-sided saturated function from Eq. (2.17) and the smoothing factor $_1$ the Hamiltonian is defined as follows

$$\begin{aligned} H = {}_1u_{e1}^2 + g\lambda_v \sin\theta + \lambda_x v\cos\theta + \lambda_{\xi_1} u_{e1} + \lambda_y v \sin\theta \\ + \mu_1 \left(-\exp\left(\xi_1\right)\, u_{e1} + v\sin\theta + v\cos\theta\right) + 1 \end{aligned} \tag{2.30}$$

From this augmented Hamiltonian, the costate dynamics are derived as:

$$\dot{\lambda}_x = 0 \tag{2.31a}$$

$$\dot{\lambda}_y = 0 \tag{2.31b}$$

$$\dot{\lambda}_v = -\lambda_x \cos\theta - \lambda_y \sin\theta - \mu_1 \left(-\sin\theta - \cos\theta\right) \tag{2.31c}$$

$$\dot{\lambda}_{\xi_1} = \mu_1 u_{e1} \exp\xi_1 \tag{2.31d}$$

The differential equations for the control variables are derived as shown in Section 2.4. The actual equations are not listed here for brevity. The boundary conditions for the system including the new state, costates, and the control variable ‘states’ are defined as:

$$x(0) = y(0) = 0 \tag{2.32a}$$

$$\lambda_{\xi_1}(0) = 0 \tag{2.32b}$$

$$x(T) = -y(T) = 10 \tag{2.32c}$$

$$\lambda_v(T) = 0 \tag{2.32d}$$

$$\exp \xi_1 + 1.0 + x(T) + y(T) = 0 \tag{2.32e}$$

$$H(T) = 0 \tag{2.32f}$$

$$g\lambda_v \cos(\theta) - \lambda_x v \sin(\theta) + \lambda_y v \cos(\theta) + \mu_1 \left(-v \sin(\theta) + v \cos(\theta)\right) \;_{t=T} = 0 \tag{2.32g}$$

$$2 \;_1 u_{e1} + \lambda_{\xi_1} - \mu_1 \exp \xi_1 \;_{t=T} = 0 \tag{2.32h}$$

$$-\exp \xi_1 \; u_{e1} + v \sin(\theta) + v \cos(\theta) \;_{t=T} = 0 \tag{2.32i}$$

The boundary conditions and the differential equations derived above are the only ones that are required to enforce the linear path constraint of the problem, regardless of how many times the constraint is active or inactive throughout the trajectory.

Analysis

The constrained Brachistochrone problem is solved using both the MPBVP and ICRM methods using a multiple-shooting algorithm, and the results are compared in this section. The homotopy continuation method described in Ref. 13 is used to solve the MPBVP version of the problem. First, the Brachistochrone problem is solved without any path constraints. After this, a short constrained arc is introduced at the point with the highest constraint violation by adjusting the constraint limit. The constraint limit is then changed back to its original value in a series of continuation steps to obtain the final solution. This results of this process is illustrated in Figures 2.3 and 2.4. It can be seen from the control history plot that the constrained arc is a distinctly separate horizontal line compared to the two unconstrained arcs at the beginning and end of the trajectory.

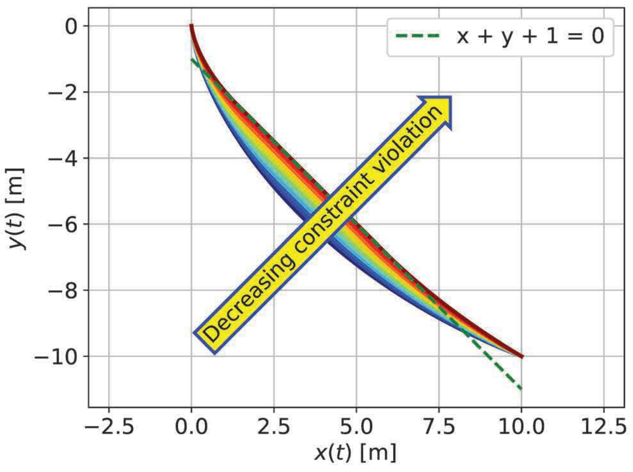


Figure 2.3. Constrained Brachistochrone - MPBVP Continuation - Trajectory

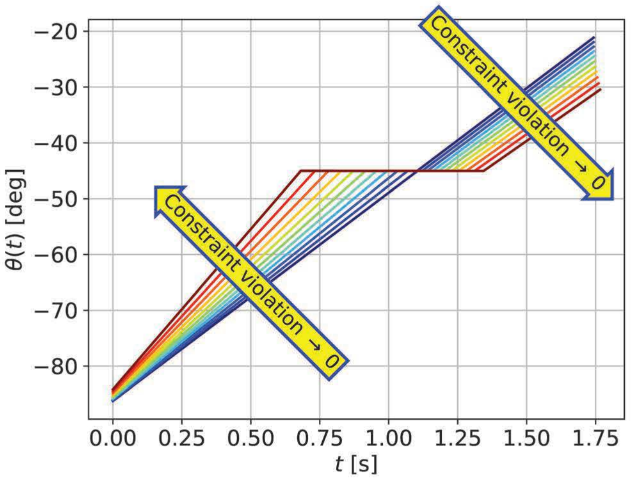


Figure 2.4. Constrained Brachistochrone - MPBVP Continuation - Control History

Unlike the MPBVP solution, when using ICRM, the trajectory remains a single arc that never violates the path constraint. The solution obtained using ICRM with $\epsilon_1 = 10^{-5}$ is shown in Figure 2.5 and the corresponding optimal control is shown in Figure 2.6. Both figures also show how the ICRM solution compares to the MPBVP solution. It can be seen that the actual trajectories practically overlap, while the control history is very close to what the MPBVP method predicts.

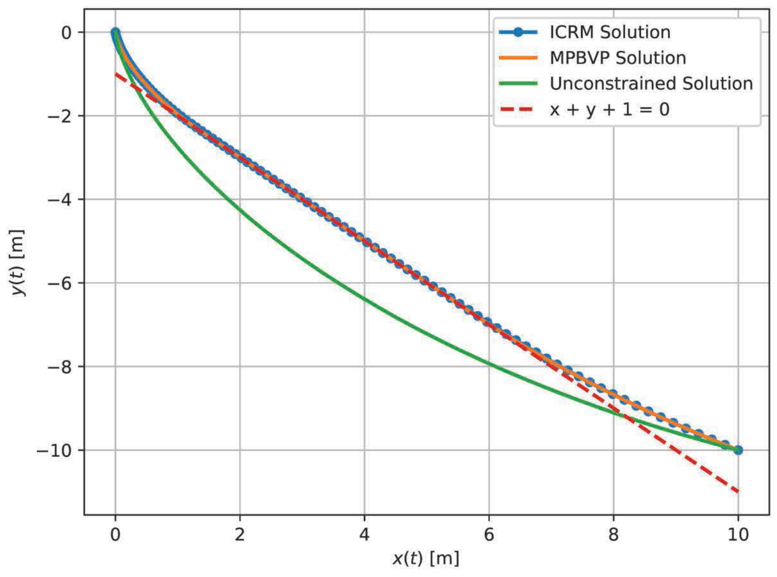


Figure 2.5. Constrained Brachistochrone Solution - Trajectory

The results in Figures 2.5 and 2.6 show that ICRM gives a solution that is very close to the optimal solution while avoiding the difficulties of solving multi-point boundary value problems. ICRM also retains the structure of a general two-point boundary value problem which can be solved by most numerical BVP solvers without requiring any special modifications for processing extra algebraic constraints.

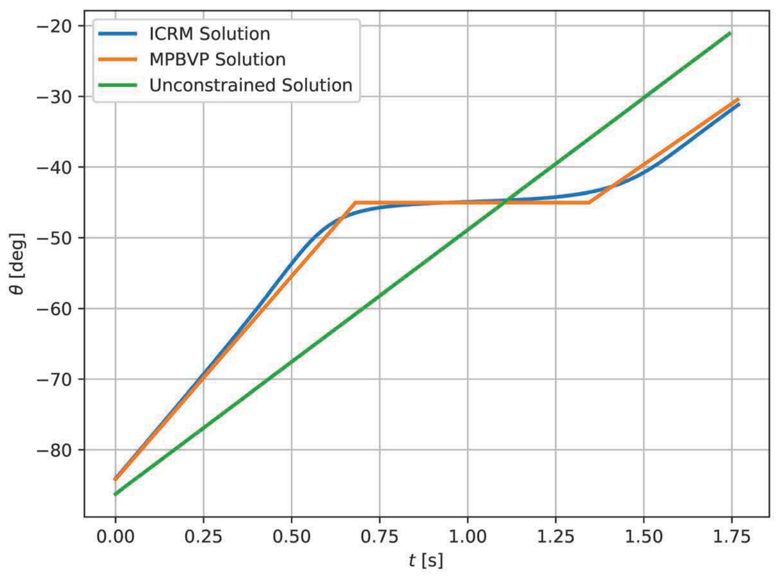


Figure 2.6. Constrained Brachistochrone Solution - Control History

Costate Discrepancies

When a constrained optimal control problem is solved using the MPBVP method, the costates may have jump conditions in them at the entry and/or exit junctions of the constrained arcs (Eq. (1.20)). In case of the constrained Brachistochrone problem, these jump conditions appear in λ_x and λ_y. From the costate profiles in Figure 2.7, it can be seen that these jump conditions *do not* appear when using ICRM. Even so, the control and the state trajectories still closely match the MPBVP solution as shown in Figures 2.5 and 2.6. The discrepancy is due to the fact that the control law equation is different in the case of ICRM and incorporates more variables, u_{e1}, ξ_1, ϵ_1 and μ_1.

For additional validation that the costates are correct, the regularized optimal control problem with the added state variable ξ_1 and control, u_{e1} is solved using the direct solver GPOPS [6]. GPOPS is able to estimate the costates corresponding to the states using the Covector Mapping Theorem [48]. These results are compared

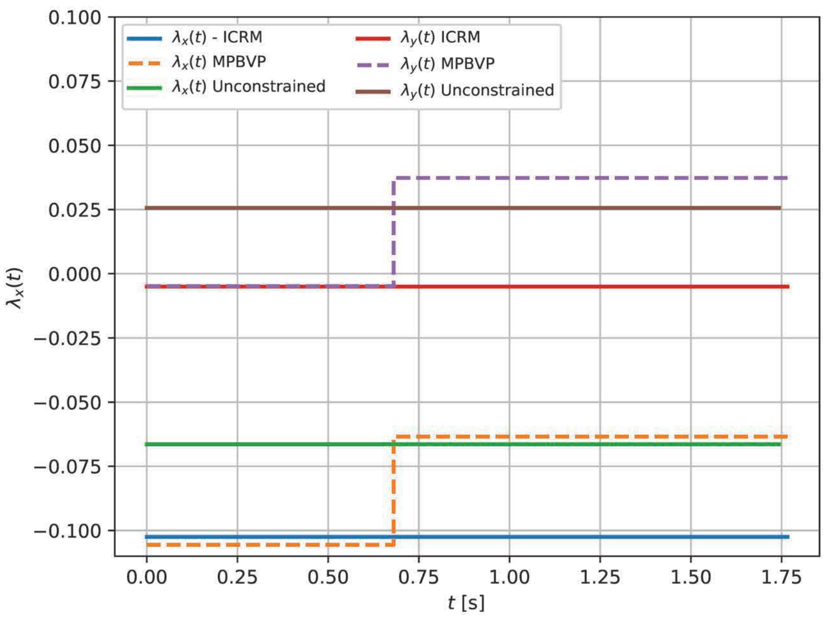


Figure 2.7. Constrained Brachistochrone Solution - Costates

to the costate trajectories from ICRM in Fig 2.8. It can be seen that the costates estimated by GPOPS matches very well to the results obtained using ICRM.

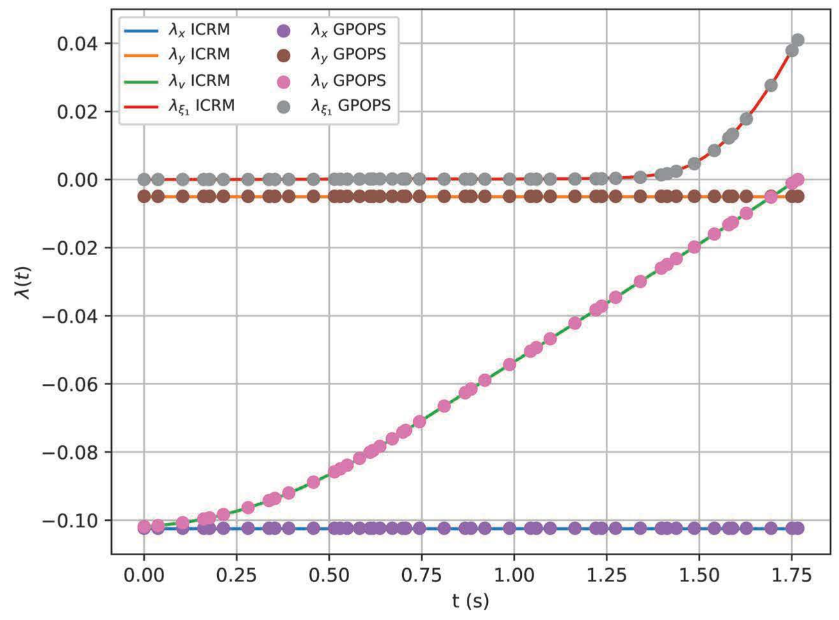


Figure 2.8. Constrained Brachistochrone - Costate Comparison with GPOPS

2.8.2 One-Dimensional Free-Flight Problem with Control Constraint

This problem scenario consists of a point-mass with a single thruster that can provide a specific amount of acceleration. The motion of the body is constrained to be along a single axis. The thrust has a finite magnitude and is implemented as a control limit using ICRM. The objective is to move the body from one point to another in the minimum time possible using the thruster. The dynamics model of this problem is a simple double integrator for which a closed-form analytical solution can be found as demonstrated in the next section.

Problem Statement:

$$\text{Min } T \tag{2.33a}$$

Subject to :

$$\dot{x} = v \tag{2.33b}$$

$$\dot{v} = u \tag{2.33c}$$

$$x(0) = v(0) = v(1) = 0 \tag{2.33d}$$

$$x(1) = 1 \tag{2.33e}$$

$$|u| \leq l \tag{2.33f}$$

where u is the control.

Analytical Solution

The Hamiltonian can be defined as $H = \lambda_x v + \lambda_v u + 1$, and the costate rates are defined as:

$$\dot{\lambda}_x = -\frac{\partial H}{\partial x} = 0 \tag{2.34}$$

$$\dot{\lambda}_v = -\frac{\partial H}{\partial v} = -\lambda_x \tag{2.35}$$

Since the problem is linear in control, the optimal solution is a bang-bang control according to Pontryagin's Minimum Principle(PMP) [11]. According to PMP, the control law in such a problem is dictated by a switching function, $\frac{\partial H}{\partial u}$. The control law is defined as:

$$u = \begin{cases} -1 & \frac{\partial H}{\partial u} > 0 \\ +1 & \frac{\partial H}{\partial u} < 0 \\ \text{undefined} & \frac{\partial H}{\partial u} = 0 \end{cases}$$

For the one-dimensional free-flight problem, the switching function is defined in Eq. (2.36) and the corresponding control law in Eq. (2.37).

$$\frac{\partial H}{\partial u} = \lambda_v \tag{2.36}$$

$$u = \begin{cases} -1 & \lambda_v > 0 \\ +1 & \lambda_v < 0 \\ \text{undefined} & \lambda_v = 0 \end{cases} \tag{2.37}$$

The dynamic equations for the costates can be analytically solved as:

$$\dot{\lambda_x} = 0 \implies \lambda_x(t) = C_1$$

$$\dot{\lambda_v} = -\lambda_x = -C_1 \implies \lambda_v(t) = -C_1 t + C_2$$

Since the switching function is linear, there is exactly one switch in the control. Common-sense dictates that the acceleration should be positive at the initial point for the particle to move towards the destination, i.e, $u(0) = +1$. Similarly it should be negative when arriving at the destination, or $u(T) = -1$. From the free-final time condition and because time does not appear explicitly in the Hamiltonian,

$$H(t) = 0 \implies H(0) = C_2 u(0) + 1 = 0 \implies C_2 = -1$$

$$H(T) = 0 \implies C_1\, v(T) + (-C_1 T + -1)u(T) + 1 = 0 \implies C_1 T + 2 = 0 \implies C_1 = -\frac{2}{T}$$

Therefore the switching function is $\lambda_v = (2/T)t - 1$, which crosses zero at $t = T/2$. The velocity profile is then given by:

$$v(t) = \begin{cases} t & 0 \leq t \leq T/2 \\ T - t & T/2 \leq t \leq T \end{cases}$$

Applying the boundary condition on x,

$$\begin{aligned} x(T) &= \int_0^{T/2} t\ dt + \int_{T/2}^{T} (T-t)\ dt \\ &= \frac{T^2}{8} + T^2 - \frac{T^2}{2} - \frac{T^2}{2} + \frac{T^2}{8} = 1 \\ \implies \frac{T^2}{4} &= 1 \\ T &= 2.0 \end{aligned} \tag{2.38}$$

Therefore the analytical solution for $x(t)$ is

$$x(t) = \begin{cases} t^2/2 & 0 \leq t \leq T/2 \\ Tt - \frac{t^2}{2} - \frac{T^2}{4} & T/2 \leq t \leq T \end{cases} \tag{2.39}$$

Regularized Optimal Control Formulation

Since the path constraint in this problem is a control bound, it is a zero-order constraint. Therefore the only extra variable to be added to the optimal control problem is the new control, u_{e1}. The two-sided saturation function from Eq. (2.18) is used to regularize the control bound. The new Hamiltonian is then defined as:

$$H = \lambda_x v + \lambda_v u + 1 + \epsilon u\,_{e1}^{2} + \mu\left(u - \psi(u_{e1})\right) \tag{2.40}$$

where ψ is defined in Eq. (2.18).

The extended two-point DAE-BVP is then obtained as:

$$\dot{x} = v, \quad \dot{v} = u \tag{2.41a}$$

$$\dot{\lambda_x} = 0, \quad \dot{\lambda_v} = u \tag{2.41b}$$

$$x(0) = v(0) = v(1) = 0 \tag{2.41c}$$

$$x(1) = 1 \tag{2.41d}$$

$$\frac{\partial H}{\partial u} = \lambda_v + \mu = 0 \tag{2.41e}$$

$$\frac{\partial H}{\partial u_{e1}} = \psi\, u_{e1}\mu + 2\epsilon u\,_{e1} = 0 \tag{2.41f}$$

$$\frac{\partial H}{\partial \mu} = u - \psi(u_{e1}) = 0 \tag{2.41g}$$

Initial Guess and Solution Strategy

The DAE-BVP in Eq. (2.41) is first converted into an explicit ODE by differentiating the algebraic conditions as described in Section 2.4. After trial and error, the initial guess for the costates was chosen as $\lambda_x = -0.2$ and $\lambda_v = -0.4$, integrated until $T = 10$. The states were set to zeros as in the actual initial conditions in the problem statement. Homotopy continuation was performed on the terminal position and velocity until the problem requirements were satisfied as shown in Fig 2.9. The dark blue line represents the initial guess and the red line is the final case where the boundary conditions in the problem statement for x and v are satisfied. The regularization parameter, ϵ was initially set to 2. It was later brought down to 10^{-3} using continuation in a later step as discussed in the next section. Constraint normalization is not required in this problem as the constraint limit is already equal to one.

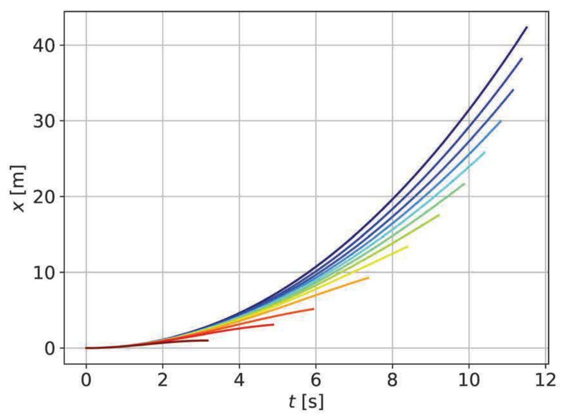


Figure 2.9. One-Dimensional Free Flight – Initial Continuation

Analysis

The evolution of the trajectory as ϵ is decreased from 10^{-1} to 10^{-3} during continuation is shown in Figure 2.10. The line in dark blue with an ϵ of 10^{-1} has a larger T, i.e, a less optimal result. As ϵ gets smaller, the trajectory approaches the optimal solution while still keeping the state and controls a single smooth arc. The actual closed-form optimal solution has a discontinuity as shown in Eq. (2.39).

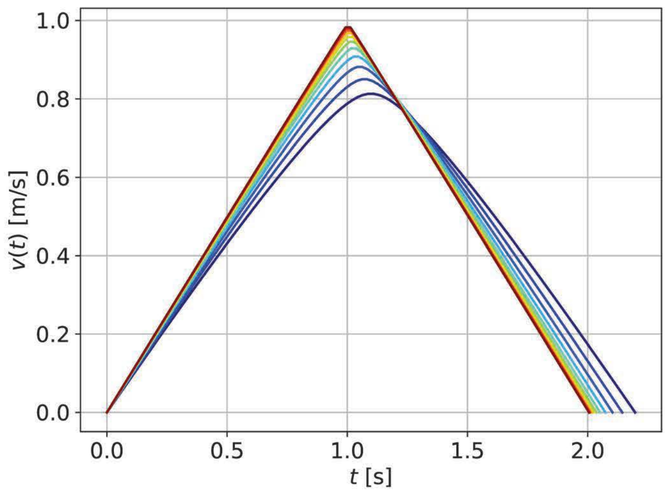


Figure 2.10. One-Dimensional Free Flight – Evolution of Regularized Solution

The trajectories for x and v, obtained using ICRM with ϵ=10^{-3} is compared to the analytical solution in Figure 2.11, and the corresponding control history can be seen in Figure 2.12. The control histories and the trajectories obtained using ICRM are seen to match very closely with the analytical solution. The optimal control computed using ICRM very closely approaches the bang-bang solution, and the corner in v at $T/2$ is modeled as closely as possible. This example demonstrates that ICRM is capable of solving problems that have a bang-bang optimal control solution in the presence of control bounds.

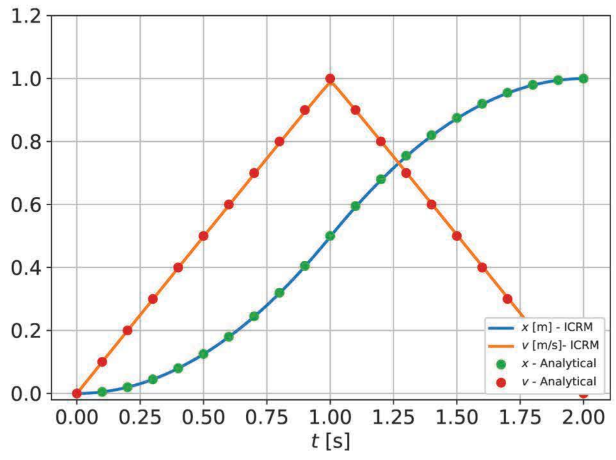


Figure 2.11. One-Dimensional Free Flight - Trajectory

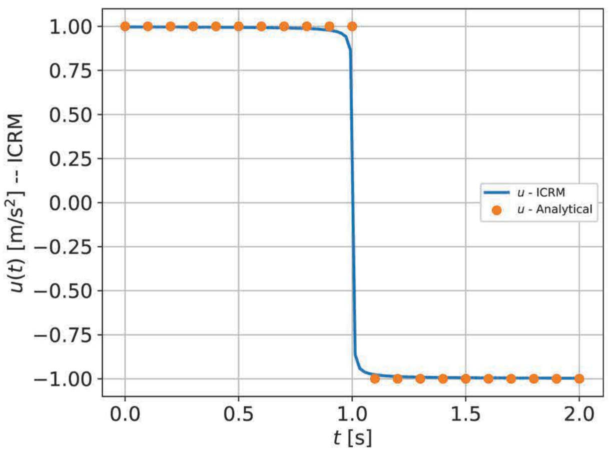


Figure 2.12. One-Dimensional Free Flight - Control History

2.8.3 Maximum Terminal Energy Hypersonic Trajectory with Heat Rate Constraint

To demonstrate that ICRM can be applied to more complex aerospace problems, a scenario involving maximum terminal energy trajectories of a slender hypersonic vehicle is examined. The vehicle is assumed to be capable of angle-of-attack (AoA) control and having a peak L/D of around 2.4. The boundary conditions and the environment parameters are listed in Tables 2.1 and 2.2 respectively. The same dynamic model and parameters are also used in the next section where an additional constraint is added to the problem.

Table 2.1. Boundary Conditions.

State	h	v	γ	θ
Staging (t=0)	80,000 m	5000 m/s	-60 deg	0 deg
Terminal (t=T)	15,000 m	free	free	1 deg

Table 2.2. Environment Parameters.

Parameter	Value
μ	$3.986 \times 10^{14}\,\mathrm{m}^3/\mathrm{s}^2$
R_E	$6.3781 \times 10^6\,\mathrm{m}$
ρ_0	$1.2\,\mathrm{kg/m}^3$
h_s	$7500\,\mathrm{m}$

The problem is defined as follows:

$$\text{Max } v(T)^2 \tag{2.42a}$$

$$\text{Subject to :}$$

$$\dot{h} = v \sin\gamma \tag{2.42b}$$

$$\dot{\theta} = \frac{v\cos\gamma}{r} \tag{2.42c}$$

$$\dot{v} = \frac{-D}{m} - \frac{\mu\sin(\gamma)}{r^2} \tag{2.42d}$$

$$\dot{\gamma} = \frac{L}{mv} + \quad \frac{v}{r} - \frac{\mu}{vr^2} \quad \cos(\gamma) \tag{2.42e}$$

$$\begin{aligned}
r &= R_E + h \\
D &= qC_D A_{ref} \\
L &= qC_L A_{ref} \\
q &= \frac{1}{2}\rho v^2 \\
\rho &= \rho_0 \exp(-h/h_s)
\end{aligned}$$

$$\begin{aligned}
C_L &= \; C_{L1}\alpha + C_{L0} \\
C_D &= \; C_{D2}\alpha^2 + C_{D1}\alpha + C_{D0}
\end{aligned} \tag{2.42f}$$

In this section the problem is solved with a single path constraint, and the result is validated using the MPBVP formulation of the necessary conditions. A stagnation point heat rate constraint is applied to the problem defined in Eqs. (2.42). The heat rate is computed using the Sutton-Graves convective heating equations [127] as shown in Eq. (2.43).

$$\dot{Q} = k \quad \overline{\frac{\rho}{r_n}} v^3, k = 1.74153 \times 10^{-4} \text{ for Earth} \tag{2.43}$$

Necessary Conditions of Optimality – MPBVP

The necessary conditions of optimality is derived for both the constrained and unconstrained versions of the problem. The continuation methodology described in Chapter 1 is used to solve the problem starting with a trivial initial guess.

For the unconstrained problem, the Hamiltonian is defined as:

$$\begin{aligned} H = \boldsymbol{\lambda}^T \boldsymbol{f} =& \lambda_h v \sin\gamma + \lambda_\theta \frac{v\cos\gamma}{r} + \lambda_v \quad \frac{-D}{m} - \frac{\mu \sin\gamma}{r^2} \\ &+ \lambda_\gamma \quad \frac{L}{mv} + \quad \frac{v}{r} - \frac{\mu}{vr^2} \quad \cos\gamma \end{aligned} \tag{2.44}$$

The two point boundary value problem is defined as:

$$\dot{h} = v\sin\gamma \tag{2.45a}$$

$$\dot{\theta} = \frac{v\cos\gamma}{r} \tag{2.45b}$$

$$\dot{v} = \frac{-D}{m} - \frac{\mu\sin\gamma}{r^2} \tag{2.45c}$$

$$\dot{\gamma} = \frac{L}{mv} + \quad \frac{v}{r} - \frac{\mu}{vr^2} \quad \cos\gamma \tag{2.45d}$$

$$\dot{\lambda}_h = -\frac{\partial H}{\partial h} \tag{2.45e}$$

$$\dot{\lambda}_\theta = -\frac{\partial H}{\partial \theta} \tag{2.45f}$$

$$\dot{\lambda}_v = -\frac{\partial H}{\partial v} \tag{2.45g}$$

$$\dot{\lambda}_\gamma = -\frac{\partial H}{\partial \gamma} \tag{2.45h}$$

with the algebraic condition for the optimal control law being defined as:

$$\frac{\partial H}{\partial \alpha} = 0 \tag{2.46a}$$

$$\tag{2.46b}$$

The boundary conditions on the states are given in Table 2.1, and the boundary conditions on the costates and the free-final time condition are defined as follows:

$$\lambda_v(T) = -2v(T) \tag{2.47a}$$

$$\lambda_\gamma(T) = 0 \tag{2.47b}$$

$$H(T) = 0 \tag{2.47c}$$

When solving the constrained problem, the assumption is that there is one constrained arc in the solution at the point where the heat rate peaks. The trajectory is split at this point and a new arc is inserted into the unconstrained problem. New boundary conditions are required at this boundary point along with different dynamic equations for costates and a new control law. The heat rate constraint is found to be of order 1 since it has to be differentiated once before the control variable, α appears.

$$S(\boldsymbol{x}) = k \quad \overline{\frac{\rho}{r_n}} v^3 \tag{2.48a}$$

$$S^{(1)} = 3kv^2 \quad \overline{\frac{\rho}{r_n}} \quad -\frac{D}{m} - \frac{\frac{\mu \sin\gamma}{r^2}}{\dot{Q}_{max}} \quad - kv^4 \quad \overline{\frac{\rho}{r_n}} \frac{\sin\gamma}{2\ h_s\ \dot{Q}_{max}} \tag{2.48b}$$

$S^{(1)}$ is adjoined to the Hamiltonian using a Lagrange multiplier μ_1 to form the new Hamiltonian, H_c.

$$H_c = H + \mu_1 \quad 3kv^2 \quad \overline{\frac{\rho}{r_n}} \quad -\frac{D}{m} - \frac{\frac{\mu \sin\gamma}{r^2}}{\dot{Q}_{max}} \quad - kv^4 \quad \overline{\frac{\rho}{r_n}} \frac{\sin\gamma}{2\ h_s\ \dot{Q}_{max}} \tag{2.49}$$

The constrained costate rates and control laws are computed based on H_c as shown below:

$$\dot{\lambda}_h = -\frac{\partial H_c}{\partial h} \tag{2.50a}$$

$$\dot{\lambda}_\theta = -\frac{\partial H_c}{\partial \theta} \tag{2.50b}$$

$$\dot{\lambda}_v = -\frac{\partial H_c}{\partial v} \tag{2.50c}$$

$$\dot{\lambda}_\gamma = -\frac{\partial H_c}{\partial \gamma} \tag{2.50d}$$

The Lagrange multiplier μ_1 is a time-varying quantity which is zero for the unconstrained trajectory and is computed simultaneously with the control α for the constrained arc.

$$\frac{\partial H_c}{\partial \alpha} = 0, \quad \frac{\partial H_c}{\partial \mu_1} = 0 \tag{2.51}$$

At the entry to the constrained arc, the following corner conditions apply.

$$\boldsymbol{x}(t_1^-) = \boldsymbol{x}(t_1^+) \tag{2.52a}$$

$$\boldsymbol{\lambda}(t_1^-) - \boldsymbol{\lambda}(t_1^+) + \boldsymbol{\Pi}^T \mathbf{N}_x = 0 \tag{2.52b}$$

$$H(t_1^-) = H_c(t_1^+) \tag{2.52c}$$

$$\text{where } \mathbf{N}_x = \frac{\partial S(\boldsymbol{x})}{\partial \boldsymbol{x}} \tag{2.52d}$$

Π is an unknown parameter to be estimated during the numerical solution process, and t_1 represents the entry-junction for the constrained arc. At the exit junction t_2, the states, costates, and Hamiltonian are assumed to be continuous as shown in Eq. (2.53). These conditions together form a multi-point boundary value problem with three arcs that is solved using a multiple shooting algorithm.

$$\boldsymbol{x}(t_2^-) = \boldsymbol{x}(t_2^+) \tag{2.53a}$$

$$\boldsymbol{\lambda}(t_2^-) = \boldsymbol{\lambda}(t_2^+) \tag{2.53b}$$

$$H_c(t_2^-) = H(t_2^+) \tag{2.53c}$$

$$\tag{2.53d}$$

Necessary Conditions of Optimality – ICRM

Before regularizing using saturation functions, the constraint is normalized and stated as:

$$\frac{\dot{Q}}{\dot{Q}_{max}} \leq 1 \tag{2.54}$$

Since the heat-rate constraint is of order 1, only one extra state variable, ξ, and one control, u_{e1}, need to be added to the system. The dynamic equation for the new state is defined as $\dot{\xi} = u_{e1}$. The one-sided saturation function from Eq. (2.16) is used to regularize the path constraint.

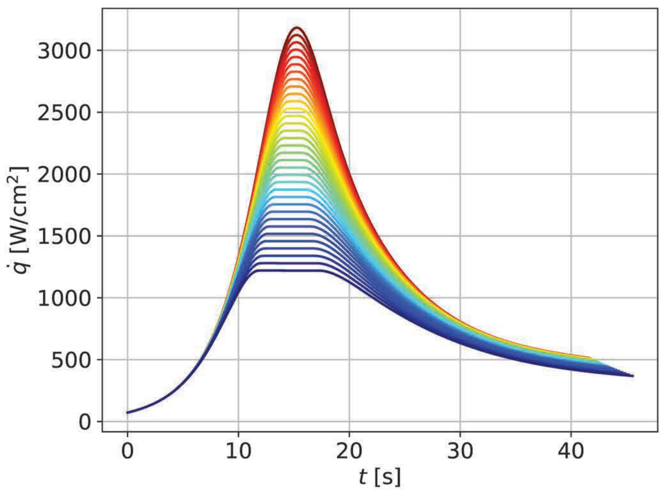
$$\begin{aligned} S = {} & k \quad \overline{\frac{\rho}{r_n}} v^3 = 1 - \exp\left(-\xi\right) \\ \Longrightarrow {} & k \quad \overline{\frac{\rho}{r_n}} v^3 - \left(1 - \exp\left(-\xi\right)\right) = 0 \end{aligned} \tag{2.55}$$

$$\begin{aligned} & S^{(1)} = \exp\left(-\xi\right)\, u_{e1} \\ & \quad \Longrightarrow S^{(1)} - \exp\left(-\xi\right)\, u_{e1} = 0 \\ & \quad \Longrightarrow 3kv^2 \quad \overline{\frac{\rho}{r_n}} \quad - \frac{D}{m} - \frac{\frac{\mu \sin\gamma}{r^2}}{\dot{Q}_{max}} \quad - kv^4 \quad \overline{\frac{\rho}{r_n}} \frac{\sin\gamma}{2\ h_s\ \dot{Q}_{max}} \\ & \qquad - \exp\left(-\xi\right)\, u_{e1} = 0 \end{aligned} \tag{2.56}$$

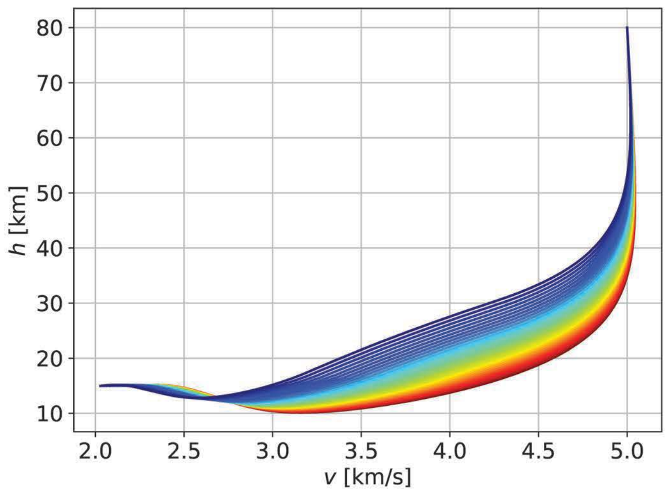
Eq. (2.55) forms the initial condition on the new state variable ξ. Eq. (2.56) is added as an equality constraint to the problem. The path cost of the problem is changed to $L = {}_0^T \epsilon u_{e1}^2 dt$ where is the regularization parameter. The necessary conditions of optimality for the extended optimal control problem were derived using Euler-Lagrange equations, and the control dynamic equations were calculated as described in Section 2.4. The resulting two-point boundary value problem was solved using a shooting method.

Solution Strategy - MPBVP

The initial guess for the numerical solver is created by integrating the dynamic equations forward from the entry interface conditions except for the flight-path angle γ which is set to -90 degrees. The costates are all set to 0.1 and the trajectory was integrated for 0.1 seconds. When solving using conventional optimal control, initially the unconstrained problem is solved in this manner. The constrained arc is then introduced at the point with maximum constraint violation. In subsequent steps, this constraint violation is reduced down to zero to obtain the solution to the original constrained problem. This continuation process is illustrated in Figure 2.13. The lines transitioning from red to blue denotes a change in $\dot{Q}_{max}$ from its value in the unconstrained solution to the required value of 1200 W/cm^2. As expected, a penalty in cost $v(T)$ can be observed as a lower peak heat-rate is enforced. The final solution obtained using MPBVP is compared to that obtained using ICRM in the next section.



(a) Heat Rate Profile

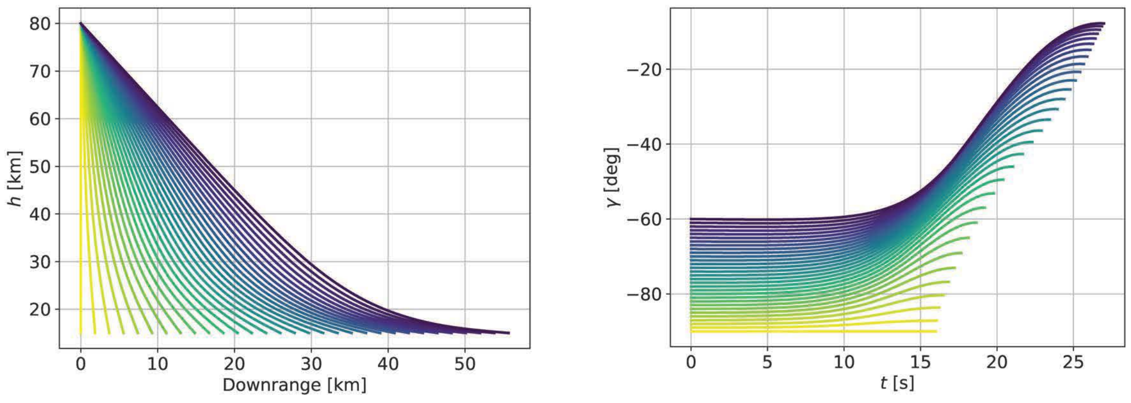


(b) Energy Plot

Figure 2.13. Heat Rate Constraint – MPBVP Continuation in $\dot{Q}_{max}$

Solution Strategy - ICRM

The same initial guess that was used for the MPBVP solution was used with ICRM. When using ICRM, the regularization parameter, ϵ for the heat-rate constraint is initially set to 10^{-2} and $\dot{Q}_{max}$ is set to 10,000 W/cm^2 in order to ensure that the constraint does not impede with the solution process especially at lower altitudes. Continuation in terminal altitude is used to extend the trajectory towards the ground. Once the trajectory reaches the targeted altitude (15 km), a continuation is performed on the initial flight-path angle to -60 deg and the terminal downrange distance to 55 km as shown in Figure 2.14. Further continuation is performed on $\theta(T)$ to extend it to out to 110 km.



(a) Altitude-Downrange (b) Flight-Path Angle Profile

Figure 2.14. Heat Rate Constraint – ICRM Continuation in $\gamma(0)$ and $\theta(T)$

After the boundary conditions in altitude and downrange distance were matched with the problem statement, continuation is performed on the heat-rate limit, $\dot{Q}_{max}$, and then on the regularization parameter, ϵ. This is performed in multiple steps some of which are shown in Figures 2.15 and 2.16.

Figure 2.15 plot shows the effect on the trajectory as the constraint limit, $\dot{Q}_{max}$ is changed from 10,000 W/cm^2 down to the design limit of 1,200 W/cm^2. It can be seen that through the entire process, the actual peak heat rate does not exactly reach the constraint. This is due to the effect of ϵ which is still relatively large enough at 10^{-4} to cause this "push-off" factor. As seen in Fig 2.14(a), the vehicle is forced to climb higher to avoid the higher heat-rate that is encountered in the lower atmosphere.

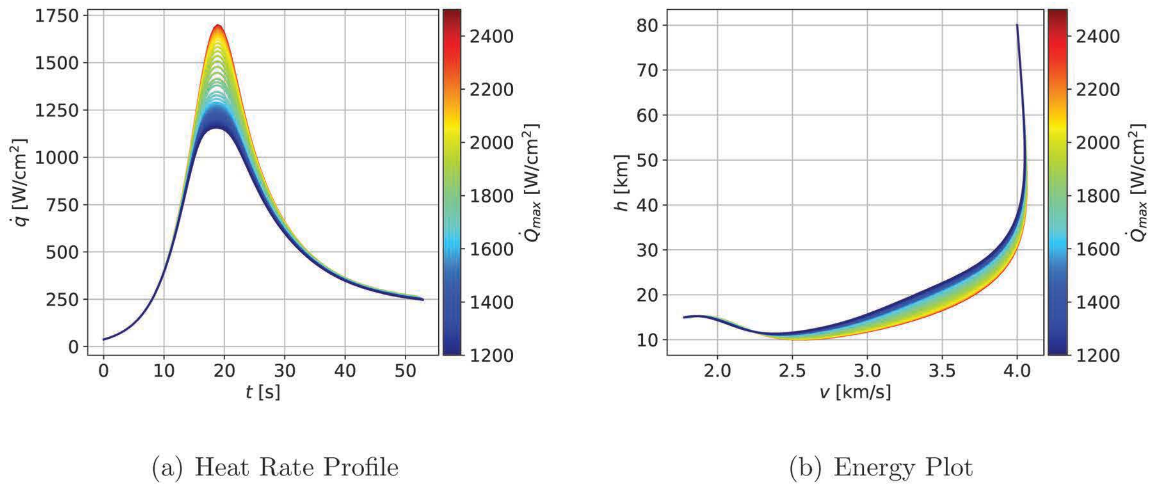
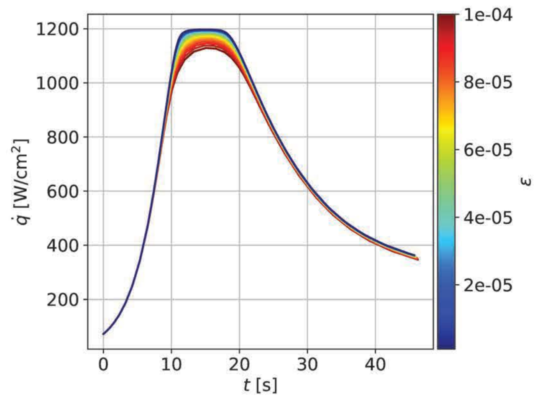


Figure 2.15. Heat Rate Constraint – ICRM Continuation in $\dot{Q}_{max}$ from 2500 W/cm^2 (red) to 1200 W/cm^2 (blue) with $\epsilon = 10^{-4}$

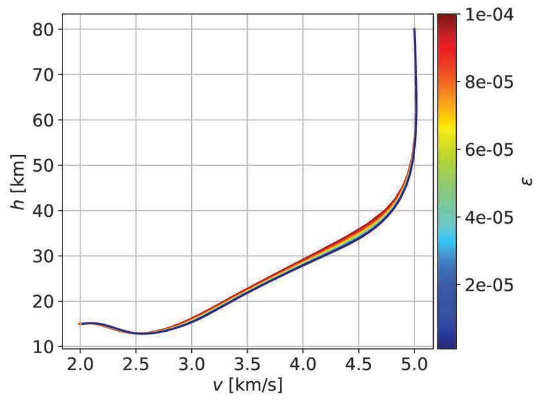
Once $\dot{Q}_{max}$ is at the desired design value of 1200 W/cm^2, the next step is to reduce the regularization parameter to bring the trajectory closer to the optimal solution. This process is illustrated in Figure 2.16 where ϵ is reduced from 10^{-2} to 10^{-6}. As discussed before in Section 2.7, reducing ϵ makes the constraint less restrictive and a slightly higher peak-heat rate is allowed once ϵ equals 10^{-6}. This is reflected in the energy plot (Fig 2.16(b)) as well with the vehicle diving slightly deeper into the atmosphere to attain a slightly more optimal solution at the cost of more heating.

Analysis

The constrained heat-rate solution obtained using ICRM is compared to that obtained using MPBVP for validation in Figure 2.17. With $\epsilon = 10^{-6}$, the two trajectories can be seen to match very well. The trajectory touches the constraint and is



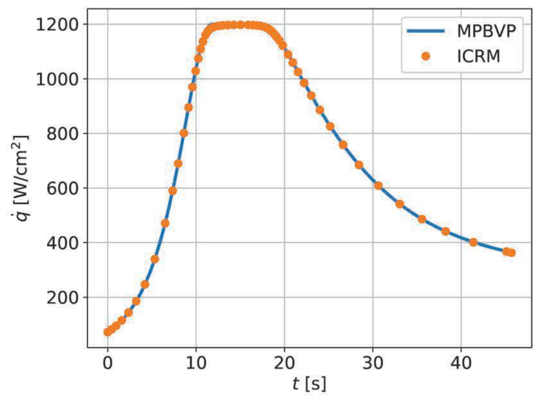
(a) Heat Rate Profile



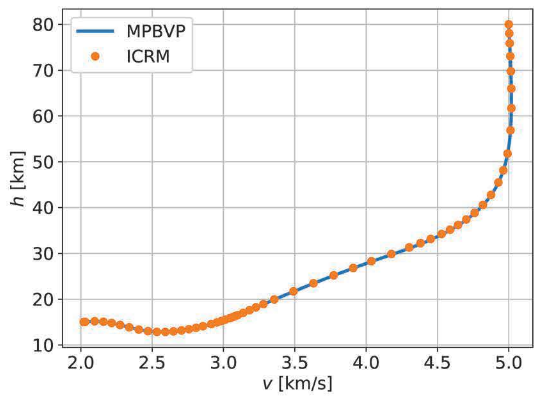
(b) Energy Plot

Figure 2.16. Heat Rate Constraint – ICRM Continuation in ϵ from 10^{-4} (red) to 10^{-6} (blue)

on it for 8 seconds while remaining a single continuous curve. The cost function (the terminal velocity) also matches that which was obtained using the MPBVP method.



(a) Heat Rate Profile



(b) Energy Plot

Figure 2.17. Heat Rate Constraint – MPBVP vs ICRM

2.8.4 Maximum Terminal Energy Hypersonic Trajectory with Heat Rate and Angle-of-Attack Constraints

When the hypersonic trajectory problem is formulated as in the last section, there is a side effect that there is no actual bound on the angle-of-attack. For a maximum terminal energy problem, this is not a significant issue as the optimal solution tends to minimize the angle-of-attack.

Figure 2.18 shows the control history for the heat-rate constrained trajectory from the last section. It can be seen that initially, the angle of attack has very high and unrealistic values of close to 80 degrees. This is not too critical as this happens in a phase of the trajectory where there is hardly any atmosphere. However, this can be mitigated by adding a constraint on the angle-of-attack.

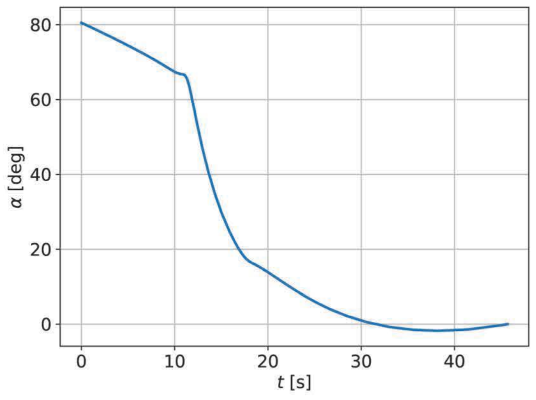


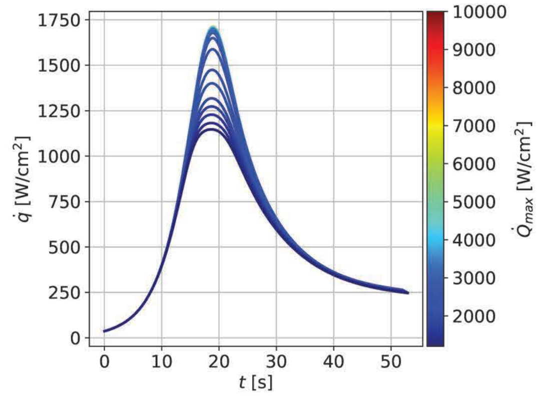
Figure 2.18. Control History for Heat Rate Constrained Trajectory

In order to demonstrate the handling of multiple path constraints using ICRM, a new problem is set up where an angle-of-attack constraint can be enforced in the problem simultaneously with the heat rate constraint, and both constraints are satisfied at all points. Since this is a two-sided constraint, the saturation function from Eq. (2.18) is used to implement this constraint in ICRM. α_{max} is set to 40°.

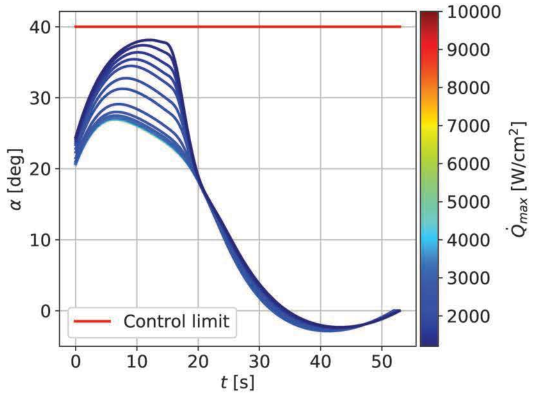
The normalized angle-of-attack constraint is defined as:

$$-1 \leq \frac{\alpha}{\alpha_{max}} \leq 1 \tag{2.57}$$

A continuation methodology similar to that used in the previous section is used to build up the trajectory from a trivial initial guess. The regularization parameter for both the heat-rate (ϵ_1) and the angle-of-attack (ϵ_2) constraints, are initially set to 10^{-4}. Figure 2.19 shows the continuation in $\dot{Q}_{max}$ with both constraints enforced simultaneously. The push-off factor due to the relatively high values or ϵ can be seen in both the heat-rate and the angle-of-attack profiles in Figures 2.19(a) and 2.19(b), respectively. The control effort increases as the vehicle is forced to fly at a higher altitude in order to maintain a lower heat-flux as required by the heat-rate constraint.



(a) Heat Rate Profile

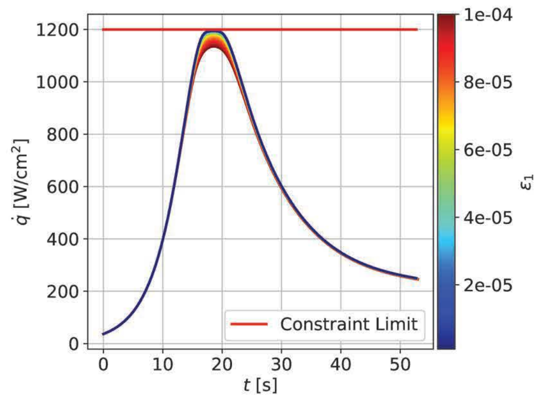


(b) Angle-of-Attack profile

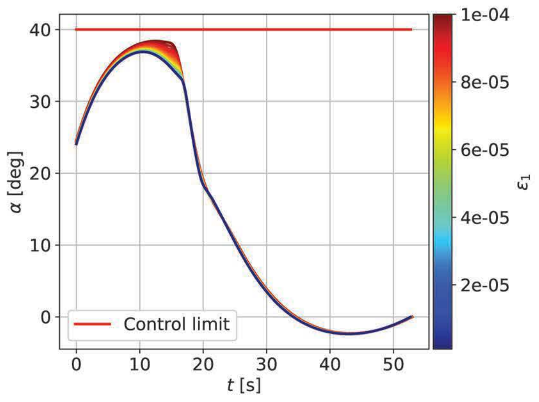
Figure 2.19. Heat Rate & AoA Constraint – Evolution in $\dot{Q}_{max}$ from 10,000 W/cm^2 to 1200 W/cm^2

In the next continuation step, the regularization parameter for the heat-rate constraint, ϵ_1 is reduced from 10^{-4} to 10^{-6}, bringing the trajectory closer to the heat-rate constraint. This evolution is shown in Figure 2.20. Changing ϵ_1 also affects control profile as seen in Fig 2.20(b), reducing the peak value of α. This is a side -effect of the change in relative weights of different terms of the cost functional as discussed before in Section 2.7. As ϵ_1 is brought down to 10^{-6}, the heat-rate profile can be seen to change so that it touches the constraint as shown in Fig 2.20(a).

The final step in the process is to reduce ϵ_2 from 10^{-4} to 10^{-6} so that the control profile also follows the constraint as shown in Figure 2.21. The change in the heat-rate



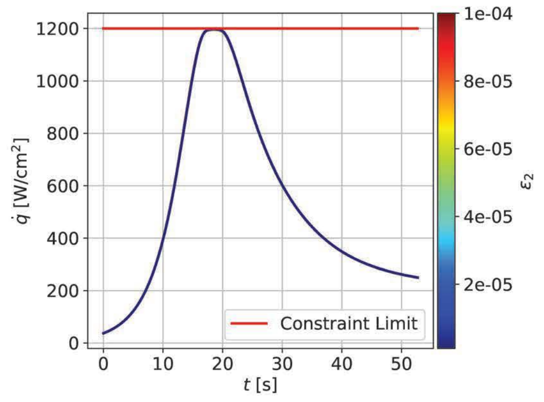
(a) Heat Rate Profile



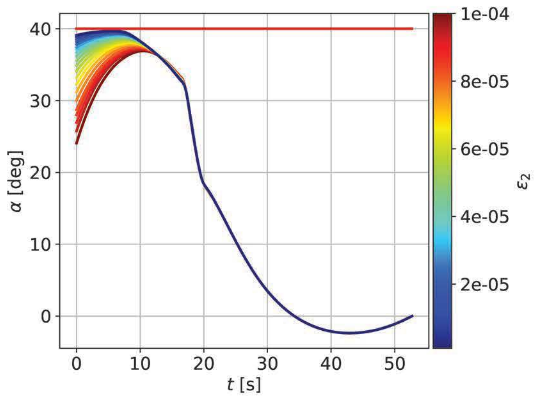
(b) Angle-of-Attack profile

Figure 2.20. Heat Rate & AoA Constraint – Evolution in ϵ_1 from 10^{-4} (red) to 10^{-6} (blue)

profile is almost imperceptible at this point as seen in Figure 2.21(a) since the peak value for the angle-of-attack happens high in the atmosphere where it is not able to impact the trajectory of the vehicle in a significant manner. However, by enforcing the control constraint, it is possible to ensure that the optimal solution obtained is satisfying all the design constraints in the original problem.



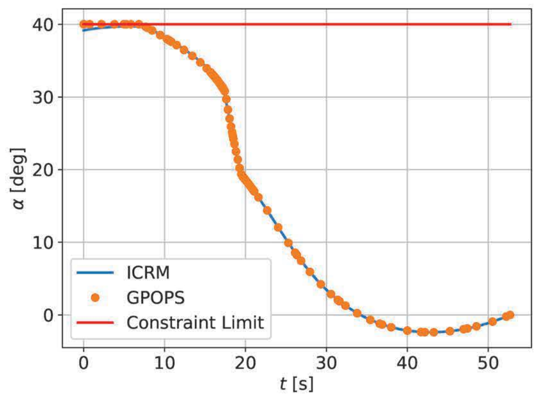
(a) Heat Rate Profile



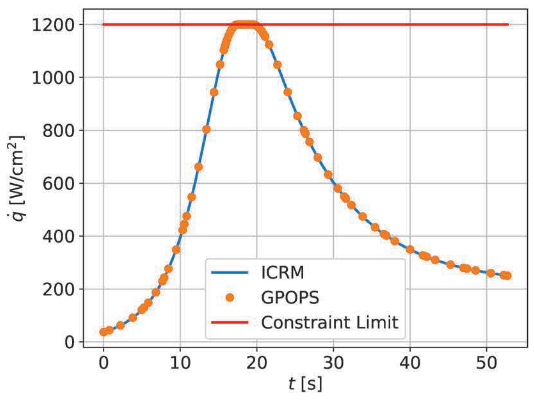
(b) Angle-of-Attack profile

Figure 2.21. Heat Rate & AoA Constraint – Evolution in ϵ_2 from 10^{-4} (red) to 10^{-6} (blue)

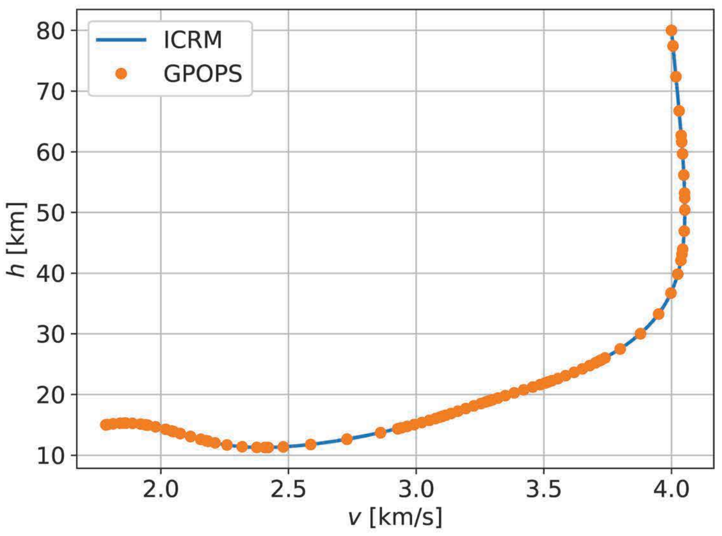
Adding the control constraint does not influence the trajectory much, as the control saturation occurs in a region with very little dynamic pressure. However, adding this constraint demonstrates that ICRM is capable of solving problems in which more than one path constraint is enforced. The final solution obtained using ICRM is compared to that obtained using the direct solver GPOPS in Figures 2.22.



(a) Control History



(b) Heat Rate Profile



(c) Energy Plot

Figure 2.22. Heat Rate & AoA Constraint – ICRM vs GPOPS

2.9 Application Problem

2.9.1 Problem Statement

The capabilities of ICRM are demonstrated in this section by applying it to a problem with multiple constraints. The problem consists of a co-operative engagement scenario with multiple vehicles in the terminal guidance phase. It is based on prior work in Ref. 107, which addresses optimal guidance laws for nonlinear missile models with impact time and angle constraints for a single vehicle. The same dynamic model is used, and the scenario is extended to be a three-dimensional model that includes multiple vehicles engaging a target simultaneously with impact angle and time constraints along with a keep-out zone path constraint. The objective is to minimize total control effort expended by all vehicles.

The dynamic model for each vehicle is defined as follows:

$$\dot{x} = V \cos\psi \cos\gamma \tag{2.58a}$$

$$\dot{y} = V \sin\psi \cos\gamma \tag{2.58b}$$

$$\dot{z} = -V \sin\gamma \tag{2.58c}$$

$$\dot{\psi} = a/V \tag{2.58d}$$

where x, y and ψ are the position and heading angle of the missile respectively. The missile is assumed to be capable of maintaining a constant velocity V, and commanding any flight-path angle (glide slope) with negligible delay. The missile is assumed to start at some initial position (X_0,Y_0,Z_0) with the target at the origin. The initial and/or terminal headings may also be constrained based on the scenario being examined.

As in Ref. 107, the problem is non-dimensionalized so that it does not depend on the constant missile velocity or impact time. If t_f is the reference impact time and V is the reference speed, the non-dimensional state and control variables are defined as:

$$\bar{x} = \frac{x}{Vt_f} \tag{2.59a}$$

$$\bar{y} = \frac{y}{Vt_f} \tag{2.59b}$$

$$\bar{z} = \frac{z}{Vt_f} \tag{2.59c}$$

$$\bar{\tau} = \frac{t}{t_f} \tag{2.59d}$$

Since there are multiple vehicles in the problem, each flying at a constant velocity, the speed of the first vehicle is chosen as the reference value and those of the remaining vehicles are added as free states (with constant values) that can optimized. The non-dimensional optimal control problem for n vehicles is then stated as:

$$\text{Min } J = {}_{0}^{T} \; {}_{i=1}^{n} \; \bar{u}_i^2 + \gamma_i^2 \tag{2.60a}$$

$$\text{Subject to:} \tag{2.60b}$$

$$\dot{\bar{x}}_i = \bar{v}_i \cos\psi_i \cos\gamma \tag{2.60c}$$

$$\dot{\bar{y}}_i = \bar{v}_i \sin\psi_i \cos\gamma \tag{2.60d}$$

$$\dot{\bar{z}}_i = -\bar{v}_i \sin\gamma \tag{2.60e}$$

$$\dot{\bar{v}}_i = 0 \tag{2.60f}$$

$$\dot{\psi}_i = \bar{u}_{max}\bar{u}_i \tag{2.60g}$$

$$\bar{x}_i(0) = \bar{X}_{i0}, \quad \bar{y}_i(0) = \bar{Y}_{i0}, \quad \bar{z}_i(0) = \bar{Z}_{i0} \tag{2.60h}$$

$$\bar{v}_0(0) = 1 \tag{2.60i}$$

$$\psi_i(T) = \psi_{if} \tag{2.60j}$$

$$|\bar{u}_i| \leq 1 \tag{2.60k}$$

$$\overline{(\bar{x}_i - x_{c1})^2 + (\bar{y}_i - y_{c1})^2} \geq r_{c1} \tag{2.60l}$$

$$\overline{(\bar{x}_i - x_{c2})^2 + (\bar{y}_i - y_{c2})^2} \geq r_{c2} \tag{2.60m}$$

where the control variables are $\bar{u}_i$ and γ_i, the turn rate and flight-path angle of the i-th vehicle, respectively. $\bar{u}_{max}$ is a scaling factor representing the magnitude of the acceleration command. The path constraints in Eqs. (2.60l) and (2.60m) and the control limits in Eq. (2.60k) are enforced on all vehicles simultaneously. The nominal scaling values used for non-dimensionalizing the problem are defined in Eq. (2.61). The regularization parameter, , for all path constraints is set to 10^{-6} in all the scenarios in this section.

$$V_{ref} = 300 \text{ m/s} \tag{2.61a}$$

$$T_{ref} = 50 \text{ s} \tag{2.61b}$$

2.9.2 Nominal Solution

The path constraints are initially positioned so that only one of them is active on the trajectory. The boundary conditions applied to the vehicle states are shown in Table 2.3. These values are scaled using the scale factors defined from Eq. (2.61) to get the boundary conditions for the non-dimensional state variables. The nominal path constraint parameters are shown in Table 2.4, and these values are also scaled in the same manner.

Table 2.3. Nominal Boundary Conditions

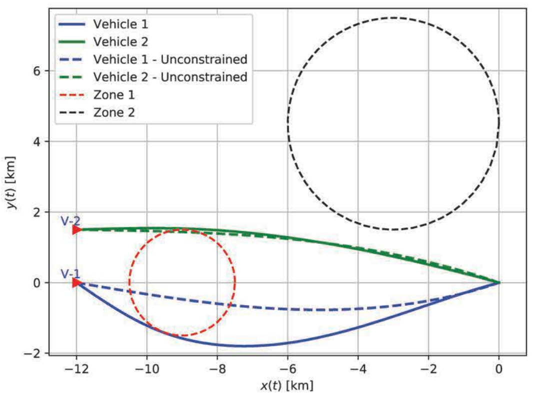
	$X_i(0)$	$Y_i(0)$	$Z_i(0)$	$\psi_i(T)$
Vehicle-1	-12 km	0.0 km	1.5 km	-15 deg
Vehicle-2	-12 km	1.5 km	1.5 km	15 deg

The trajectory solutions for the nominal problem conditions with and without the no-fly zones are shown in shown in Figure 2.23. In this 2D plot and other similar

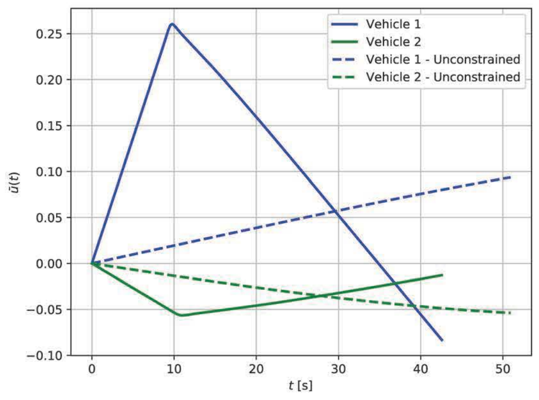
Table 2.4. Path Constraint Parameters – One Active Constraint

	x_c	y_c	r_c
Zone-1	-9.0 km	0.0 km	1.5 km
Zone-2	-3.0 km	4.5 km	3.0 km

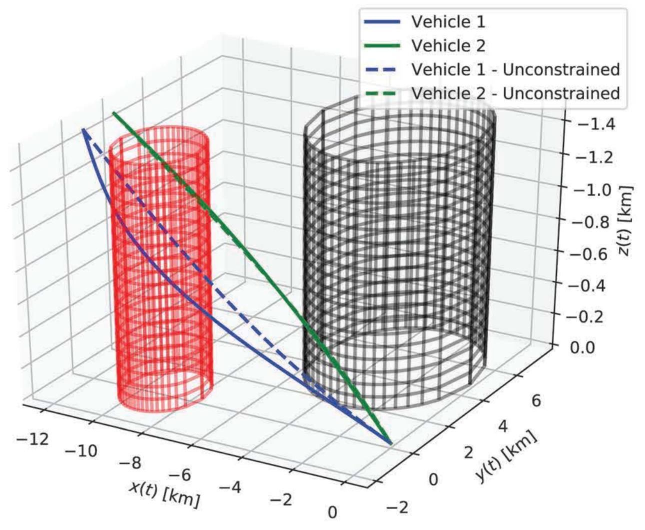
plots that follow in this section, the trajectories of Vehicles 1 and 2 will be marked as "V-1" and "V-2", respectively. Figure 2.23(c) shows a 3D view of the trajectories showing the significant variation in the first vehicle's trajectory due to the no-fly zone. The control histories of the two vehicles are shown in Figure 2.23(b). The control has corners where the trajectory intersects with the constraint as expected from optimal control theory (Section 1.2.3). The more advantageous starting position of the second vehicle makes it so that it barely touches the first constraint while maintaining an optimal trajectory, leading to a less drastic change in its control.



(a) Planar Trajectory Profile



(b) Control History



(c) 3D Trajectories

Figure 2.23. Two-Vehicle Co-operative Engagement – One Active Constraint

2.9.3 No-Fly Zone Position

In this section, the effect of changing the position of the no-fly zones is studied. The starting condition of the scenario is modified from that in the previous section to match those given in Table 2.5. In this case, both constraints are active in the trajectory of at least one of the two vehicles as shown in Fig. 2.24.

Table 2.5. Path Constraint Parameters – Two Active Constraints

	x_c	y_c	r_c
Zone-1	-9.0 km	0.0 km	1.5 km
Zone-2	-3.75 km	3.325 km	3.0 km

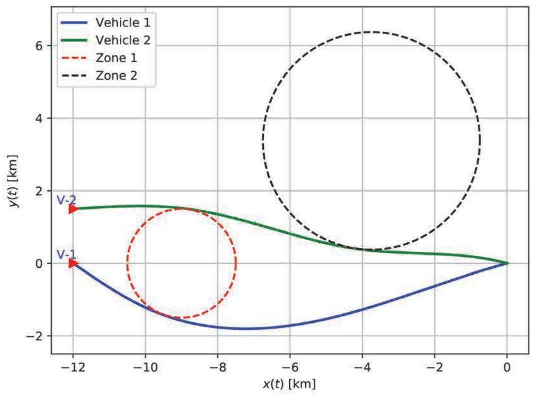
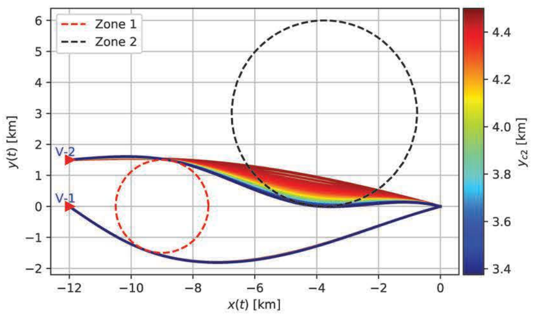
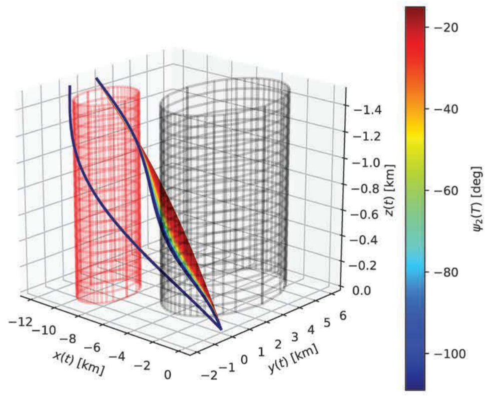


Figure 2.24. Two-Vehicle Co-operative Engagement – Two Active Constraints

In the next step, Zone-2 is moved further South so that it significantly affects the trajectory of Vehicle-2. The effects of this change in y_{c2} is shown in Fig. 2.25. Moving Zone-2 in this manner does not have any significant effect on the altitude profile of the trajectory as seen in Fig 2.25(b). However, the presence of the constraint does affect the constant speed, v_2 of Vehicle-2 as shown in Fig. 2.26(a). v_2 remains unchanged with change in y_{c2} until the constraint becomes active as part of the continuation process. As the no-fly zone is moved further South, v_2, though constant for each individual solution, is different for the different solutions in the trade-study as y_{c2} changes. This happens in order for Vehicle-2 to match the impact time with Vehicle-1 which remains unaffected by Zone-2. The control history also drastically changes as the constraint becomes active. It can be inferred from these plots that the optimal way to go around these no-fly zones is to touch them at a single point when possible rather than following the curve. This is also illustrated in the sharp corners that show up in the control history (Fig. 2.26(b)) as the no-fly zone is moved South.



(a) Planar Trajectory

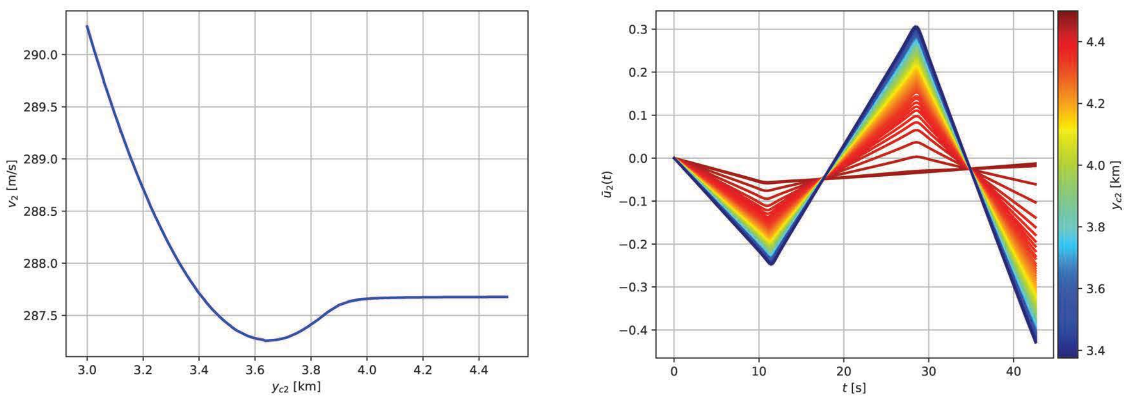


(b) 3D Trajectory

Figure 2.25. Two-Vehicle Co-operative Engagement – Zone-2 - Trajectories

2.9.4 Terminal Impact Angle Constraints

The problem defined in Eq. (2.60) is solved for different values of terminal headings for Vehicle-2, and the effects on the trajectory is examined. Starting with the condition in Figure 2.23(a), the terminal heading of Vehicle-2 is changed from -15 deg using a continuation process (Section 5.6.3). The resulting change in the vehicle trajectories are shown in Fig 2.27(c). The problem becomes extremely difficult to solve once $\psi_2(T)$ is increased beyond 110 deg. This could be because the solution is very close to being infeasible. This is also reflected in Vehicle-2's optimal control profile shown in Fig 2.27(b). The control is saturated at the end of the trajectory and



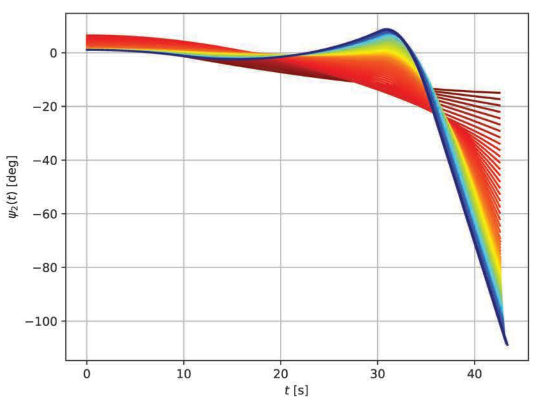
(a) v_2 vs. y_{c2} (b) Control History

Figure 2.26. Two-Vehicle Co-operative Engagement – Moving Zone-2 – Velocity and Control History

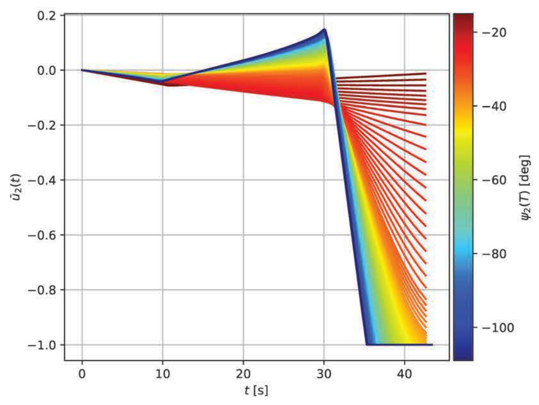
the vehicle is unable to turn any sharper to arrive at the destination from further to the right without violating other constraints such as impact-time and the keep-out zone path constraints. The steeper the arrival heading, the closer this control gets to a bang-bang style control. The Zone-2 constraint is also seen to be active for a significant amount of time as the terminal heading is increased. This is reflected in the sudden switch in $\bar{u}_2$ at the 30 second mark.

A 3D view of the evolution of Vehicle-2's trajectory is shown in Fig 2.28. The steeper arrival heading constraint causes Vehicle-2 to stay higher in the atmosphere and then suddenly dive down at the end as opposed to flying in a straight line. This is for increasing the distance flown, and thereby allowing flexibility in the time of flight to match impact time with Vehicle-1. The altitude profile forms an extra degree of freedom that is leveraged for avoiding the no-fly zones while obeying other geometry and timing constraints.

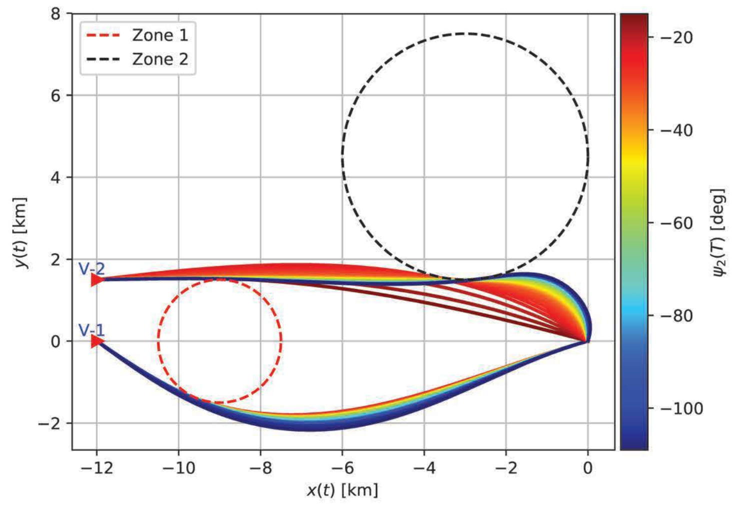
The velocity of Vehicle-2, while constant for each individual trajectory, is a free parameter that can be optimized. The initial heading of both vehicles are also free parameters. Fig. 2.29 shows the variation in the free initial heading of Vehicle-1 and the speed of Vehicle-2 as the constrained arrival heading of Vehicle-2 is changed. v_2



(a) Vehicle-2 – Heading



(b) Control History



(c) Planar Trajectory Profiles

Figure 2.27. Two-Vehicle Co-operative Engagement – Evolution of Trajectories w.r.t $\psi_2(T)$

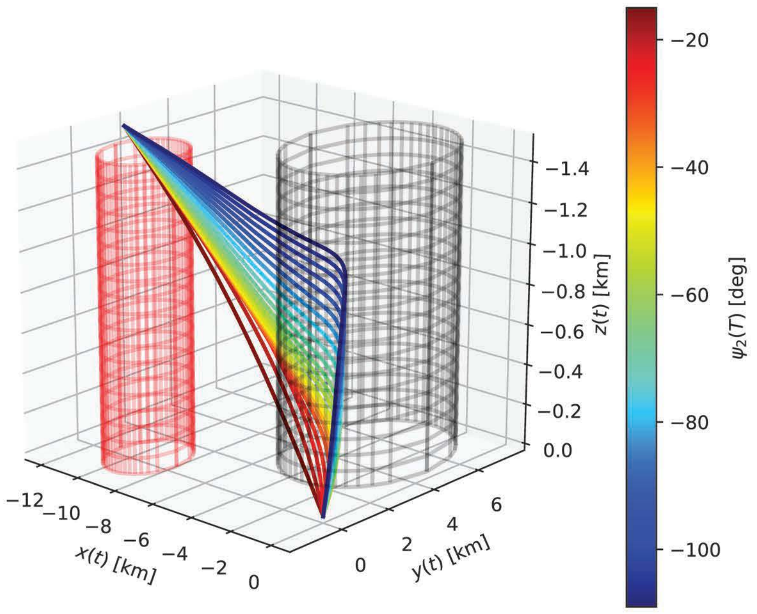


Figure 2.28. Two-Vehicle Co-operative Engagement – Evolution of 3D Trajectory of Vehicle-2 with Change in $\psi_2(T)$

is initially 287 m/s when $\psi_2(T) = 15$ deg. As this terminal boundary condition changes, the speed increases given that Vehicle-2 now has to travel a longer distance in the same amount of time. This is because the impact time of both vehicles is constrained to be the same. However, close to the maximum value of $\psi_2(T)$, there is sudden reversal and a decrease in v_2. The reason for this can be seen in the corresponding profile of the free initial heading of Vehicle-1. While it initially stays close to constant, as $\psi_2(T)$ crosses -100 deg, it is seen to rapidly decrease. This corresponds to the non-intuitive side-effect that can be seen in this scenario where Vehicle-1 actually starts flying further south and away from Zone-1 as $\psi_2(t)$ is made steeper. With these combination of constraints, it is more optimal for Vehicle-1 to stay further away from Zone-1 so that Vehicle-2 has more time to perform its maneuvers. This is one example of non-intuitive cross-coupling effects that appear when optimizing trajectories of multiple vehicles simultaneously.

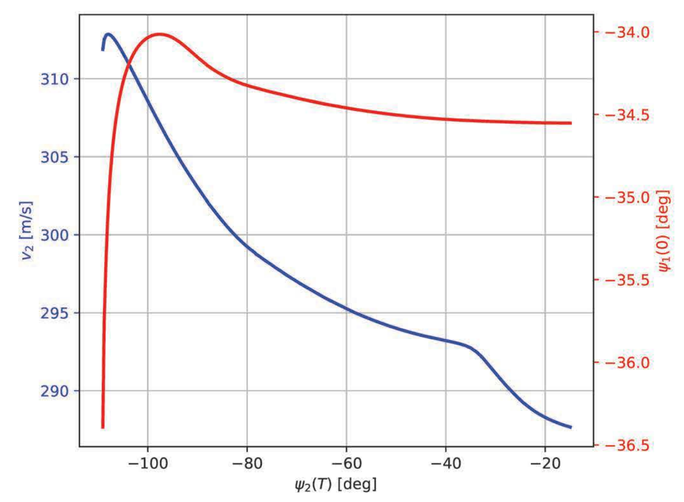


Figure 2.29. Two-Vehicle Co-operative Engagement – Cross-coupling of $\psi_1(0)$, v_2 and $\psi_2(T)$

Another notable effect seen in the optimal path for Vehicle-2 is that the Zone-1 constraint, while initially active, becomes inactive as the arrival heading, $\psi_2(T)$, turns further North. However, further changes in $\psi_2(T)$ reverses this trend, and the constraint becomes active again in order reduce the distance to target.

This scenario is one example whereby using ICRM, a smooth transition from unconstrained arcs to constrained arcs and vice versa can be achieved, facilitating the study of non-intuitive cross-coupling effects such as the ones described in this section. Performing this same trade study would be a non-trivial task if the trajectory was split into multiple arcs for handling path constraints as is the norm when using indirect methods.

2.10 Summary

The Integrated Control Regularization Method (ICRM) has been shown to be a viable process for incorporating path constraints into optimal control problems when using indirect methods. ICRM converts constrained optimal control problems into two-point boundary value problems with a few extra states for each constraint effectively making the BVP bigger. However, this is a reasonable trade-off considering that the alternative is a huge multi-point boundary problem that grows in size with the number of times each constraint is active and quickly becomes highly impractical at scale. ICRM helps avoid the challenges conventionally associated with path constraints in indirect methods, such as finding the right sequence of constrained and unconstrained arcs and providing initial guesses for interior boundary conditions. As shown in one of the scenarios in Section 2.9, keeping the solution as a single arc allows trades to be performed where path constraints smoothly change from being active to inactive and vice-versa. This is something that cannot be done with the MPBVP formulation of constrained optimal control problems.

Another possible application for ICRM is for incorporating high-fidelity models when using indirect methods. These models may be expressed as black-box functions for which analytic derivatives may not be available. This makes it very difficult to obtain a control law when using conventional optimal control theory. For example, Ref. 15 used high-fidelity atmosphere and aerodynamic models while numerically solving the optimal control law every time the system dynamics was evaluated. This resulted in very long computation times while solving the BVP. ICRM avoids this problem by folding in the control law computation into the overall root-solving process of solving the BVP. While this may lead to some numerical stability issues in some cases, it is still better than embedding a numerical root-solving process inside of the numerical integration process.

Regularizing path constraints using ICRM represents a first step towards obtaining high quality solutions for highly constrained trajectory optimization problems which would generally be considered practically impossible to solve using indirect *or* direct methods. It forms a key component of the general indirect trajectory optimization framework detailed in Chapter 5. However, ICRM also adds extra states for every constraint that is added, resulting in larger BVPs that are to be numerically solved. As the number of vehicles and the number of constraints increase, the computation time can get prohibitively large when using existing numerical methods such as multiple shooting. In order to address this, a new numerical method is developed specifically to exploit parallel computing architectures, as detailed in the next chapter.

3. QUASILINEAR CHEBYSHEV-PICARD ITERATION (QCPI)

3.1 Background

3.1.1 Prior Work

The use of emerging parallel computational architectures is one way to accelerate numerical solution of large boundary value problems. There have been different approaches for parallelizing the numerical methods underlying indirect [30], [32] as well as direct methods [128], with varying degrees of success. Past work [79] showed that while it is indeed possible to accelerate the numerical methods used for solving BVPs associated with indirect methods, there are challenges once the problems get larger. This points to a need to develop a BVP solver that is inherently parallel and can efficiently exploit parallel computational resources for solving large-dimensional problems.

Graphics processing units (GPUs) were originally designed to be used as dedicated processors for rendering three-dimensional graphics on computers. Therefore GPUs are specialized in efficiently running compute-intensive, highly parallel operations especially matrix operations that are required for rendering 3D graphics. CPUs were designed with more transistors dedicated to data caching and flow control, leading to very small latencies as opposed to high throughput. GPUs, in contrast have slower memory access and allows parallel execution of thousands of threads of execution, with some limitations. Hence, a GPU is especially suited to problems which can be expressed as large numbers of data-parallel computations [129], with a high ratio of arithmetic operations to memory operations. These operations should ideally be independent of each other and require very little cross-communication.

In a prior work, a highly parallel indirect optimization strategy for the rapid design of optimal trajectories was developed [79]. The multiple shooting method was used to develop this custom algorithm, *bvpgpu*, that ran very efficiently on a GPU. It was demonstrated that indirect optimization methods can be used to rapidly solve complex optimization problems by utilizing this GPU-accelerated multiple shooting method as shown by the benchmarks in Fig. 3.1. The benchmarks involved solving maximum terminal energy trajectories for a hypersonic vehicle with varying combinations of initial and terminal point constraints as well as path constraints. The test problem in this case was relatively small in terms of number of dimensions (6-24 states). These benchmarks showed a speedup of 2x-4x by using a GPU-based shooting method instead of *bvp4c*.

The multiple shooting algorithm, while not very parallel in its original formulation, has several elements in it that could be modified to make it run fast on a GPU. The computation of the State Transition Matrix (STM) is the most computationally intensive part of the multiple shooting method. At the most basic level, computing the STM involves propagating N^2 extra differential equations for a dynamic system of N equations. On a CPU, algorithms can be parallelized by delegating independent parts of the code to separate threads of execution. GPU computation also involves threads that are conceptually similar, but drastically different in implementation. The naive way of porting the multiple shooting algorithm over to a GPU would involve assigning each equation (from both the original system of equations as well as the STM) to a separate thread on the GPU. While this is very simple to implement, it is also very inefficient. In fact, benchmarking showed that this made the process twice as slow as performing the same operation on a CPU. In order to optimize the code for maximum performance on the GPU, it is necessary to understand how the threads are scheduled and executed by CUDA (NVIDIA's GPU computing library). The various algorithmic optimizations that help maximize GPU performance by accounting for GPU processor occupancy, memory access coalescing, and parallel matrix operations were examined and implemented in our prior work [30], as well as the manner in which the problem is structured is crucial to obtaining optimum performance on a GPU. The methodology was shown to provide significant speedups over using a CPU-based solution method such as MATLAB's *bvp4c* to solve trajectory optimization problems.

However, the multiple shooting method is still not "parallel enough" to scale well as the BVPs become large as in the case of multi-vehicle problems. This is partly because the method was not originally formulated with the express purpose of utilizing parallel computational architectures. In order to obtain the 2x-4x speed-up seen in the benchmarks, the multiple shooting method had to be reformulated to make it more parallel in nature. This motivates the need for developing a numerical method that is inherently parallel and is designed specifically to exploit parallel computing architectures. This chapter will describe the design of a new, scalable, highly parallel numerical method which advances the state-of-the-art for solving large nonlinear boundary value problems.

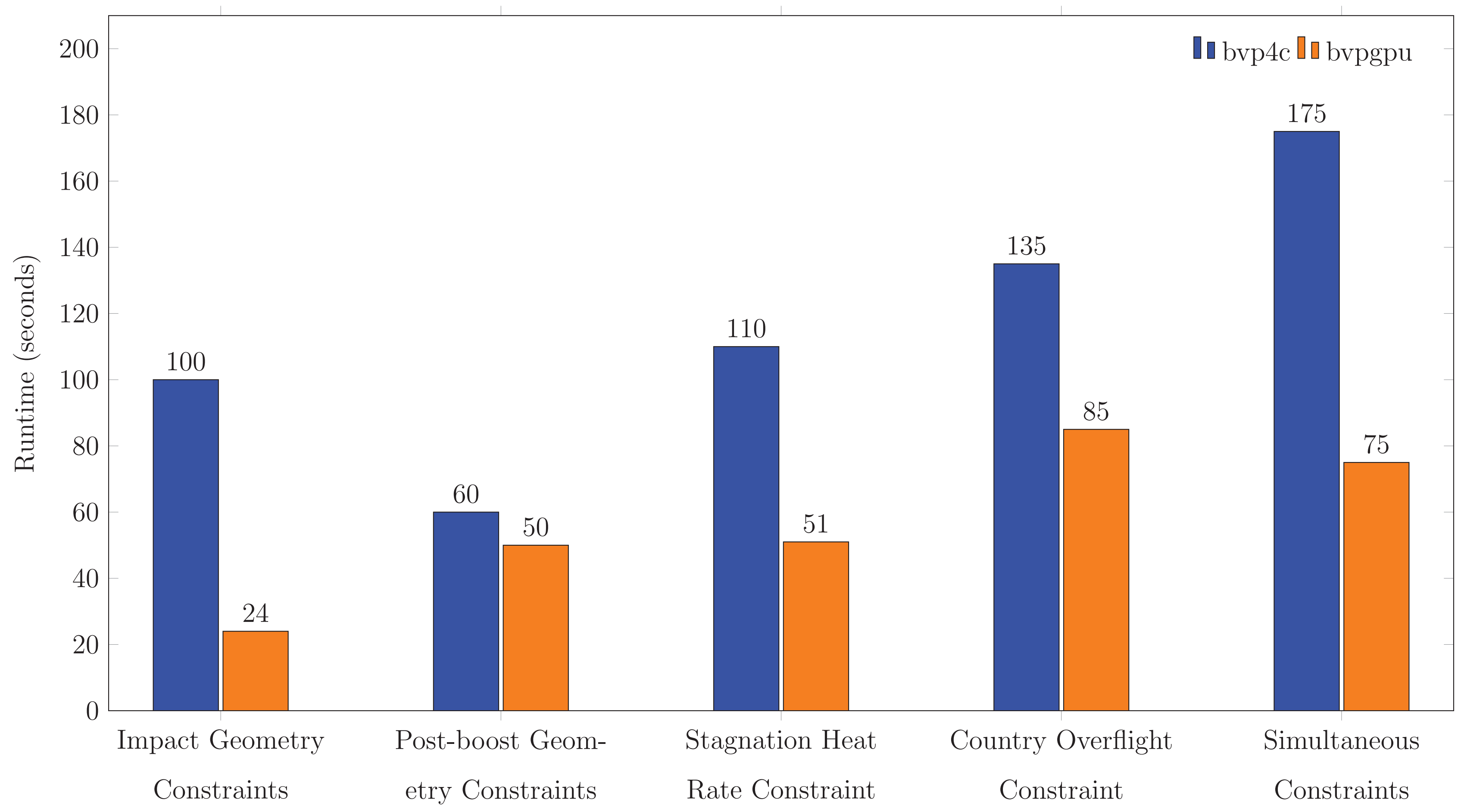


Figure 3.1. Benchmarks – *bvp4c* vs. *bvpgpu* [79]

3.1.2 Picard Iteration

The starting point for the numerical method developed in this dissertation is the Picard iteration, also known as the Picard–Lindelöf theorem [93]. It is a method that was originally used to prove the existence and uniqueness of solutions to first-order differential equations with a given set of initial conditions.

$$\begin{aligned} y\ &= f(t, y(t)),\ y(t_0) = y_0 \\ \phi_0 &= y_0 \\ \phi_{k+1} &= y_0 + \int_{t_0}^{t} f(s, \phi_k(s))ds, \text{ for iteration } k \end{aligned} \tag{3.1}$$

The Picard-Lindelöf Theorem shows that this series summation (ϕ_n) in Eq. (3.1) converges to $y(t)$ at the limit [93]. An example of the application of this theorem is shown in Eq. (3.2) for a simple first-order initial value problem (IVP), $y\ = f(t, y(t)) = -y; y(t_0) = 1.0$.

$$y\ = f(t, y(t)) = -y \tag{3.2a}$$

$$\phi_0 = y_0 = 1.0 \tag{3.2b}$$

$$\phi_1 = y_0 + \int_{t_0}^{t} -1\ ds = 1 - t \tag{3.2c}$$

$$\phi_2 = y_0 + \int_{t_0}^{t} (-1 + s)\ ds = 1 - t + \frac{t^2}{2} \tag{3.2d}$$

$$\phi_3 = y_0 + \int_{t_0}^{t} (-1 + s)\, ds = 1 - t + \frac{t^2}{2} - \frac{3t^3}{6} \tag{3.2e}$$

It can be seen that this series converges to the analytical solution of the system at the limit as: $y(t) = 1 - t + \frac{t^2}{2} - \frac{3t^3}{6} + ... = \exp(-y)$. This iteration forms the core of the Modified Chebyshev Picard Iteration algorithm.

3.1.3 Chebyshev Polynomials

Chebyshev polynomials [130] are a complete set of orthogonal polynomials that are commonly used for function approximation. There are two kinds of Chebyshev polynomials. For convenience, Chebyshev polynomials of the first kind are referred to as simply Chebyshev polynomials in this work. These polynomials are defined through a recurrence relation:

$$T_0(x) = 1 \tag{3.3}$$

$$T_1(x) = x \tag{3.4}$$

$$T_{k+1}(x) = 2\tau T_k(x) - T_{k-1}(x) \tag{3.5}$$

where T_k represents the k-th order Chebyshev polynomial. They may also be computed using a trigonometric relation, $T_k(x) = \cos(k \arccos x)$ where $x \in [-1, 1]$. Chebyshev polynomials $T_k(x)$ up to $k = 5$ are shown in Figure 3.2.

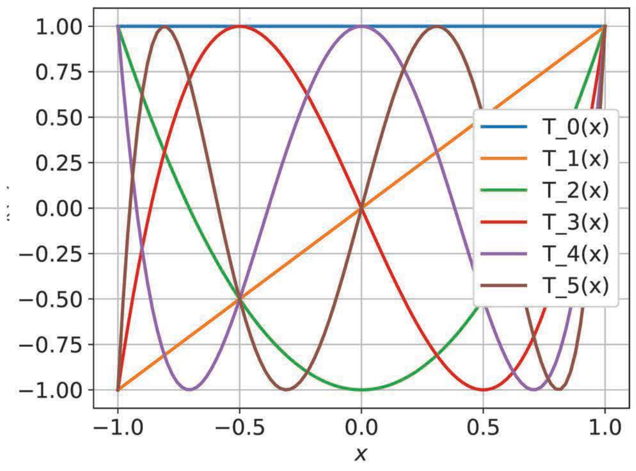


Figure 3.2. Chebyshev Polynomials up to $k = 5$

The zeros of these polynomials are called Chebyshev-Gauss-Lobatto (CGL) nodes. The N+1 CGL nodes for an Nth order Chebyshev Polynomial can be calculated as:

$$x_k = \cos \frac{k\pi}{n} \;, \qquad k = 0, 1, 2, ..., \; N + 1 \tag{3.6}$$

When these nodes are used for polynomial interpolation, Runge's phenomenon is minimized, and the best function approximation under the minimax norm can be obtained [131, 132]. Ref. 131 shows that if a smooth function $f(\tau)$ is approximated by an N-th order Chebyshev polynomials as $f(\tau) \approx \sum_{k=0}^{N} \alpha_k T_k(\tau)$, the coefficients α_k can be computed as:

$$\alpha_k = \frac{2}{N} \sum_{j=0}^{N} f(\tau_j) T_k(\tau_j), \qquad k = 0, 1, ..., N \tag{3.7}$$

The " in the summation denotes that the first and the last terms in the summation are to be halved. The integral of Chebyshev polynomials is defined by:

$$T_k(x) = \frac{1}{2} \quad \frac{T_{k+1}}{k+1} - \frac{T_{k-1}}{k-1} \tag{3.8}$$

The relation in Eq. (3.8) forms the basis for numerical methods that use Chebyshev polynomials to solve differential equations. There are many past works describing such methods that solve initial value problems and boundary value problems [133–139]. There are also some direct methods for solving optimal control problems using Chebyshev polynomials [43, 140, 141]. In the next section, some of these methods that combine the Picard Iteration with Chebyshev polynomials to solve IVPs and BVPs are examined.

3.1.4 Chebyshev-Picard Methods

Clenshaw's work in Ref. 134 is one of the first works where the Picard iteration was combined with Chebyshev polynomials to create a practical numerical method for solving IVPs and BVPs. This method was later applied to several astrodynamics problems involving interplanetary trajectories by Feagin [142]. The suitability of this Chebyshev-Picard method for efficient implementation on parallel processors was examined by Shaver [143] by using it to create a parallel orbit propagation algorithm and a parallel orbit estimation algorithm. A vector-matrix formulation of the same algorithm was designed by Feagin in Ref. 144 but no experimental results were shown. This was the precursor to the method used in MCPI which forms part of the numerical method developed in this dissertation. Ref. 145 shows an implementation of the Chebyshev-Picard method on a vector computer. However, this implementation was in some cases slower than the scalar version of the code owing to some overheads and inefficiencies. The Modified Chebyshev Picard Iteration algorithm [31] built on these existing works and created a unified matrix-vector method for solving both IVPs and certain classes of BVPs. MCPI was shown to be capable of solving several important celestial mechanics problems. The original work also showed techniques for improving the convergence domain of Chebyshev-Picard methods.

Modified Chebyshev-Picard Iteration Method

The Modified Chebyshev Picard Iteration (MCPI) method is a numerical method for solving Initial Value Problems (IVPs) and certain classes of Boundary Value Problems (BVPs) without directly propagating the equations of motion or evaluating gradients [31]. MCPI is based on the Picard iteration method described in Section 3.1.2. The algorithm represents the integrand in Eq. (3.1) as a weighted sum of Chebyshev polynomials of sufficiently high order. The integration step of Picard iteration is performed using the quadrature rule in Eq. (3.8). The coefficients of the Chebyshev polynomials representing the solution is solved for based on the boundary conditions of the problem.

While this iteration can also be implemented using other orthogonal polynomial sets such as Legendre polynomials, this particular formulation of Chebyshev polynomials was chosen because it is possible to fit a function to these polynomials without solving a linear algebra problem [131]. As shown before in Eq. (3.7), the calculation of polynomial coefficients on a Chebyshev mesh for a given function is a long summation operation which can be reformulated into a simple matrix multiplication operation [31].

The MCPI algorithms for initial value problems (MCPI-IVP) and boundary value problems (MCPI-BVP) are very similar in their implementation. The simpler of the two, MCPI-IVP, using Chebyshev polynomial series of order N, is summarized below.

For a given dynamic system,

$$\dot{\mathbf{y}} = \Phi(\mathbf{y}, t) \qquad \mathbf{y}(0) = \mathbf{y}_0 \tag{3.9}$$

a scaled version, ϕ, is formulated which can be evaluated between -1 and 1.

$$\phi(\mathbf{y}, t) = \frac{T}{2}\Phi(\mathbf{y}, \frac{T}{2}\tau + \frac{T}{2}) \tag{3.10}$$

The solution is evaluated over Chebyshev-Gauss-Lobatto (CGL) mesh points of a given order, N, that are defined by:

$$\tau_j = \cos{(j\pi/N)}, \qquad j = 0, 1, ..., N \tag{3.11}$$

The main steps in the algorithm are as follows. First, the dynamic equations are evaluated over the CGL mesh, and the coefficients for the Chebyshev polynomials, F_k, corresponding to these equations are calculated.

$$F_k = \frac{2}{N} \overset{N}{\underset{j=0}{\quad}} \phi(t, \mathbf{y}_k)\ T_k(\tau_j) \tag{3.12}$$

Each coefficient F_k is obtained through the summation of $N + 1$ terms, each involving the product of the scaled dynamic equations ϕ and the Chebyshev polynomial T_k evaluated at node τ_j. By applying the Picard iteration and the integration rule from Eq. (3.8) to Eq. (3.12), the dynamic equations are integrated to obtain the coefficients, β_k $(k = 0, 1, 2..., N)$, corresponding to the solution.

$$\beta_r = \frac{1}{2r}\left(F_{r-1} - F_{r+1}\right), \qquad r = 1, 2, ..., N-1 \tag{3.13}$$

$$\beta_N = \frac{F_{N-1}}{N} \tag{3.14}$$

$$\beta_0 = 2y_0 + \overset{N}{\underset{j=1}{}} (-1)^{j+1}\beta_j \tag{3.15}$$

$$\mathbf{y}_{k+1}(\tau) = \overset{N}{\underset{j=0}{}} \beta_j T_j(\tau) \tag{3.16}$$

The algorithm starts with an initial guess for the entire solution expressed on a Chebyshev-Gauss-Lobatto grid and continues until the change in $\mathbf{y}_k$ is less than a desired tolerance. The original author showed that the operations involved in calculating β_k, as well as computing the solution from β_k in each iteration can be expressed as a series of matrix-vector operations [31]. The steps in Eqs. (3.12)-(3.16) then condense to the following form:

$$\beta = C_\alpha \phi(\mathbf{y}_k, \tau) + \mathbf{y}_0 \tag{3.17}$$

$$\mathbf{y}_{k+1} = C_x \beta \tag{3.18}$$

C_x and C_α are constant matrices for a given order of Chebyshev polynomials. The overall structure of these matrices are given in Appendix A. These matrices can be computed and cached before the iteration process begins, and hence the "integration" consists entirely of matrix-vector multiplication operations. Such operations are ideal for parallel implementation such as on a GPU or multi-core CPUs for accelerated processing. A GPU implementation of MCPI-IVP is shown in Ref. 146, where it was used for high-precision parallel orbit propagation.

The BVP version of the algorithm specifies a different update equation for each state depending on whether it is constrained at the initial point, terminal point, or both [91]. If both the initial and terminal values are given for a state, the update equations for β are changed as shown below:

$$\beta_r = \frac{1}{2r}\left(F_{r-1} - F_{r+1}\right), r = 2, 3, ..., N-1 \tag{3.19a}$$

$$\beta_N = \frac{F_{N-1}}{N} \tag{3.19b}$$

$$\beta_0 = y_0 + y_f - 2\left(\beta_2 + \beta_4 + \beta_6 + ...\right) \tag{3.19c}$$

$$\beta_1 = \frac{y_f - y_0}{2} - \left(\beta_3 + \beta_5 + \beta_7 + ...\right) \tag{3.19d}$$

An astrodynamics trajectory problem was solved in the original work using this method, and its performance and solution quality were compared to a direct pseudospectral method. Significant speedups were obtained over direct methods, and it was also shown that the method can derive huge benefits from implementation on GPU computing architectures.

However, there is a significant drawback when it comes to using this algorithm for solving BVPs arising in trajectory optimization problems. The MCPI-BVP formulation assumes that every state in the problem has at least one boundary condition defined for it. If the BVP does not define a boundary condition for a particular state, a boundary condition has to be derived for it from other domain-specific information available in the problem, if any. This is only possible for a limited class of problems such as the astrodynamics problems demonstrated in the original implementation [31]. The algorithm also assumes that the boundary conditions are simple equality constraints at the initial and terminal points. This was the motivation for the development of a more generalized MCPI-BVP algorithm that is capable of handling general nonlinear boundary conditions such as those encountered in optimal control problems.

3.1.5 A Generalized MCPI-BVP Algorithm

A more generalized formulation of the MCPI-BVP method will enable the fast computation of optimal trajectories for large dimensional problems. Initially, a version of MCPI was created which linearized the boundary conditions and tried to solve the polynomial coefficients by solving a linear system [147]. The original MCPI algorithm was formulated by assuming a set of fixed boundary conditions of the form shown in Eq. (3.20) and solving a linear system analytically to obtain the expressions for the Chebyshev coefficients β_k.

$$y(t_0) = y_0, \vec{\ } \; y(t_f) = y_f \tag{3.20}$$

In order to formulate a more generalized version of this method, it is necessary to start with a more generic boundary condition function such as the one described by Eq. (3.21) for a two-point boundary value problem.

$$\begin{aligned} &b\left(y(t_0), \vec{y}(t_f)\right) = 0 \\ &\text{Subject to: } \frac{d\vec{y}}{dt} = f(t, y(t)) \end{aligned} \tag{3.21}$$

The boundary conditions are linearized in Eq. (3.22) and combined with the expressions for initial and terminal states, Eq. (3.23), in order to obtain the Chebyshev coefficients (β_k) of the solution as shown in Eq. (3.24).

$$\begin{aligned} b \approx \quad & \mathcal{M} \times (y(t_0) - y_0) + \mathcal{N} \times (y(t_f) - y_f) + b(y_0, \vec{y}_f) \\ \text{where } \mathcal{M} = & \frac{\partial b}{\partial \vec{y}_0},\ \mathcal{N} = \frac{\partial b}{\partial \vec{y}_f} \end{aligned} \tag{3.22}$$

$$\begin{aligned} x(t_0) =& \frac{\beta_0}{2} - \beta_1 + \overset{N}{\underset{k=2}{\ }} (-1)^{k+1} \beta_k \\ x(t_f) =& \frac{\beta_0}{2} + \beta_1 + \overset{N}{\underset{k=2}{\ }} \beta_k \end{aligned} \tag{3.23}$$

$$
\begin{aligned}
x(\tau) \approx & \sum_{k=0}^{N} \overrightarrow{\beta}_k T_k(\tau), \quad \text{where} \\
\beta_r =& \frac{1}{2r}(F_{r-1} + F_{r+1}), \; r = 1, 2, ..., N-1 \\
\beta_N =& \frac{F_{N-1}}{2N} \\
\frac{\mathcal{M}+\mathcal{N}}{2}, \mathcal{N}-\mathcal{M} \begin{pmatrix} \overrightarrow{\beta_0} \\ \overrightarrow{\beta_1} \end{pmatrix} =& \; \mathcal{M} \times (y_0 - \sum_{k=2}^{N} (-1)^{k+1} \beta_k) \\
& + \mathcal{N} \times (y_f - \sum_{k=2}^{N} \beta_k) - b(y_0, \vec{y}_f)
\end{aligned}
\tag{3.24}
$$

Eq. (3.24) outlines one way to incorporate non-linear boundary conditions into the MCPI algorithm. In Ref. 147, this algorithm was demonstrated using the Brachistochrone problem. However, it was found that it was not capable of solving problems with more numerical sensitivity such as hypersonic optimal control problems. This prompted the search for a different approach to solving boundary value problems that complements the drawbacks of MCPI-BVP and can be combined with MCPI to create a more general numerical method.

Modified Quasi-Linearization Algorithm

The method of particular solutions for solving linear two-point boundary value problems is described by Miele in Ref. 148. The boundary-value problem is solved by linearly combining several particular solutions of the original differential system. This method was further expanded to include some classes of nonlinear problems in Ref. 149 with nonlinear dynamic equations. The modified quasi-linearization algorithm (MQA) [108, 150] is a further refinement of the method of particular solutions that allows nonlinear boundary conditions at the terminal point. Ref. 151 explores the use of MQA for solving optimal control problems. In the modified quasi-linearization algorithm, the known initial conditions and guesses for the unknown initial states are used to generate a reference solution using numerical integration. Then, small, linearly independent perturbations of the unknown states are also propagated using a numerical integrator. The resulting perturbations at the terminal point are used to compute corrections for the for the entire state history until all the boundary conditions are satisfied.

Consider a nonlinear dynamic system with n states as follows:

$$\dot{\mathbf{y}} = \phi(\mathbf{y}, t), \qquad 0 \leq t \leq t_f \tag{3.25}$$

with the initial conditions,

$$b_{0j}(\mathbf{y}(t_0)) = 0 \qquad j = 1, 2, ..., p \tag{3.26}$$

and terminal conditions,

$$b_{fj}(\mathbf{y}(t_f)) = 0 \qquad j = 1, 2, ..., q \tag{3.27}$$

Taking a first order approximation of b_f,

$$\frac{\partial b_f(\mathbf{y}(t_f))}{\partial \mathbf{y}} \Delta\mathbf{y}(t_f) + b_f(\mathbf{y}(t_f)) = 0 \tag{3.28}$$

Let $\mathbf{A}_j(t)$ denote the perturbations from the reference solution for a small perturbation in a free initial state. $\mathbf{A}_j(t_f)$ is computed for $q+1$ perturbed initial conditions to form the linear combination:

$$\mathbf{A}(t) = \mathop{\qquad}\limits_{j=1}^{q+1} k_j \mathbf{A}_j(t) \tag{3.29}$$

Ref. 149 shows that this linear combination satisfies the system in Eq. (3.28). Therefore the coefficients k_j can be computed by solving the following linear system:

$$\mathop{\qquad}\limits_{j=1}^{q+1} k_j = 1 \qquad \psi_{\mathbf{y}}(\mathbf{y}(t_f)) \mathop{\qquad}\limits_{j=1}^{q+1} k_j \mathbf{A}_j(t) + \psi(\mathbf{y}(t_f)) = 0 \tag{3.30}$$

The correction for the solution is computed as shown in Eq. (3.31). This correction is applied not just to the initial state, but to the entire state history. This feature makes it apt for inclusion into a method like MCPI where unlike a shooting method, the entire trajectory is approximated at all times.

$$\Delta y(t) = \alpha \mathop{}\limits_{j=1}^{q+1} k_j \mathbf{A}_j(t) \quad \text{where } 0 \leq \alpha \leq 1 \tag{3.31}$$

The step-size, α can be determined by a one-dimensional line-search of the performance index, P, defined in Eq. (3.32), the cumulative error in the differential equations and the boundary conditions. Ref. 149 proves the use of this performance index gives the algorithm its *descent property*: If the step-size, α, is sufficiently small, the reduction in P is guaranteed. The search is started with $\alpha = 1.0$ and continues until $P(\alpha) < P(0)$.

$$P(\alpha) = \mathop{}\limits_{0}^{T} (\dot{\mathbf{y}} - \phi)^T (\dot{\mathbf{y}} - \phi) \; dt + b_f^T b_f + b_0^T b_0 \tag{3.32}$$

A recent work [92] examined the use of the method of particular solutions (MPS) [149] along with MCPI for computing perturbed orbits of orbital debris by solving Lambert's problem. The current work is focused on expanding this to include the ability to solve optimal control problems using a hybrid method that uses both MCPI and MQA, called the Quasi-Linear Chebyshev-Picard Iteration (QCPI) algorithm.

3.2 QCPI Algorithm Implementation

The Quasi-Linear Chebyshev-Picard Iteration (QCPI) method leverages MCPI and the modified quasi-linearization algorithm to solve nonlinear two-point boundary value problems such as those arising in trajectory optimization. It incorporates MCPI as the IVP integrator and uses MQA to perform solution updates for the free parameters in the problem.

The algorithm is designed to solve a general nonlinear two-point boundary value problem of the following form:

$$\dot{\mathbf{x}} = \phi(t, \mathbf{x}) \tag{3.33a}$$

$$\boldsymbol{b_0}(\mathbf{x}(0), 0) = 0 \tag{3.33b}$$

$$\boldsymbol{b_f}(\mathbf{x}(T), T) = 0 \tag{3.33c}$$

The solution is approximated using a Chebyshev Polynomial series of order N, with separate coefficients for each state. The algorithm consists of the following steps.

1. Define the matrices C_a and C_x as well as the independent variable mesh $\tau \in [-1, 1]$ for the given value of N. The structure of C_a and C_x are detailed in Appendix A.

2. For the initial state, x^0, The perturbed initial states, x_p are initialized as:

$$A_j = \delta_{ij}\Delta x_i, \quad \text{for } i, j = 1, 2..., n \tag{3.34}$$

$$\mathbf{x}_p(0) = [\mathbf{x}^0 + A_0, \mathbf{x}^0 + A_1, ..., \mathbf{x}^0 + A_n] \tag{3.35}$$

   where δ_{ij} is the Kronecker delta function, and n is the number of ODEs in the BVP. $\mathbf{x}_p$, is a row vector of size n^2.

3. The initial guess matrix, x_{guess}, is initialized. This is either using the value from a previous iteration or by calling a separate MCPI-IVP integrator to propagate the equations of motion with the actual initial state, x^0, along with the perturbed states, $x_p(0)$, to generate x_{guess} for the given value of N. This combined state vector will now be denoted as X and contains the original state vector followed by state vectors with each state perturbed one at a time.

$$X(0) = \begin{matrix} \mathbf{x}(0) & \mathbf{x}(0) + A_0 & \mathbf{x}(0) + A_1 & ... & \mathbf{x}(0) + A_n \end{matrix} \tag{3.36}$$

4. Evaluate the dynamic equations of the BVP at every point of the CGL mesh, for both the original and the perturbed initial conditions. The results are stored in the matrix $\boldsymbol{\Phi}$ which has the same dimension as x_{guess}.

$$\Phi = w_1 \begin{bmatrix} \phi(\tau_0, \mathbf{x}(\tau_0)) \\ \phi(\tau_1, \mathbf{x}(\tau_1)) \\ \vdots \\ \phi(\tau_N, \mathbf{x}(\tau_N)) \end{bmatrix}_{n^2 \times (N+1)} \qquad \text{where } w_1 = T/2 \tag{3.37}$$

The above computation assumes that the function $\phi(\tau, x)$ returns a row vector of length n.

5. The derivative information in Φ is fit to a Chebyshev polynomial series of order N. By using the Matrix-Vector form described by Feagin [31,144] and Bai [31], the computation of the polynomial coefficients representing the solution, β, is done with a simple matrix multiplication operation:

$$\beta = 2\chi_0 + C_a \times \Phi_i \quad \text{where } \chi_0 = \begin{bmatrix} X(0) \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{(N+1)\times 1} \tag{3.38}$$

6. The solution and the guess for the next iteration, x_{new}, is obtained as:

$$x_{new} = C_x \times \beta \tag{3.39}$$

7. The change in x is calculated as $e = x_{new} - x_{guess}$. If the L2-norm of e is greater that the required tolerance, skip to step 13.

8. Once e is under the required tolerance, the integration is complete, and x_{new} contains dynamically feasible unperturbed and perturbed trajectories. The perturbations due to the change in each state is used to update the solution to satisfy the nonlinear boundary conditions. The Jacobian matrices for the initial and terminal boundary conditions are calculated using finite-difference methods. The residual error in the boundary conditions are also evaluated.

$$\mathbf{r}_0 = \boldsymbol{b_0}(0, \mathbf{x}(0)) \tag{3.40}$$

$$\mathbf{r}_f = \boldsymbol{b_f}(T, \mathbf{x}(T)) \tag{3.41}$$

$$\mathbf{b}_{0x} = \frac{\partial b_0}{\partial \mathbf{x}}|_{\mathbf{x}=\mathbf{x}(0)} \tag{3.42}$$

$$\mathbf{b}_{fx} = \frac{\partial b_f}{\partial \mathbf{x}}|_{\mathbf{x}=\mathbf{x}(T)} \tag{3.43}$$

9. If the L2-norms of residual errors $\mathbf{r}_g$ and $\mathbf{r}_f$ are below the desired tolerance, the solution for the BVP has converged and the iteration process can be stopped.

10. The required correction for the solution to reduce the residual errors to zero is assumed to be a linear combination of all the initial perturbations in state, or $\Delta x = \;{}_{j=0}^{n+1} K_j A_j(T)$, where $A_j(t)$ is the perturbation at t when starting with a perturbation in the j-th state. It is to be noted that A_0 is a zero-vector which is used to add an additional constraint on the coefficients K_j that $\;{}_{j=0}^{n+1} K_j = 1$. The coefficients of this linear combination, $\mathbf{K}$, are calculated by solving the following linear system.

$$\begin{bmatrix} 1 & \cdots\cdot & 1 \\ & \mathbf{b}_{0x} \times A_j(0) & \\ & \mathbf{b}_{fx} \times A_j(T) & \end{bmatrix} \underbrace{\begin{bmatrix} K_1 \\ K_2 \\ \vdots \\ K_{n+1} \end{bmatrix}}_{\mathbf{K}} = \begin{bmatrix} 1 \\ -\mathbf{r}_0 \\ -\mathbf{r}_f \end{bmatrix} \tag{3.44}$$

11. The direction of the state-correction vector is given by $\Delta\mathbf{x}(t) = {}_{j=0}^{n+1} K_j A_j(t)$. Ref. 148 shows that a sufficiently small step-size α applied to this direction vector will reduce the residual error, leading to convergence. α is found by performing a line-search on the performance index, $P(\alpha) = ||\boldsymbol{b_f}(x(T) + \alpha\Delta x(T))||_2 + ||\boldsymbol{b_0}(x(0) + \alpha\Delta x(0))||_2$, to find a value of α such that $P(\alpha) < P(0)$. This value is then selected as the step-size for the iteration.

12. The solution is updated using the correction vector as $\mathbf{x}(t) = \alpha\Delta x(t)$.

13. x_{guess} is replaced with x_{new}. Repeat from Step 4 until convergence criteria is satisfied or maximum number of iterations exceeded.

The algorithm is summarized in the flowchart in Figure 3.3.

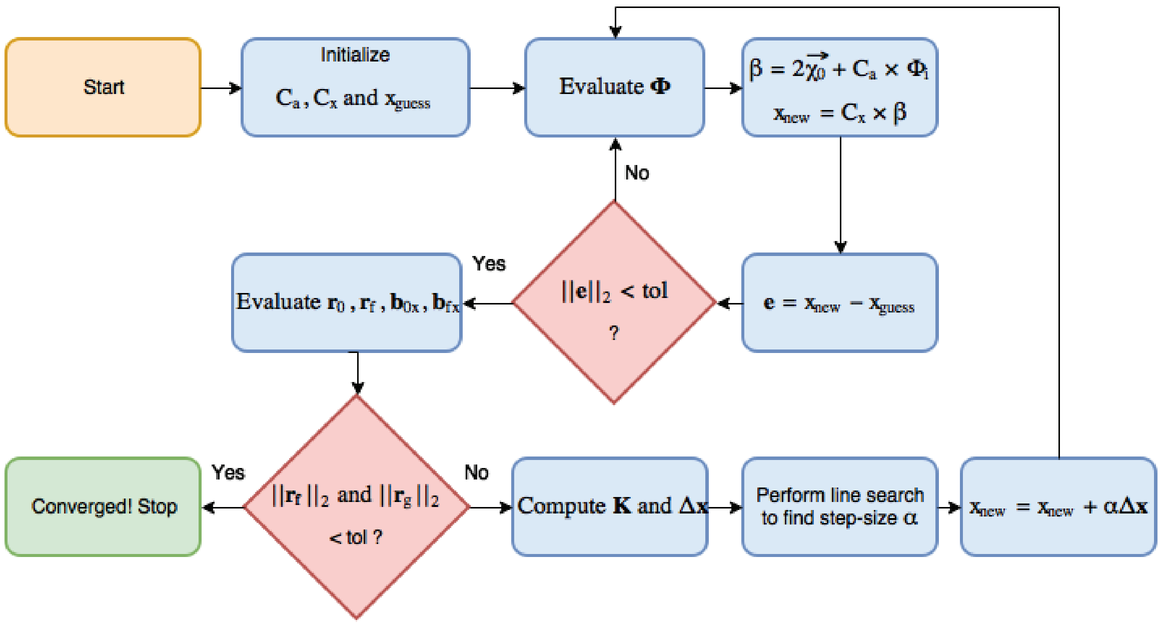

Start
Initialize
C_a, C_x and x_{guess}
Evaluate Φ
$\beta = 2\vec{\chi_0} + C_a \times \Phi_i$
$x_{new} = C_x \times \beta$
No
$e = x_{new} - x_{guess}$
$||e||_2 < tol$?
Yes
Evaluate r_0, r_f, b_{0x}, b_{fx}
$||r_f||_2$ and $||r_g||_2$ < tol ?
Yes
Converged! Stop
No
Compute K and Δx
Perform line search to find step-size α
$x_{new} = x_{new} + \alpha\Delta x$

Figure 3.3. QCPI Algorithm Implementation – Flowchart

3.3 Acceleration using *Numba* Just-In-Time (JIT) Compiler

Parallelization of computational methods is generally a very time-consuming task that requires careful organization of data-parallel operations and creation of special data-structures required for exploiting parallel computation architecture. Prior work [30] explored in detail an efficient GPU implementation of the multiple shooting method in MATLAB. In contrast, QCPI is parallelized in a more automated manner using the JIT compiler *Numba* [106]. *Numba* helps speed up computation-heavy code written in Python by compiling it to high-performance native machine code with speeds comparable to C/FORTRAN without having to switch languages or Python interpreters. Numba is based on the LLVM (Low-Level Virtual Machine) compiler which can inspect and analyze code on-the-fly and generate optimized native machine code. It is designed to work with multi-core CPUs or GPUs and can integrate directly with the Python scientific software stack such as NumPy and SciPy.

Numba supports three different compiler modes:

- Python JIT mode which allows the use of Python data structures such as dictionaries and objects and is the slowest of all three. This option includes a compilation overhead the first time the code is executed.
- *nopython* JIT mode – this restricts the types of variables that can be included in a function. This mode can achieve performance close to C or FORTRAN native code. Since it is "just-in-time" compiled, there is an added overhead the first time the code is executed.
- Ahead-Of-Time mode – This compiles code into machine-specific binary ahead of time and can be used later without Numba.

For QCPI, the *nopython* JIT mode was used. Both of the JIT compilation modes also support automatic parallelization of certain types of loops, as long as the loop does not have cross-iteration dependencies (with some exceptions). This is an extremely useful feature when calculating Jacobian matrices and when evaluating equations with multiple sets of perturbed states. Each individual iteration is run on separate CPUs in parallel at close to C/FORTRAN speeds by just adding some annotations to the Python code. In one case, the use of *nopython* JIT mode gave a speed up of nearly 60x over pure Python code.

3.4 Validation

The QCPI solver is validated by testing it on some representative optimal control problems with known solutions. The results are compared to those obtained using a multiple shooting algorithm.

3.4.1 Classical Brachistochrone Problem

The classical Brachistochrone problem is the minimum-time problem described in Section 2.8.1 but without the path constraint. It is used for validation as it is one of the simplest nonlinear optimal control problems with a known solution.

$$\text{Min } T \tag{3.45a}$$

$$\text{Subject to :}$$

$$\dot{x} = v\cos\theta \tag{3.45b}$$

$$\dot{y} = v\sin\theta \tag{3.45c}$$

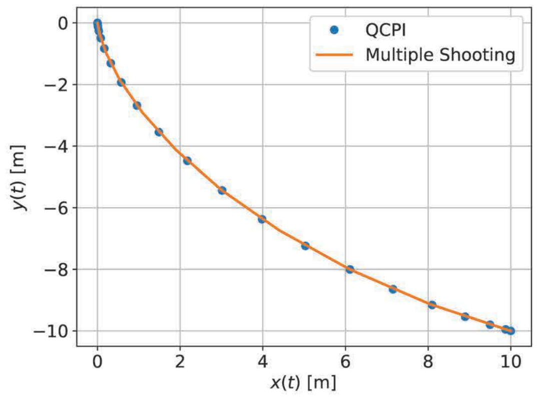
$$\dot{v} = g\sin\theta \tag{3.45d}$$

$$x(0) = y(0) = 0, x(T) = -y(T) = 1 \tag{3.45e}$$

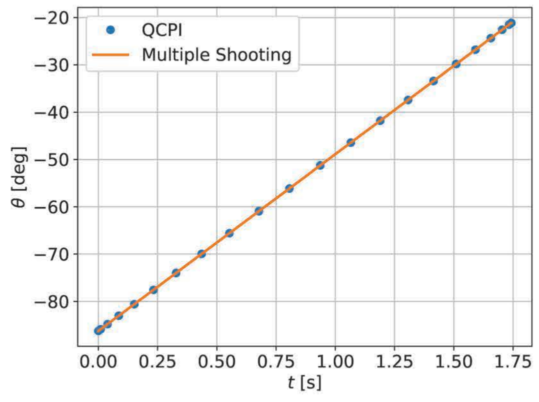
$$g = -9.81 \tag{3.45f}$$

where θ is the control.

The initial guess was created by propagating the equations of motion form the initial conditions with a fixed initial values (=-0.1) for the costates for 0.1 seconds.

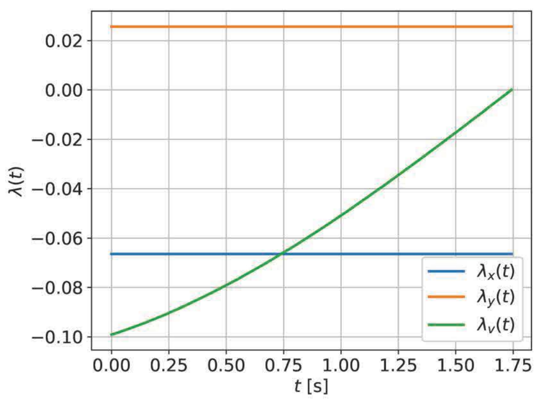


(a) Trajectory

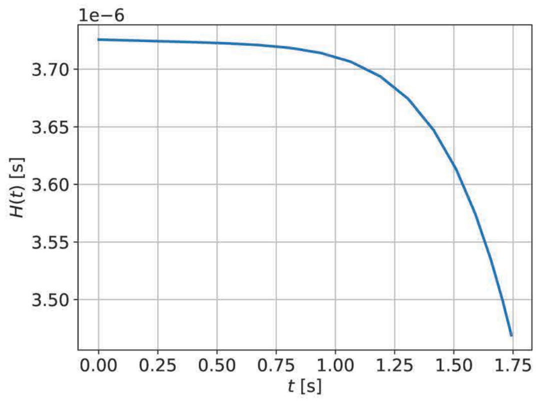


(b) Control History

Figure 3.4. QCPI Validation - Classical Brachistochrone Problem



(a) Costates



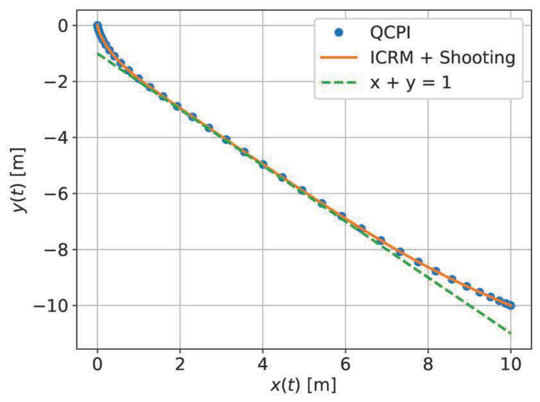
(b) Hamiltonian

Figure 3.5. QCPI Validation - Classical Brachistochrone - Optimality Conditions

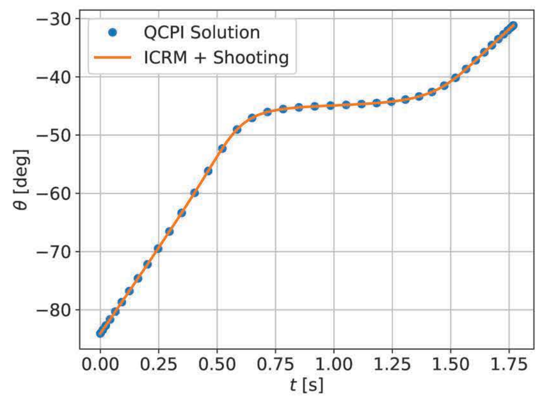
The terminal conditions were updated to the design value of $(x, y) = (10, -10)$ over 11 continuation steps. The converged trajectory and control history are shown in Fig 3.4. The optimality of the solution is verified in Fig 3.5. The costate profiles in Figure 3.5(a) match the necessary conditions of optimality with λ_x and λ_y being constant, and $\lambda_v(T)$ being equal to zero as $v(T)$ is unconstrained. Fig 3.5(b) shows that the Hamiltonian remains very close to zero as it should for the optimal solution.

3.4.2 Constrained Brachistochrone Problem

The same problem described in Section 2.8.1 is solved using QCPI after regularizing the constraint using ICRM. The initial guess was generated in a similar manner to how it was done in Section 2.8.1. However, a key difference in this case was that QCPI is able to solve the constrained problem starting with a low value of $\epsilon = 10^{-4}$ for the regularization parameter. The shooting method required that ϵ be defined as one in the beginning and then reduced later using a continuation strategy. This factor also contributed to the fast convergence time for QCPI. The result is compared to that obtained using ICRM and the shooting method in Figure 3.6 in order to validate it.



(a) Trajectory



(b) Control History

Figure 3.6. QCPI Validation - Constrained Brachistochrone Problem with $\epsilon = 10^{-4}$

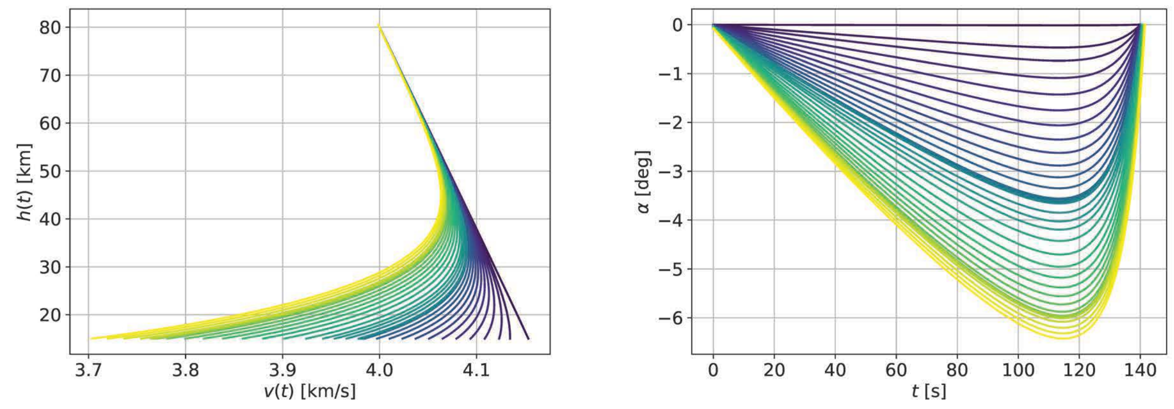
3.4.3 Unconstrained Maximum Terminal Energy Hypersonic Trajectory

In this scenario, a modified version of the hypersonic trajectory problem from Section 2.8.3 is solved using QCPI. All the staging conditions are the same as those specified in Table 2.1 from Chapter 2 except for the flight-path angle which is left unconstrained. The terminal boundary condition was changed to have the vehicle fly a longer distance of 566 km downrange, corresponding to a longitude of 5 degrees.

The same problem was also solved using the shooting method. This is a particular example where QCPI in its current form performs worse than the shooting method even after parallelization. The reason for this is that in its current form, QCPI uses a CGL mesh of a fixed size and structure that is set before starting the solution process. The shooting method on the other hand, uses an adaptive numerical integrator method, Runge-Kutta-Fehlberg-45, which is able to adaptively select the mesh size based on the sensitivity of the dynamic equations. Due to this reason, regions of high numerical sensitivity towards the middle of the trajectory tends to cause the solver to diverge. Therefore, while both methods required the use of continuation, starting with a trivial initial guess, it is to be noted that the continuation strategy used for QCPI was different from that used for the shooting algorithm.

In case of the shooting algorithm, continuation was performed only on the boundary conditions of the problem as described in Section 1.3. However, for QCPI the problem initially had to be solved with the atmospheric density parameter, ρ_0, set to a very low value of 0.0012 kg/m^3 (0.1% of the actual value). Once the near-ballistic trajectory connecting the starting and ending points is solved, ρ_0 was increased up to its actual value of 1.2 kg/m^3. The majority of the time of the solution process is spent on changing ρ_0 to its actual value. This extra step is required due to the limitation of QCPI when dealing with problems with high-sensitivity regions near the middle of the trajectory as described in Section 3.6.1. By solving the trajectory first with very low atmospheric density, it is possible to avoid intermediate trajectories with high sensitivity regions. It is in fact, possible to solve such problems with the current implementation of QCPI if the number of nodes are significantly increased, at the cost of very high computation time. This is address further in the next chapter in Section 4.3.4.

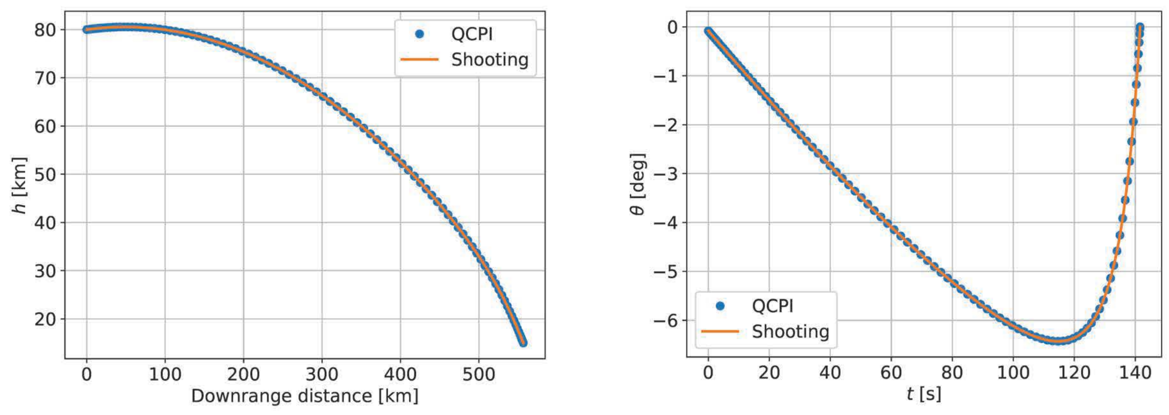
The evolution of the trajectory and control history with changing ρ_0 is shown in Figure 3.7. The control effort increases as the atmospheric density increases so that the vehicle can utilize lift to fly higher. There is also a significant decrease in the terminal velocity due to atmospheric drag.



(a) Energy Plot

(b) Control History

Figure 3.7. QCPI Validation - Unconstrained Hypersonic Trajectory Problem - Continuation in ρ_0



(a) Energy Plot (b) Control History

Figure 3.8. QCPI Validation - Unconstrained Hypersonic Trajectory Problem

3.5 Benchmark Problem and Performance Comparison

3.5.1 Problem Definition

A large multi-vehicle problem is set up for testing the performance of QCPI against that of shooting methods. The problem is based on the application problem detailed in Chapter 2, but uses a two-dimensional model in place of the 3-DOF system. There are also no path constraints enforced in the problem, and instead the number of vehicles are increased from 2 to 25. The motivation for doing this is to demonstrate scalability of this method as the number of dynamic variables in the BVP increases. The initial positions of the vehicles are linearly spaced out between ± 2.25 km and the impact heading are linearly distributed between ± 45 degrees.

3.5.2 Test Setup

The benchmarking was performed on a computer with the specifications given in Table 3.1. The performance of QCPI is compared against three separate implementations of the shooting method.

1. A single shooting method, with automatic parallelization and code-acceleration using the *Numba* Just-In-Time compiler.

2. A parallel shooting solver with an explicit multi-core implementation. *Python*'s built-in parallel computing primitives are used to propagate each trajectory arc and its sensitivity matrix in parallel on separate CPU cores. The dynamic equations are still compiled to binary using *Numba* in this implementation.

3. A parallel shooting solver, with automatic parallelization and code-acceleration using the *Numba* Just-In-Time compiler.

Both the single and multiple shooting solvers use an explicit Runge-Kutta 4(5) adaptive integration algorithm based on the DOPRI5 code described in Ref. 152. A 21-st order Chebyshev Polynomial Series was used for function approximation in QCPI. All the four solvers being tested used an integration tolerance of 10^{-6} and a convergence tolerance of 10^{-4}.

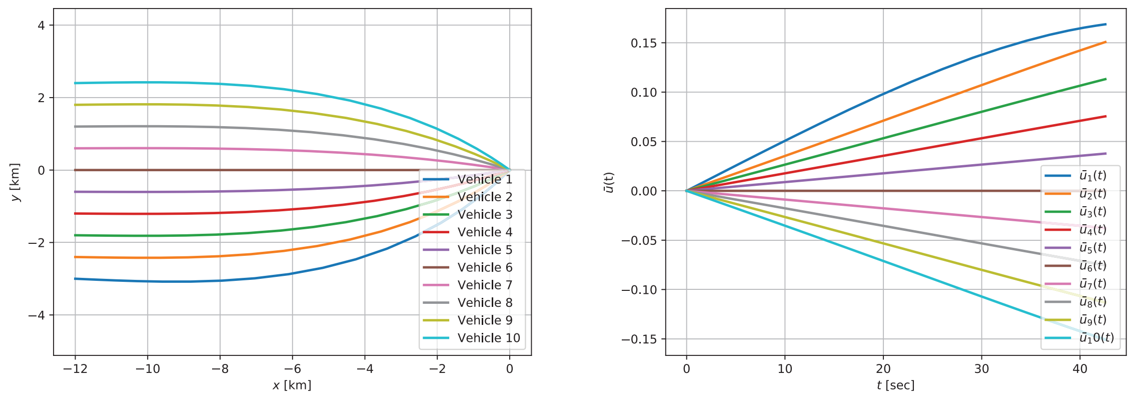
The general multi-vehicle problem is solved for different values of n and the run-time is recorded. It is to be noted that the run-time values described in the benchmarks do not include the time taken by *Numba* for compiling the functions to efficient parallel binary code. In order to ensure that the compilation time is accounted for, the solver was run twice for each test case. Since the compilation always happens the first time that the dynamic equations are evaluated, the second run gives pure run-time statistics.

Table 3.1. Benchmarking Hardware & Software Specifications

Processor	AMD FX-8320 Eight-core @ 3.5 GHz
Memory	8 GB DDR3 - 2400 MHz
OS	Ubuntu Linux 17.04
Python	v3.6.1

3.5.3 Solution

The solution for the problem with $n = 10$ is shown in Fig. 3.9 with the trajectories in Fig. 3.9(a) and the corresponding control profiles in Fig. 3.9(b). Since there are no path constraints, the optimal trajectory mainly involves a smooth turn with gradually increasing turn-rate that allows the vehicles to reach their target with the desired impact headings.



(a) Planar View (b) Control History

Figure 3.9. QCPI Benchmark Problem – Optimal Solution

3.5.4 Performance Benchmarks

The number of vehicles in the problem, n, is varied and the time taken for solving the problem is measured. Figure 3.10 compares the run-time performance of QCPI against the single and parallel shooting solvers for solving the candidate problem in Section 3.5.1. The runtime performance is also tabulated in Table 3.2.

The number of arcs used by the parallel shooting solvers is also varied between 4 and 8 arcs. It is important to note that the convergence characteristics of the parallel shooting solver changes with the number of arcs. While a larger number of arcs may require fewer iterations to converge, the linear system to be solved in each iteration grows progressively larger with the number of arcs.

An interesting result here is that the *Numba* version of parallel shooting is significantly faster than the explicit parallel shooting implementation. The explicit parallel version starts to achieve similar performance to the *Numba* version once $n = 25$. This could be due to significant overheads when the parallelization is implemented at a high-level in the *Python* language. The *Numba* version analyzes code and parallelizes the code automatically where possible. For example, *Numba* unrolls iterations of loops that can be run independently and executes them in parallel. This happens transparently wherever such loops are present and therefore can significantly boost performance. Also, all the parallel code exists at the binary level as opposed to in interpreted *Python* code, which leads to very low overhead.

The relative magnitudes of the run-times of the different implementations remain the same going from two vehicles to 10. However, once the number of vehicles increases from 10 to 25, these values change radically. One factor that could be causing this drastic increase is the size of the state-transition matrix (STM) in the shooting methods. With double-precision floating point numbers, the memory used by the STM goes from around 64 kilobytes for 10 vehicles to 403 kilobytes for 25 vehicles. A matrix of this size gets initialized and multiplied during every time-step of integration when evaluating the dynamic equations corresponding to the STM. This is in addition the state vector and other intermediate variables which will themselves be of similar sizes. The size of the L1 cache and L2 cache in the processor used for the benchmarks is 384 KB and 8 MB, respectively. With the larger sized matrices, there is a higher probability of the matrices not fitting in the cache, therefore requiring more expensive RAM access. In the case of QCPI, while a similar number of ODEs are integrated, such large matrices are only operated on as part of the MCPI integration process rather than when evaluating the dynamic equations.

Profiling the code also revealed that the number of times the dynamic equations are evaluated is approximately $8\times$ more for the single shooting method as compared to QCPI. This further contributes to higher run-time as the number of vehicles is increased. This particularly affects single shooting more due to the more parallel nature of multiple-shooting as well as the fixed overhead associated with parallel processing that may not change much with size of the problem.

It can be seen that the QCPI implementation obtains a consistent 3.5× to 4.5× speed-up over the fastest parallel shooting implementation for values of n ranging from 1 to 25. It is to be noted that these speed-ups were obtained using only automated parallelization by *Numba* and no explicitly parallel optimized code was developed for QCPI. This highlights to a certain extent the potential of QCPI for huge performance benefits with custom parallel implementations, particularly those targeting architectures such as GPUs. However, given the trends in the benchmarks discussed above, further analysis is needed to exactly quantify the degree of parallelism in the algorithm. It would also be beneficial to establish a lower bound on problem size that is required to efficiently exploit parallel architectures using QCPI.

Table 3.2. QCPI Benchmarks – Runtime vs. Number of Vehicles

Solver	$n = 1$	$n = 2$	$n = 5$	$n = 10$	$n = 25$
QCPI	0.02	0.05	0.15	0.43	2.86
Single Shooting	0.19	0.37	1.29	3.65	26.91
Parallel Shooting Explicit (8 arcs)	2.92	3.07	5.05	6.88	15.73
Parallel Shooting Explicit (4 arcs)	3.89	4.29	6.29	7.71	15.11
Parallel Shooting Numba (8 arcs)	0.08	0.16	0.659	2.233	14.12
Parallel Shooting Numba (4 arcs)	0.07	0.16	0.595	1.952	11.14

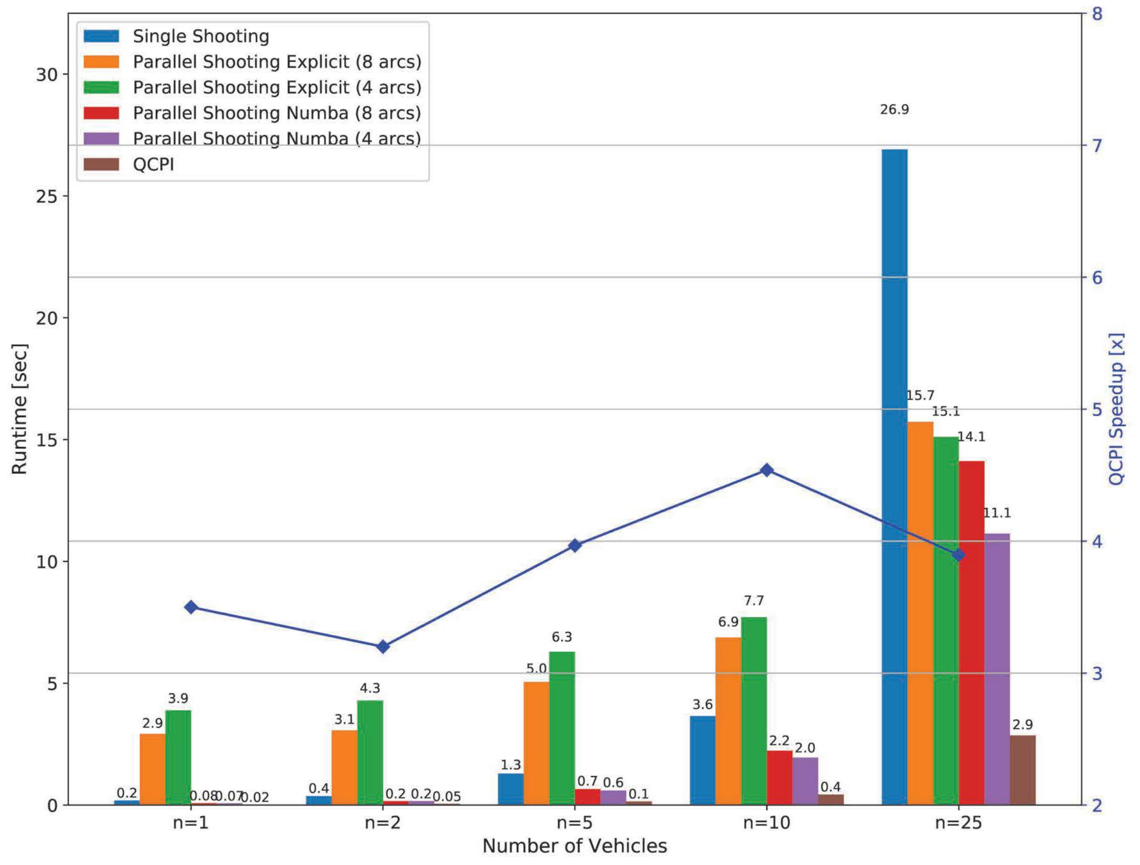


Figure 3.10. QCPI Benchmark - Comparison with Shooting Solvers

3.6 Limitations

3.6.1 Numerical Instability due to Fixed Mesh Size

The Quasilinear Chebyshev Picard Iteration algorithm has much in common with collocation methods. Like some collocation-based methods, QCPI represents the solution using an orthogonal polynomial series and uses quadrature rules to integrate the dynamic equations in the problem. This causes QCPI to have some of the same drawbacks as collocation based methods. The solution is represented on an uneven mesh of Chebyshev-Gauss-Lobatto (CGL) nodes as shown in Fig. 3.11. In this mesh, the nodes are clustered at the beginning and end of the trajectory with fewer nodes in the middle. This causes numerical instabilities when solving problems with dynamically sensitive regions in the middle of the trajectory. In shooting methods, the use of adaptive numerical integrators help avoid this issue. In collocation-based solvers such as GPOPS, adaptive mesh refinement methods [153, 154] are used to dynamically change the node positions, usually by concatenating meshes of different sizes. This allows the solver to add extra nodes in regions where the trajectory is highly sensitive, thereby improving the numerical stability of the solver.

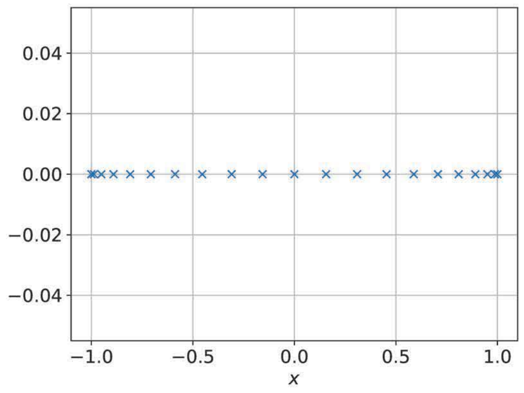


Figure 3.11. CGL Nodes for Chebyshev Polynomial Series of Order, $N = 20$

QCPI in its current implementation, uses a fixed size grid that is specified a-priori. While it is still applicable to many nonlinear optimal control problems as illustrated in this chapter, this limits its use for problems with numerically sensitive regions in the middle of the trajectory. As such, the multi-vehicle application problem detailed in the next chapter is chosen such that the numerically sensitive regions appear in the beginning and/or end of the trajectory. The challenges posed by this limitation and some strategies for mitigating them are explored further in Section 4.3.4 and Section 6.2.4, respectively.

3.6.2 Compilation Delays from *Numba*

One of the drawbacks of using *Numba* is the relatively long compilation stage the first time that the accelerated code is executed. Though *Numba* was used for accelerating both QCPI and the shooting solver (where possible) for this benchmark, the compilation time required by the two methods are starkly different as shown in Fig. 3.12. In the shooting solver, the only part that is parallelized using *Numba* is the computation of the Jacobian matrix used for generating the State Transition Matrix (STM). The fraction of code that can be parallelized using *Numba* in the case of the shooting solver is much smaller than in the case of QCPI. Consequently, this results in the compilation of QCPI code taking much longer to compile than the shooting solver.

Another reason for this compilation overhead is that the every state of every vehicle in the problem are treated as having unique equations of motion even if that is not the case in a particular problem. This is especially true in the case of multi-vehicle systems. *Numba* performs in-depth automated analysis of the code to perform its parallelization, and the increase in number of equations to be analyzed further increases the compilation time. Explicit parallelization of the code, such as what Ref. 30 does for shooting methods, would significantly improve the run-time performance as well as the compile-time performance of the numerical method.

Special handling of repeated equations in the problem can help significantly speed up the algorithm for multi-vehicle systems.

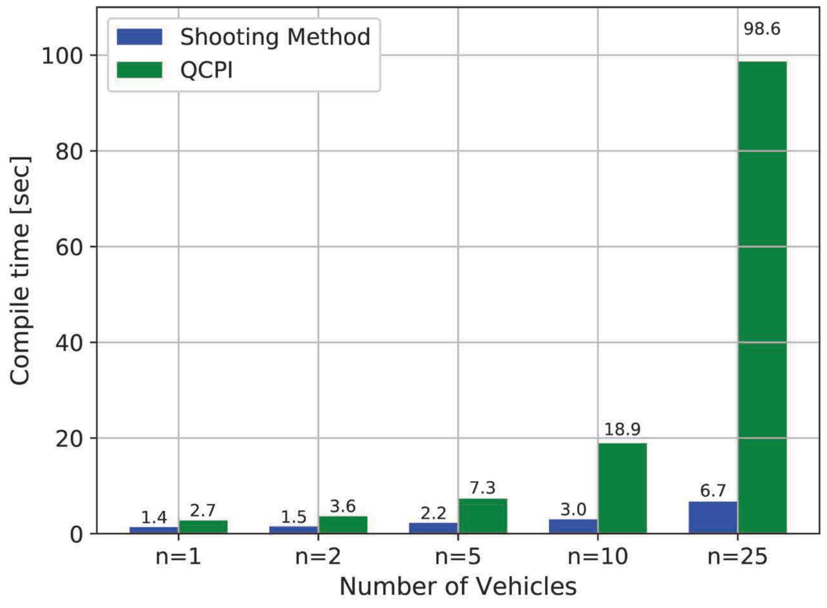


Figure 3.12. Benchmark - Compilation Time for QCPI and Shooting Methods

3.7 Summary

The Quasilinear Chebyshev Picard Iteration (QCPI) method builds on prior work utilizing a Chebyshev Polynomial series and the Picard Iteration combined with the Modified Quasi-linearization Algorithm. The capabilities of the numerical method are validated by solving some representative nonlinear optimal control problems. The performance of the solver is benchmarked against existing numerical solvers using a large multi-vehicle optimal control problem. QCPI is shown to obtain speedups in the range of 3.5x-4.5x when compared to a parallel shooting solver for solving the same boundary value problems when running on an 8-core processor. The results demonstrate that QCPI has a lot of potential for leveraging parallel computing architectures and can greatly benefit from implementation on highly parallel architectures such as GPUs.

Even with the limitations of its current implementation, QCPI has been demonstrated to be a viable, fast numerical method for solving large nonlinear boundary value problems. It advances the state-of-the-art in using indirect methods for solving large scale trajectory optimization problems. This is further illustrated in the next chapter where QCPI is combined with ICRM to solve a large multi-vehicle constrained trajectory optimization problem.

4. MULTI-VEHICLE CONSTRAINED TRAJECTORY OPTIMIZATION

4.1 Problem Statement

A constrained multi-vehicle trajectory optimization problem is posed in this Chapter to demonstrate the combined application of the ICRM and QCPI algorithms described in this dissertation. This problem is an extension of the application problem in Section 2.9 and the benchmark problem in Section 3.5.1. It consists of five vehicles each with a control constraint along with a path-constraint that enforces a keep-out zone. The scenario models a co-operative, simultaneous engagement of two targets by five vehicles and is shown in Fig 4.1. The objective is to minimize total control effort. The full optimal control problem is stated in Eq. (4.1). The various problem parameters and boundary conditions are listed out in Table 4.1 and Table 4.2 respectively. In Section 4.3, these conditions are modified in order to study the evolution of optimal trajectories for varying terminal geometry conditions. All the coordinates are specified in terms of a flat Cartesian coordinate system centered around the first target with the Y-axis pointing North and the X-axis pointing East. It is to be noted that unlike the benchmark problem used in the previous chapter, the current problem uses a three-dimensional model.

Table 4.1. Multi-Vehicle Trajectory Optimization – Problem Parameters

	x_c	y_c	r_c
Zone 1	-9.0 km	0.0 km	1.5 km

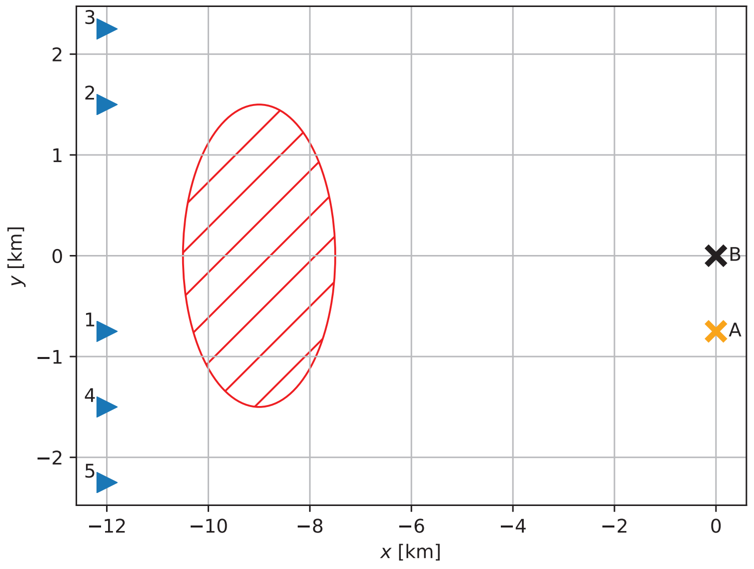


Figure 4.1. Multi-Vehicle Trajectory Optimization - Scenario Overview

Table 4.2. Multi-Vehicle Trajectory Optimization – Boundary Conditions

Vehicle	$X_i(0)$	$Y_i(0)$	$Z_i(0)$	$X_i(T)$	$Y_i(T)$	$Z_i(T)$	$\psi_i(T)$
Vehicle-1	-12.0 km	-0.750 km	1.5 km	0.0 km	0.0 km	0.0 km	+15 deg
Vehicle-2	-12.0 km	+1.500 km	1.5 km	0.0 km	0.0 km	0.0 km	-15 deg
Vehicle-3	-12.0 km	+2.250 km	1.5 km	0.0 km	0.0 km	0.0 km	-30 deg
Vehicle-4	-12.0 km	-1.500 km	1.5 km	0.0 km	0.75 km	0.0 km	+30 deg
Vehicle-5	-12.0 km	-2.250 km	1.5 km	0.0 km	0.75 km	0.0 km	+45 deg

$$\text{Min } J = \mathop{}_{0}^{T} \mathop{}_{i=1}^{n} \bar{u}_i^2 + \gamma_i^2 \tag{4.1a}$$

Subject to: (4.1b)

$$\dot{\bar{x}}_i = \bar{v}_i \cos\psi_i \cos\gamma \tag{4.1c}$$

$$\dot{\bar{y}}_i = \bar{v}_i \sin\psi_i \cos\gamma \tag{4.1d}$$

$$\dot{\bar{z}}_i = -\bar{v}_i \sin\gamma \tag{4.1e}$$

$$\dot{\bar{v}}_i = 0 \tag{4.1f}$$

$$\dot{\psi}_i = \bar{u}_m ax \bar{u}_i \tag{4.1g}$$

$$\bar{x}_i(0) = \bar{X}_{i0}, \quad \bar{y}_i(0) = \bar{Y}_{i0}, \quad \bar{z}_i(0) = \bar{Z}_{i0} \tag{4.1h}$$

$$\bar{v}_0(0) = 1 \tag{4.1i}$$

$$\psi_i(T) = \psi_{if} \tag{4.1j}$$

$$|\bar{u}_i| \leq 1 \tag{4.1k}$$

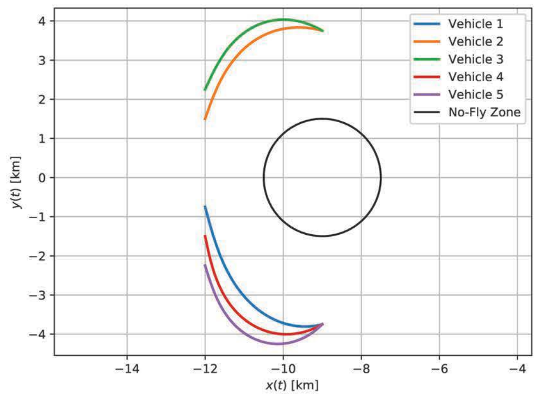
$$\overline{(\bar{x}_i - x_c)^2 + (\bar{y}_i - y_c)^2} \geq r_c \tag{4.1l}$$

$$\text{where } i = 1, 2, ..., 5$$

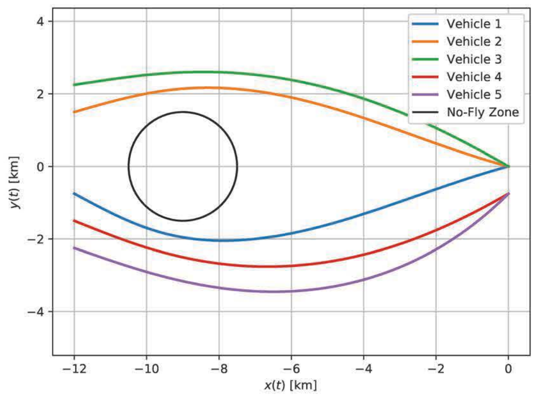
This multi-vehicle constrained trajectory optimization problem, as defined above, consists of 24 state variables, 10 control variables, 5 path constraints and 5 control constraints. On using ICRM to compute the necessary conditions of optimality, it is converted into a two-point boundary value problem with 89 ODEs. This number includes the original state variables, the extra states added by ICRM to incorporate the path constraints, the corresponding costates, the original control variables, the ICRM regularization controls, and the free final time. The Quasilinear Chebyshev Picard Iteration algorithm developed in this dissertation is used to numerically solve this nonlinear two-point boundary value problem and obtain the optimal solution.

4.2 Nominal Solution

As in the examples in the prior chapters, the variables were scaled based on a reference velocity of $V_{ref} = 300\ m/s$ and reference flight-time of $T_{ref} = 50$ sec. A continuation strategy consisting of two steps was used to evolve the trajectory starting from a trivial initial guess to the solution for the actual problem. The first stage of this progression with the intermediate targets is shown in Fig 4.2(a), and the solution obtained at the second step, where the trajectories are extended to the desired targets, is shown in Fig 4.2(b). Using the intermediate target allowed the continuation methodology (Section 5.6) to skip over any possible continuation steps that could have landed inside the no-fly zone.



(a) Intermediate Solution



(b) Nominal Solution

Figure 4.2. Multi Vehicle Problem – Construction of Nominal Solution

The optimal trajectory for the nominal problem setup as defined in the last section is shown in Fig 4.2(b). ICRM is used to regularize the path constraints and incorporate them into the problem. The regularization parameter ϵ is set to 10^{-4}. This relatively high value of ϵ results in a push off factor for the path constraint as seen by the trajectory of Vehicle-1 which is the closest to the no-fly zone constraint. Making ϵ smaller caused numerical instabilities when using QCPI to solve the boundary value problem. This is one of the limitations of QCPI as discussed before in

Section 3.6. This effectively increases the radius of the no-fly zone that is enforced on the trajectory and decreases the control authority available, as discussed before in Section 2.7. A specific case where for the no-fly zone is decreased to 10^{-6} is examined in Section 4.3.4.

4.3 Analysis

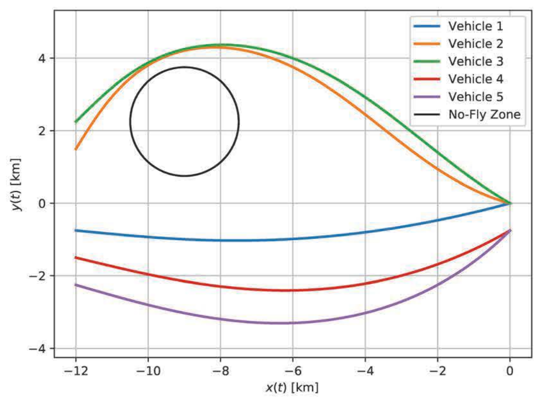
Starting with the nominal problem as shown in the previous section, boundary conditions and problem parameters are changed using homotopy continuation in order to obtain families of optimal trajectories for these conditions and illustrating the cross-coupling of dynamics in multi-vehicle systems.

4.3.1 Changing the Location of the Keep-Out Zone

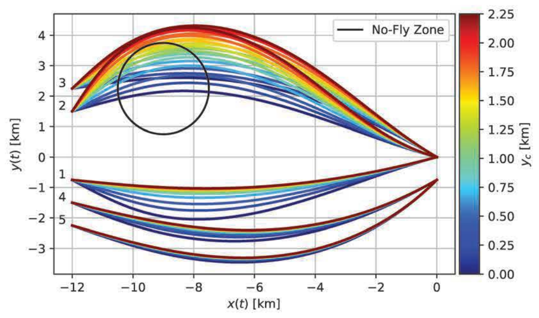
The location of the no-fly zone constraint is changed in this section using a continuation method, and the evolution of the optimal trajectories are examined in this section. The constraint is implemented as a circle positioned at (x_c, y_c) with a radius, r_c.

The position of the zone is changed by increasing y_c, pushing the constraint further North 2.25 km. The resulting change in the trajectories of all five vehicles are shown in Fig 4.3. As the constraint moves up, it becomes inactive in Vehicle-1's trajectory. This has the compound effect of also moving the trajectories of vehicles 2 and 3 further North as seen in Fig. 4.3(b). Since the speed of all vehicles except Vehicle-1 are free states, these values also change as the constraint limit is varied. Fig 4.4(a) shows the variation in vehicle speed as y_c is changed from 0 km to 2.25 km.

Vehicles 4 and 5, though initially the faster due to the placement of the path constraint, only gets slightly faster and the effect on their velocity diminishes as the path constraint gets further away from their trajectories. On the other hand, Vehicle-2 transitions from being the slowest of all when $y_c = 0$ to being the fastest when $y_c = 2.25km$. This is because the constraint and the arrival heading make the distance

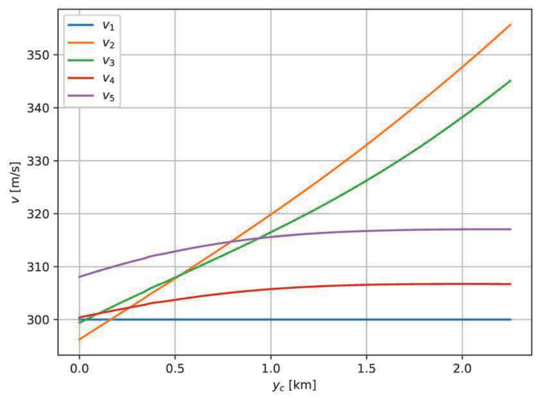


(a) Solution

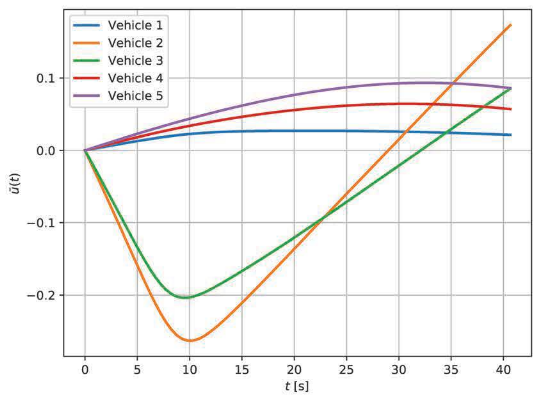


(b) Evolution of Solution with change in y_c

Figure 4.3. Multi Vehicle Problem – Moving Path Constraint – Solution



(a) v_i vs. y_c



(b) Control History for $y_c = 2.25$ km

Figure 4.4. Multi Vehicle Problem – Moving Path Constraint – Velocity and Control History

it has to travel longer than that of the other vehicles. Vehicle-3's optimal speed is significantly higher due to being pushed North in order to avoid the constraint. This is also reflected in the final control profile in Fig. 4.4(b), where $\bar{u}_2$ and $\bar{u}_3$ shows higher magnitudes of control effort compared to that of the other vehicles. The total distance flown by Vehicle-2 is the highest because there it's trajectory also has a 3D component to it and flies slightly below Vehicle-3 to arrive at the target with the right impact heading as shown in Fig. 4.5.

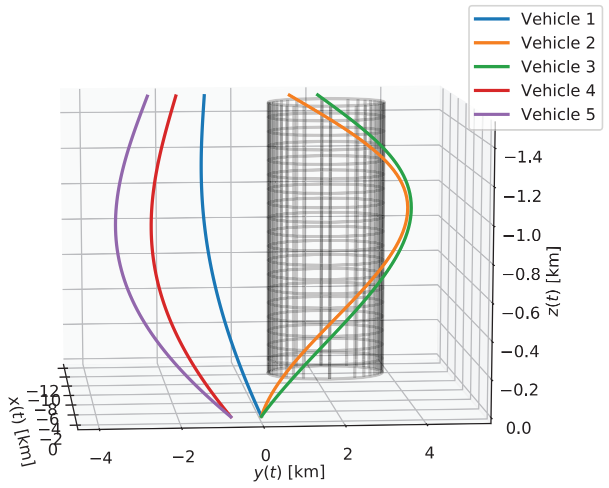


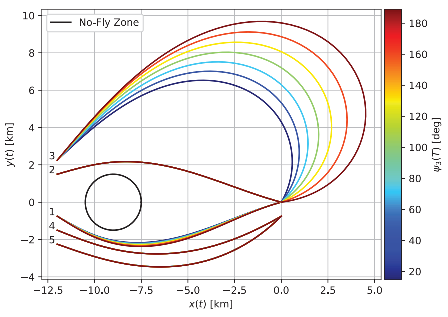
Figure 4.5. Multi Vehicle Problem – Moving Path Constraint – 3D Trajectory Profiles

4.3.2 Changing the Impact Heading of Vehicle-3

In this section, the constrained impact heading of Vehicle-3 is changed, and the effect on the overall solution structure is examined. First, the terminal heading constraint on Vehicle-3, $\psi_3(T)$, is changed from its nominal value of -30 deg to -179 deg. This limit was chosen as this approach heading would be almost directly opposite to the starting position of Vehicle-3. Changing the heading any further would result in the optimal solution changing to loop around the south of the trajectory, which causes the continuation strategy to fail. The evolution of both the trajectories are shown both 2D and 3D in Fig. 4.6. There is a significant 3D component to the change in the trajectory of Vehicle-3 as seen in Fig. 4.6(b). As heading angle turns further East, the vehicle stays higher for longer before turning around.

This scenario is one case where additional constraints are needed to ensure that the trajectory is feasible. Unlike in Section 2.9, because there is no second constraint restricting the motion of Vehicle-3, the maximum allowed velocity is the limiting factor that determines how much the impact heading can be changed. The longer flight path of Vehicle-3 results in an increase in speed as shown in the velocity profile in Fig. 4.7(a). The altitude profile, the vehicle speed, and turns are all timed so that all the vehicles reach their respective targets at the same time while satisfying all the other geometry and path constraints.

In fact, due to the significantly longer trajectory, the speed required is around 700 m/s. This shows that in this particular scenario, if all the other constraints remain the same, one of the vehicles need to be significantly different and capable of flying near two and half times as fast as the others in order to satisfy the impact heading constraint. Since the velocity is implemented as a constant value ($\dot{v}_3 = 0$) in this particular model, enforcing it as a path constraint is not possible. This is one of the limitations of the current approach. Further analysis in the next section proceeds assuming that Vehicle-3 *is* capable of achieving the required speed. The final trajectories of all 5 vehicles are shown in Fig. 4.7(b). This is the starting point for the analysis in the next section.


(a) Planar trajectories

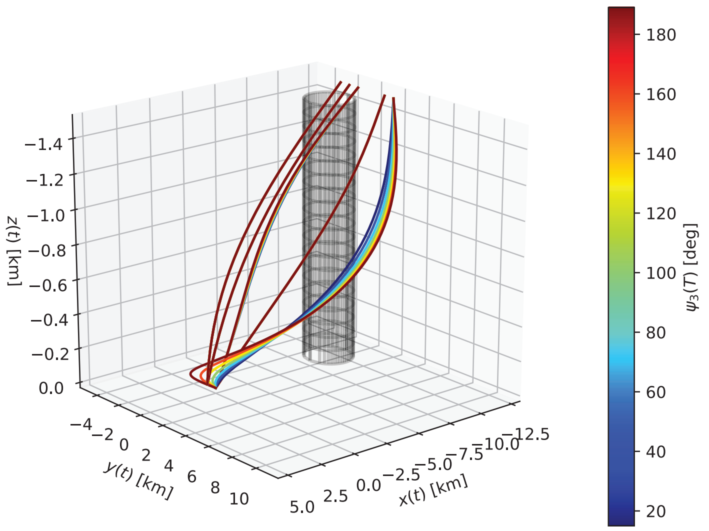
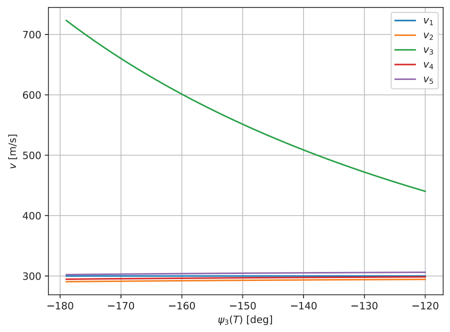

(b) 3D view

Figure 4.6. Multi Vehicle Problem – Vehicle-3 Impact Heading – Evolution of Trajectories


(a) v_i vs $\psi_3(T)$

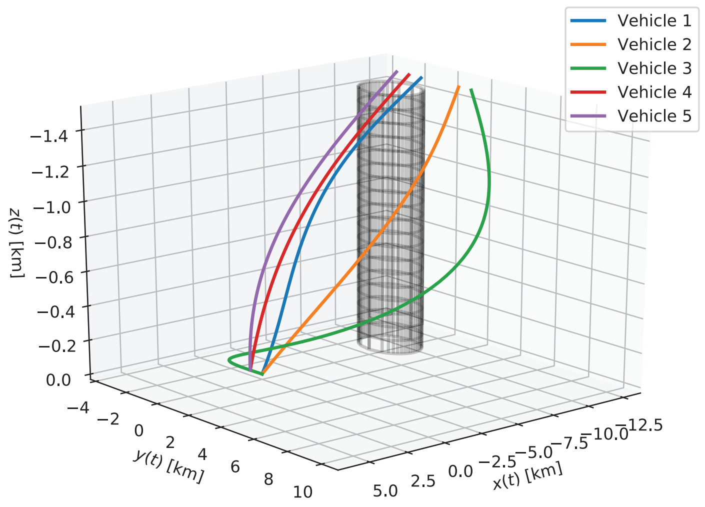
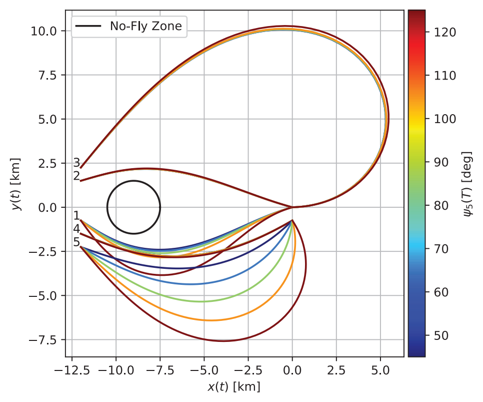

(b) Final 3D Trajectory

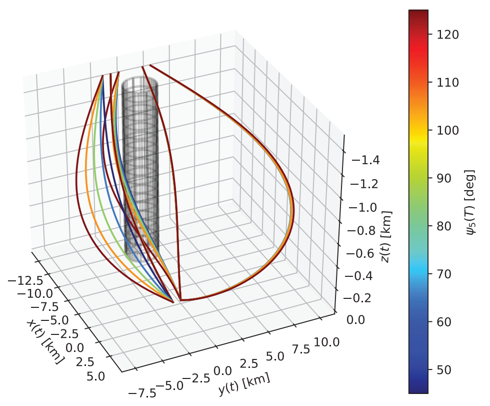
Figure 4.7. Multi Vehicle Problem – Vehicle-3 Impact Heading – Velocity Profile and Final Trajectory

4.3.3 Changing the Impact Heading of Vehicle-5

Starting with the final trajectory in the previous section (Fig. 4.7(b)), the constrained impact heading of Vehicle-5 is changed from +45 deg to +125 deg. The evolution of the vehicle trajectories are shown in Fig.4.8. The evolution of Vehicle-5's trajectory shows a similar profile to that of Vehicle-3 in the previous section. A new change in this case is that the trajectory of vehicle 1 (the "reference" vehicle) is also affected in this case. In fact as seen in the final trajectories in Fig. 4.9, this change also results in a collision between Vehicle-1 and Vehicle-4. The collision could be avoided by adding a separation distance constraint between every vehicle pair in the problem. Limitations in the current implementation makes the addition of these constraints prohibitively time-consuming. Some possible improvements to the implementation of QCPI and ICRM that can overcome these challenges are explored in the Section 6.2.



(a) Planar View

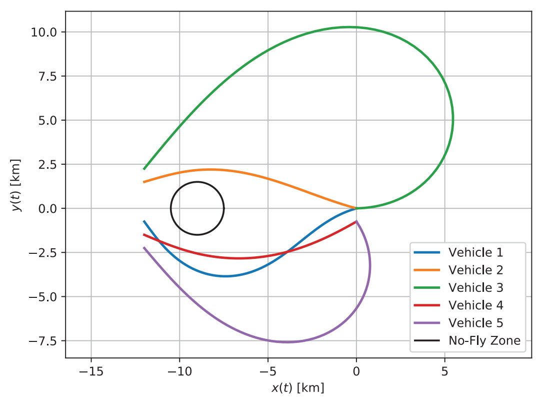


(b) 3D view

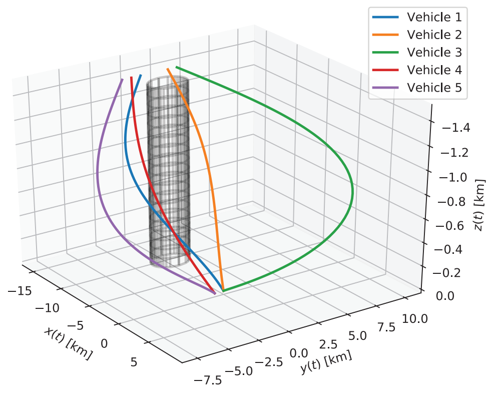
Figure 4.8. Multi Vehicle Problem – Vehicle-5 Impact Heading – Evolution of Trajectories

4.3.4 Improving Accuracy by Reducing ϵ_i

As mentioned earlier in this chapter, all of the results shown so far for this five-vehicle problem, were solved with all the regularization parameters, ϵ_i, set to 10^{-4},



(a) Planar View



(b) 3D view

Figure 4.9. Multi Vehicle Problem – Vehicle-5 Impact Heading – Final Trajectory for $\psi_5(T) = 125$ deg

for the no-fly zone constraint. It is to be noted that each vehicle has a separate parameter, ϵ_i for its no-fly zone constraint as well as a different parameter for the control limit constraint. Therefore, in the context of this section, changing ϵ_i refers to ϵ for the no-fly zone constraint for *all five* vehicles in the problem. While lower values of ϵ_i would reduce the push-off factor around the constraint, it would also make the BVP significantly more difficult to solve. This is a side-effect of the numerical sensitivity issue outlined before in Section 2.7 as well as QCPI's fixed mesh spacing (Section 3.6.1) in its current implementation.

One way to allow for smaller values of ϵ_i is to increase the number of nodes used by QCPI to discretize the problem space, at the cost of increased computation time. In this section, the number of nodes, N, used by QCPI is increased to 151 to facilitate a continuation process on ϵ_i, reducing it down to 10^{-6}. This is done to demonstrate that QCPI is indeed capable of closely tracking path constraints albeit at a significant computational cost. The trajectory obtained for $\epsilon_i = 10^{-6}$ starting with the nominal solution from Section 4.2 is shown in Fig. 4.10. It can be seen that the push-off factor around the no-fly zone that appeared in the previous trajectories in this Chapter has practically vanished with trajectories closely following the boundary of the no-fly zone. It is to be noted that when N was increased to 151, the computation time increased significantly, with the continuation process from $\epsilon_i = 10^{-4}$ to $\epsilon_i = 10^{-6}$ taking almost 4 hours on a MacBook Pro with a quad-core processor.

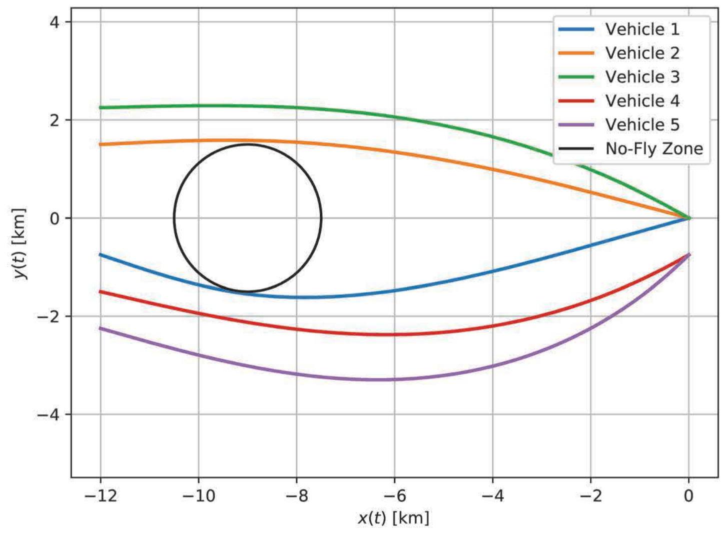


Figure 4.10. Multi Vehicle Problem – Trajectory for $\epsilon = 10^{-6}$

A rough analysis of the relation between the value of ϵ_i and the number of QCPI nodes required is also performed and the results are shown in Fig. 4.11. The continuation process for decreasing ϵ_i was repeated for different values of N until the process fully converged to a solution. If the number of nodes is too low to capture the dynamics at the middle of the trajectory, the residual error in boundary conditions plateaus at a valuer higher than convergence tolerance. The emerging trend in Fig. 4.11 shows that QCPI's numerical sensitivity issue is not intractable. Such highly sensitive problems can indeed be solved using QCPI by adding extra nodes at the affected areas of the trajectory. However, the method for adding such nodes is highly inefficient in the current implementation of QCPI, owing to the fixed mesh spacing. Some strategies for making QCPI more efficient in this aspect are outlined later in Section 6.2.4.

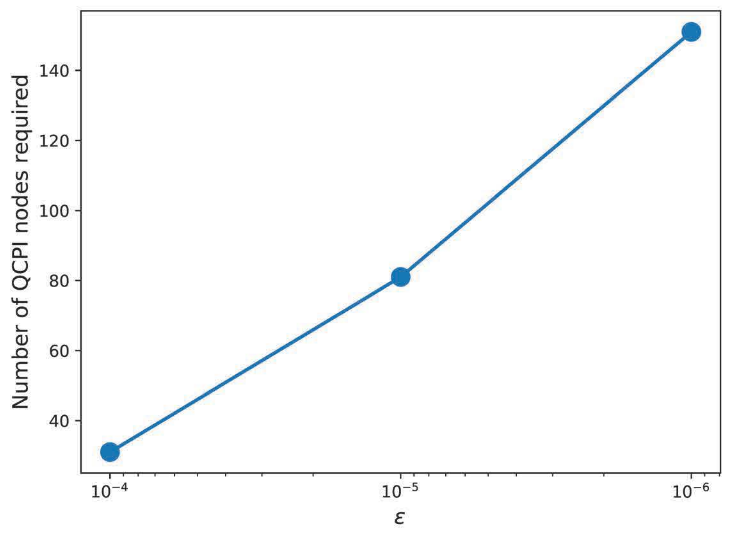


Figure 4.11. Multi Vehicle Problem – Number of QCPI Nodes Required for Different Values of ϵ_i

4.4 Summary

A multi-vehicle trajectory problem with path constraints was solved using a combination of the ICRM and QCPI methods developed in this dissertation. The scenario was set up to serve as a demonstration of the capabilities of these methods for solving large-scale trajectory optimization methods. Analysis in this chapter, along with that in Section 2.9 demonstrated some of the cross-coupling effects that appear when the optimal trajectories of multiple vehicles are simultaneously solved for while accounting for various geometry and path constraints. This is a type of problem that would be considered infeasible to solve using traditional implementations of indirect methods due to the challenges conventionally associated with these methods [8].

The scenario described in this chapter as well as the ones used for validation of the methods presented in this dissertation were solved using an open source, indirect trajectory optimization framework – *beluga*. ICRM and QCPI form integral parts of this framework that automates the construction of the necessary conditions of optimality and includes automated continuation strategies for numerically solving boundary value problems. The design and architecture of this framework is described in the next chapter.

5. *BELUGA* — AN INDRECT TRAJECTORY OPTIMIZATION FRAMEWORK

5.1 Introduction

The methods developed in this dissertation as described in the previous chapters form the key components of a generalized, open source trajectory optimization framework. In its current nascent form, the framework only supports indirect methods. However, the longer term goal of this framework is to become a viable alternative to state-of-the-art direct solvers such as GPOPS [6] and DIDO [7], by supporting a wide variety of direct and indirect methods for solving real-world optimal control problems. This chapter describes the design and overall structure of this framework and demonstrates some of its use cases. This trajectory optimization framework with full source code can be obtained at `https://github.com/Rapid-Design-Of-Systems-Laboratory/beluga/tree/tantony-phdthesis`.

A prototype of this rapid trajectory optimization framework was originally developed using MATLAB [96] as described in Ref. 155. It was used as a foundation for implementing a "mathematically unified design environment that is capable of performing rapid simultaneous hypersonic aerodynamic and trajectory optimization". The prototype framework was very specific to the application that it was created for and had limited options for the numerical solvers and symbolic engines it could use. Mathematica [73] and MATLAB Symbolic Toolbox [74] were used for symbolic computation, and the *bvp4c* [88] numerical solver was used for solving boundary value problems. In Ref. 79 and 30, this framework was further extended to be a more generalized framework for solving optimal control problems capable of leveraging GPUs for accelerated computation. The capabilities of this framework was demonstrated in a range of aerospace applications [14, 15, 110, 156–161].

Owing to licensing limitations in MATLAB as well as performance issues, the framework, now code-named *beluga*[1], was redesigned from scratch using the Python [105] programming language. Python's powerful object oriented and functional programming capabilities and close integration with C/C++ and FORTRAN [162] has made it an ideal candidate for the implementation of scientific computing libraries [163]. In the recent years, the availability of fast scientific computing and visualization libraries such as NumPy [2], SciPy [1, 163], and Matplotlib [164, 165] coupled with the huge open source community around it has led to Python emerging as one of the most favored platforms for scientific computing. Python has been used in a wide range of scientific fields such as astronomy [166], astrodynamics [167], particle physics [168], quantum mechanics [169] and biotechnology [170]. Several popular linear algebra, numerical integration, and optimization codes written in FORTRAN such as LAPACK [171] and ODEPACK [172] have been linked to Python. By leveraging these tools, it is possible write programs that utilize these really fast methods while retaining the flexibility of a high-level language like Python. All these features makes Python an ideal platform for the development of an open-source computation framework like *beluga*.

All of the optimal control problems shown in this dissertation were solved using the *beluga* optimal control framework. This chapter highlights some of the features of the framework as well as some guidelines on how to use it.

5.2 Problem Definition

As mentioned before, one of the major drawbacks of using indirect methods is the derivation of the necessary conditions of optimality. *beluga* simplifies this by automating the derivation of these conditions. However, this requires that all the components of the optimal control problem be defined. This section describes some of the major components of a problem definition file used for solving an optimal

[1] *beluga* is not an acronym. The name was chosen because we think beluga whales are interesting creature and we wanted a simple, easy-to-remember name like many other open source projects

control problem in *beluga*. Many of these variables have units associated with them which are used for dynamic scaling as described in Section 5.3. In this section, the constrained brachistochrone problem from Section 2.8.1 will be used as the example and different sections of the input file will be shown. The complete input file is shown in Appendix B.2.

State and Control Variables

State variables are the mathematical variables that define the state of the dynamic system being optimized. Each state variable has a dynamic equation and a unit associated with it while the control variable has just a name and a unit. The following code-block shows state and control variable definitions for the constrained Brachistochrone problem. The independent variable in the problem (usually time), and it's unit is also defined in the input file.

```
# Define independent variables
ocp.independent('t', 's')

# Define equations of motion
ocp.state('x', 'v*cos(theta)', 'm')\
   .state('y', 'v*sin(theta)','m')\
   .state('v', 'g*sin(theta)','m/s')

# Define controls
ocp.control('theta','rad')
```

Constants

Constants are usually used to define model parameters such as gravity, atmospheric density, etc. While these are constant while the problem is being solved, it is possible to vary these values using continuation over a series of steps. One application of this would be to solve a trajectory problem without an atmosphere and then gradually increasing the constant corresponding to surface atmospheric density to examine the effect of accounting for drag. In case of the constrained brachistochrone problem, the only constant is the acceleration due to gravity and it is defined as shown below:

```
# Define constants
ocp.constant('g',-9.81,'m/s^2')
```

Cost Functionals

beluga allows for two kinds of objective functions to be defined – path costs and terminal-point costs. For path costs of the form $J = {}_{0}^{T} L \; dt$, the integrand L is defined in the input file. In case of the constrained brachistochrone problem, since the objective is to minimize the total-time, the cost functional is defined as:

```
# Define costs
ocp.path_cost('1','s')
```

Constraints

Three kinds of constraints can be defined in the input file – initial point constraints, terminal point constraints, and path inequality constraints. The initial and terminal point constraints form the boundary conditions on the state variables in the problem. Any state variable that is not included in these definitions is assumed to be unconstrained at either end of the trajectory. In case of the constrained brachistochrone problem, these are defined as:

```
ocp.constraints() \
    .initial('x-x_0','m') \
    .initial('y-y_0','m') \
    .initial('v-v_0','m/s')\
    .terminal('x-x_f','m') \
    .terminal('y-y_f','m')
```

In the case of path constraints, additional specifiers are required to define if the path constraint is an upper bound, lower bound, or a two-sided constraint. In case of the constrained brachistochrone problem, the path constraint is defined as:

```
ocp.constraints() \
    .path('constraint1','y+x','>',-1.0,'m',start_eps=1e-4)
```

The above definition enforces the path constraint $y + x > -1$. Since the problem is to be solved using ICRM, a starting value for the regularization parameter, , is also specified (in this case as 10^{-4}). An upper bound path constraint would similarly be defined as:

```
ocp.constraints() \
    .path('constraint1','x+1','<',1,'m',start_eps=1e-4)
```

A two-sided constraint such as a control bound would be defined as :

```
ocp.constraints().path('ulim','u','<>',1,'rad/s',start_eps=1e-4)
```

Sub-expressions

Sub-expressions are optional definitions that may help simplify equations in the input file. For example, in a hypersonic trajectory problem, the atmospheric density ρ is a variable that may appear multiple times in different state equations. When using an exponential atmospheric model, it may be expanded to $\rho = \rho_0 \exp(-h/H)$ where h is the altitude, H is the scale-height of the atmospheric model and ρ_0 is the surface atmospheric density. In order to avoid repetition, *beluga* offers the provision of defining "quantities" that get automatically substituted into all equations before the necessary conditions are derived. So in case of the hypersonic problem, some common quantities such as density and dynamic pressure may be defined as:

```
ocp.quantity('rho','rho0*exp(-h/H)')
ocp.quantity('q','0.5*rho*v^2')
```

These variables, 'rho' and 'q' can then be used in place of these expressions in other equations in the problem. This helps significantly improve the readability of the input file.

5.3 Dynamic Scaling

The different states and costates in the BVP resulting from using indirect methods can vary from each other by several orders of magnitude. This presents a challenge while solving the BVP using numerical methods. For example, it may be impractical to enforce a tight error tolerance (e.g., 10^{-10}), on a state that has values on the order of 10^9. In order to mitigate this issue, the states, costates, constants, parameters, constraints, and the independent variable (time) are dynamically scaled during every iteration of the continuation method. It is generally difficult to identify scaling factors for all these parameters for complex, hypersonic problems. By starting with a simple problem and evolving it into more complex problems, it is possible to evolve the scaling factors based on the solution history of the past iterations during the continuation process. This scaling methodology is fully automated and the designer only has to specify the scaling factor associated with each of the fundamental units. In *beluga*, scaling is performed based on the units associated with the different components of the problem. For example, in the constrained brachistochrone problem, the base scaling factors are defined as:

```
ocp.scale(m='y', s='y/v', kg=1, rad=1, nd=1)
```

Here, the 'meter' unit is scaled based on the absolute magnitude of the y state variable. As the magnitude of y increases during continuation, this value also increases, thereby keeping the scaled state variable value on the order of unity. Similarly the time unit is scaled by the magnitude of the expression x/v and every other unit is left unscaled. Since all the other associated variables and constants also have their own dimensional units (e.g. m/s^2 for g), these values also get scaled accordingly.

5.4 Necessary Conditions

The framework offers two options for calculating the necessary conditions of optimality – conventional optimal control theory [12] and ICRM. The former converts the constrained optimal control problem into a multi-point boundary value problem while the latter regularizes the path constraints using the methods described in this dissertation to form a two-point boundary value problem.

All of the equations relevant to the problem are defined as strings by the designer. These strings are converted into Sympy [72] symbolic expressions so that they can be manipulated. In case of ICRM, the path constraints are regularized and extra states and control variables are added to the problem definition. The dynamic equations and cost functionals are used to formulate the Hamiltonian and then the dynamics and boundary conditions for the costates. The necessary conditions of optimality thus formulated are used to generate Python functions that evaluate both the ODEs and the boundary conditions for the two-point or multi-point boundary value problem. Each numerical solver defines a template file that converts the symbolic expressions representing the necessary conditions of optimality into executable Python code.

5.5 Numerical Solvers

beluga defines a generic interface that allows the implementation of different numerical methods for solving boundary value problems. Each numerical method defines its own pre-processing step for converting symbolic expressions into executable Python code. This allows for method-specific code optimizations to be applied. For example, QCPI implements code parallelization using the *Numba* library that is not implemented by the shooting solver. At the time of writing, *beluga* offers two numerical methods – a multiple shooting solver and a QCPI implementation. The designer specifies the numerical method in the input file. The open nature of the project offers scope for further expansion of the framework using third-party contributions.

5.6 Continuation Strategies

Continuation plays a major role in being able to use indirect methods to solve complex optimal control problems [13]. As such, *beluga* offers different strategies for generating initial guess as well as for performing continuation. Currently, two types of continuation strategies are implemented – manual strategy and bisection strategy. These methods assume that the initial guess converges to a valid solution to the boundary value problem. Continuation is then performed or the initial/terminal boundary conditions or the constants until the desired parameters are achieved.

5.6.1 Initial Guess Generation

Practically all numerical methods for solving nonlinear boundary value problems require an initial guess. Indirect methods in particular are known for requiring an accurate initial guess and having a small radius of convergence. *beluga* includes three types of initial guess generators.

Automatic Initial Guess using Integration

This method is the simplest to set up but may not work immediately for more complex problems. In this method, a starting/ending point is specified for the states along with fixed initial guess for the costates. This starting point is then integrated forward/backward numerically for a fixed amount of time, and the result is used as the initial guess for the numerical solver. When using ICRM, it is also necessary to provide starting values for the control variables. In case of the constrained Brachistochrone problem solved using ICRM, the initial guess is generated using this method, and it is defined in the input file as:

```
guess_maker = beluga.guess_generator('auto',
                start=[0,0,1], # Starting values for states
                direction='forward',
                costate_guess = 0.1,
                control_guess = [-3.14*60/180, 0.0, 0.0],
)
```

Data File Initial Guess

This method uses a previously converged solution as the starting point for continuation. In this case, the name of the data file and the index of the solution to be used is specified. The solution data is loaded from the file and passed directly to the numerical solver.

Custom or Static Initial Guess

This method is used when the designer wants to manually specify the complete initial guess data structure. This provides adequate flexibility in those cases where the automatic guess generator proves insufficient or when the designer wants to use insight into the problem to provide a custom initial guess solution.

5.6.2 Manual Continuation Strategy

This is the simplest homotopy continuation strategy implemented in *beluga*. The designer specifies the target values for the initial or terminal boundary condition as well as the number of intermediate steps to take. The framework then solves each step in sequence, using the previous solution as the initial guess for the next until the desired parameters are achieved. If any of the steps fail to converge, the iteration stops.

For example, for the Brachistochrone problem, if the automatic initial guess is used, a trajectory that is about 0.1 s long is obtained. Continuation is performed on the terminal x and y values until the desired values are reached. This is defined as:

```
continuation_steps = beluga.init_continuation()
```

```
continuation_steps.add_step('manual') \
              .num_cases(21) \
              .terminal('x', 10) \
              .terminal('y',-10)
```

It is possible to add more than one continuation step to, for example, perform a continuation in x first and then in y. This may be required in some cases to navigate around infeasible areas in the design space.

5.6.3 Bisection Continuation Strategy

The bisection strategy builds on the manual strategy and attempts to automatically find a feasible continuation step size. In case of the Brachistochrone example, such a continuation strategy would be defined as:

```
continuation_steps = beluga.init_continuation()

continuation_steps.add_step('bisection') \
              .num_cases(21) \
              .terminal('x', 10) \
              .terminal('y',-10)
```

Unlike the manual strategy, if any of the intermediate steps fail to converge, the step is cut in half, and the numerical solver attempts to solve it again. For example, if the continuation succeeded up to terminal boundary conditions of $x = 5$ and $y = -5$, but somehow failed to converge for $x = 5.5$ and $y = -5.5$, the solver would cut this step size in half and try to solve for $x = 5.25$ and $y = -5.25$. This bisection would continue until a pre-defined number of divisions (10 by default).

This strategy helps reduce some of the burden on the designer of selecting continuation steps and was used to solve all of the examples described in this dissertation.

5.6.4 Advanced Continuation Strategies

The challenge with the two continuation strategies discussed above is that there are many cases in which a smaller continuation step may result in the problem becoming even more infeasible. An example of this can be seen in this figure from Ref. 80. If continuation is simultaneously being performed on both x_1 and x_2, there are scenarios where decreasing the step-size may result in the problem becoming infeasible when the terminal boundary conditions land in the red area. The way to resolve this using manual/bisection strategies would be to perform continuation on x_1 and x_2 separately.

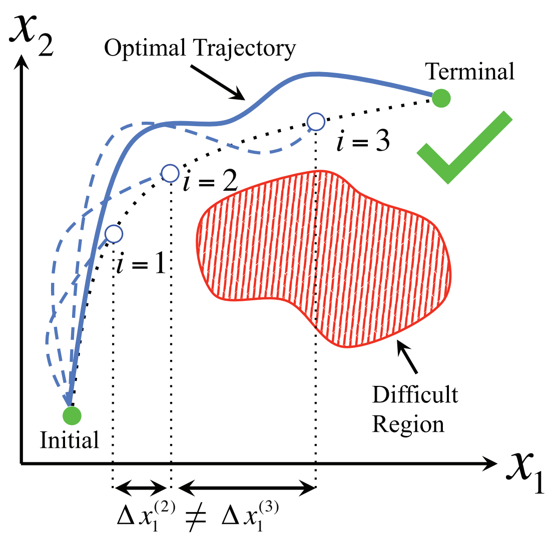


Figure 5.1. Adaptive Continuation in Design Space with Infeasible Areas [Source: Mansell [80]]

Ref. 80 describes two adaptive strategies that can automatically search in the continuation space by changing both the step-size and direction. The strategy uses Lagrange multipliers used for adjoining boundary conditions and graph-search methods such as A* and RRT for find a feasible path through complex design spaces. *beluga* offers an interface in which this and other advanced continuation strategies can be implemented.

5.7 Visualization

Visualizing the results is an important part of solving any engineering problem. *beluga* includes a flexible, extensible visualization library that supports renderer backends such as MatPlotLib [164], Bokeh [173] and ToyPlot [174]. The visualization module offers the ability to compute and plot time-series data consisting of arbitrary expressions containing constants and variables defined in the problem. These expressions can be evaluated on coarse or fine meshes using spline interpolation, transparent to the user.

The visualization module also defines a generic data source interface that can be customized to load data from a variety of sources. The default implementation includes support for *beluga* data files as well *.MAT* files generated by GPOPS [6]. This allows comparison of results obtained from different sources with the solutions obtained by *beluga*. This is very important for validation purposes. A sample plotting script used for comparing results from *beluga* and GPOPS can be seen in Appendix B.3.

5.8 Summary

beluga is being developed with the goal of being a viable contender to state-of-the-art direct solvers such as GPOPS and DIDO. In its current form, as illustrated by the various examples in this dissertation, it has made significant advances in automating the use of indirect methods for trajectory optimization. The implementation of ICRM has enabled easy inclusion of path constraints in optimal control problems, almost with the level of ease as GPOPS. Though there are numerical difficulties in some cases, the bulk of the complex analytical math is automatically performed behind-the-scenes without requiring any intervention from the designer. *beluga* offers a powerful interface for implementing custom algorithms and numerical methods. In fact, it could even be expanded to support direct solvers which could facilitate the application of a mix of direct and indirect solvers on different parts of the same problem.

Following on the path of popular and widely used scientific software projects such as SciPy [1] and Numpy [2], *beluga* is released under the permissive MIT license [3]. Being an open source project allows the community to contribute freely to the framework, further expanding its capabilities and allow faster integration of new advances to the state-of-the-art [175–178].

6. SUMMARY AND FUTURE WORK

6.1 Summary of Contributions

The contributions described in the previous chapters advance the state-of-the-art in solving large scale trajectory optimization problems using indirect methods. The Integrated Control Regularization Method (ICRM), described in Chapter 2, overcomes one of the major limitations traditionally associated with indirect methods. It does so in a way that makes it possible to still use existing numerical methods for solving boundary value problems. In Chapter 3, ICRM is complemented by a new numerical method for solving large scale nonlinear boundary value problems that is capable of utilizing parallel computing architectures. This is particularly important as modern computing hardware is trending towards highly parallel architectures. In Chapter 4, a large scale multi-vehicle problem is solved using the two methods developed in the previous chapters. It demonstrates that even with relatively simple dynamic models, it is possible to study complex behavior that emerges in optimal trajectories of multi-vehicle systems. Chapter 5 summarizes the features and design of an open source indirect trajectory optimization framework which was used to solve all of the examples in this dissertation. QCPI and ICRM are implemented as part of this framework. This chapter summarizes all these contributions and then describes future work that can overcome limitations of the current implementations.

6.1.1 Integrated Control Regularization Method (ICRM)

Incorporation of path constraints into optimal control problems using indirect methods is generally considered a non-trivial task. It is also cited one of the main reasons for not using indirect methods for solving real-world trajectory optimization problems. This is because when using indirect methods, path constraints force the solution to be split into multiple arcs with a sequence that has to be known a-priori. The arcs also introduce interior boundary conditions forming a multi-point boundary value problem (MPBVP). Providing an accurate initial guess to these MPBVPs is also a non-trivial task. Prior to this dissertation, one way for overcoming this issue was to use a continuation strategy for introducing path constraint arcs one at a time into an unconstrained solution. However, this strategy does not scale well as the number of constraints increase or when the same constraint is active and inactive multiple times in the solution. Another strategy is to regularize the path constraints using saturation functions and formulating the BVP as one consisting of a differential algebraic equation (DAE) system. This type of BVP required the development of a special numerical solver that also has difficulties scaling as the problem size increases.

The Integrated Control Regularization Method (ICRM), as described in this dissertation, uses saturation functions to incorporate path constraints into an optimal control problem, while at the same time enabling the use of existing numerical solvers such as the shooting method. It does not introduce interior boundary conditions, and the solution remains a single arc. The ability to use existing numerical methods also means that it can leverage parallel numerical methods developed for solving generic boundary value problems. Comparisons with results obtained using conventional optimal control theory as well as direct solvers such as GPOPS, where applicable, validates the accuracy of the constrained solutions generated using ICRM. The method is then applied to a two-vehicle cooperative engagement scenario with two path constraints to illustrate that it is capable of solving complex optimal control problems of the type that was previously considered impractical to solve using indirect methods.

6.1.2 Quasilinear Chebyshev-Picard Iteration (QCPI)

The main performance bottleneck when solving large scale optimal control problems using indirect methods is the numerical method used for solving the boundary value problems that result from applying indirect methods. One way to accelerate these numerical methods is by leveraging parallel computing architectures that can execute different parts of the algorithm simultaneously rather as a serial process. Prior work in Ref. 30 examined ways to structure a multiple shooting solver to leverage highly parallel GPUs for solving boundary value problems. It also highlighted some of the limitations of the method as the size of the problem increases.

In this dissertation, a new numerical method is developed with inherently parallel features for the express purpose of leveraging parallel computing architectures. The Quasilinear Chebyshev-Picard Iteration (QCPI) method builds on prior work based on the Picard Iteration and the Chebyshev-Gauss quadrature rule. While the previous method, the Modified Chebyshev-Picard Iteration (MCPI) was restricted to being able to solve specific types of nonlinear boundary value problems with fixed boundary conditions, QCPI extends it to a larger class of general nonlinear boundary value problems. MCPI was not capable of solving for free-parameters in boundary value problems or nonlinear boundary conditions. QCPI overcomes this by leveraging the Modified Quasilinearization Algorithm which can solve for free-parameters without explicitly propagating the sensitivity matrix.

One of the main features of QCPI that make it highly parallel is that it mainly consists of multiplying large matrices – an operation which has many existing efficient, highly parallel implementations. It also consists of large number of independent numerical operations evaluating functions over the entire solution space unlike multiple shooting where the solution is built up one time-step at a time. QCPI is developed in the Python programming language and then accelerated using the automatic parallelization library, *Numba*. The method is validated by solving some well-known optimal control problems and comparing the results to those obtained using a shooting solver. The performance of the solver is then benchmarked by solving a multi-vehicle cooperative engagement scenario and comparing the runtime to that of a shooting solver. It was shown that QCPI scales very well as the problem gets larger and is also able to more efficiently leverage parallel computing resources as compared to the shooting solver.

6.1.3 Large Scale Multi-Vehicle Trajectory Optimization

A multi-vehicle cooperative engagement scenario with path constraints is set up to illustrate the capabilities of the ICRM and QCPI methods developed in this dissertation. The problem consists of five vehicles, a 3DOF kinematic model, control constraints, and a keep-out zone constraint. The versatility of ICRM is demonstrated by performing trade-studies that involve changing constraint parameters and problem boundary conditions. Smooth transitions between constrained and unconstrained arcs and vice-versa are demonstrated which is something that is not possible with conventional path constraint implementations when using indirect methods. The trades are also used to demonstrate the cross-coupling effects that emerge in optimal trajectories of multi-vehicle systems even when using relatively simple dynamic models. This provides a starting point and demonstrates that indirect methods can indeed be used to solve highly constrained, large, nonlinear trajectory optimization problems.

6.1.4 Open Source Indirect Trajectory Optimization Framework

Another major challenge often cited as a drawback of indirect methods is that knowledge of optimal control theory is required in order to derive the necessary conditions of optimality. The advent of modern symbolic computation engines such as Sympy and Mathematica has allowed the automation of almost all aspects of deriving the necessary conditions of optimality. In this dissertation, the design and development of an open source, indirect trajectory optimization framework is described. The framework, called *beluga*, enables a designer to define a trajectory optimization problem and solve it using indirect methods without having to manually derive any of the necessary conditions of optimality. This framework helps bring near parity in ease-of-use between indirect methods and direct solvers such as GPOPS. Similar to

GPOPS, the designer only has to list out the various components of the problem such as states, controls, constraints and dynamic equations, and the framework automates the rest of the derivations required for applying indirect methods. It also includes a rich visualization framework that can leverage existing visualization libraries while combining data from multiple sources.

beluga also implements continuation strategies which are used to solve the nonlinear boundary value problems that result from applying indirect methods. This is required because it is often not practical to supply an accurate initial guess to the highly complex nonlinear boundary value problem. Instead, it is easier to start with a trivial initial guess for a simpler problem and then change the solution in a series of steps until the desired problem parameters are achieved. This dissertation describes two such continuation strategies, namely, manual and bisection strategies that provide some level of automation to this process. The framework was used to solve all of the optimal control problems described in this dissertation and to generate all accompanying trajectory visualizations. *beluga* was designed with the goal of becoming a viable, free, and open source alternative to state-of-the-art design software in terms of performance, accuracy, and ease-of-use, and the work presented in this dissertation lays the foundation for achieving this goal.

6.2 Future Work

6.2.1 Automated Computation of ICRM Push-Off Factor

In the multi-vehicle example in Chapter 4, the path constraint was enforced in such a way that there was a significant push-off factor with how close the trajectory approached the constraint. This was because a relatively large value was used for the regularization parameter, , while incorporating the constraint using ICRM. As discussed before in Section 2.7, smaller values of can reduce the push-off factor with the added cost of high numerical sensitivity. A multiple shooting solver with adaptive stepping is able to accommodate for this but a solver like QCPI with a fixed grid struggles to solve such BVPs. While the addition of extra nodes can help mitigate this to a certain extent, doing so can significantly increase the computation time.

An alternative approach would be to use a higher value of and modify the constraint limit such that the push-off factor ensures that the actual trajectory obeys the design constraints. However, the relationship between the push-off factor and is highly problem specific. One way to find the “right” values for and the constraint limit would be to use a bisection search strategy. A good starting value for should be first found using trial and error which keeps the problem feasible in terms of constraints while also not making the problem numerically sensitive. The constraint limit can then be adjusted using a bisection search methodology until the effective constraint limit in the problem (due to the push-off factor) matches the desired problem parameters. This would significantly improve the performance of numerical solvers when using ICRM to solve constrained optimal control problems.

6.2.2 Fully Numerical Indirect Optimal Control

The use of indirect methods for optimal control still involves a significant amount of symbolic computation when the necessary conditions of optimality are derived. Especially with the use of ICRM, all these computations involve taking one or more derivatives of expressions consisting of the dynamic equations and objective functionals of the problem. These could be formulated as a single two-point boundary value problem by using automatic differentiation algorithms [179, 180] completely avoiding the symbolic manipulation of large equations.

Another strategy for doing this would involve representing the Hamiltonian as a Chebyshev polynomial series and then using the derivative rules of Chebyshev polynomials to represent the costates and other necessary conditions of optimality. This would involve combining QCPI and ICRM into a hybrid method which directly formulates the two-point boundary value problem without symbolically deriving the necessary conditions of optimality. Developing such a unified method that leverages indirect methods would allow the easy incorporation of black-box functions into dynamic models while still maintaining the high quality of solutions guaranteed by indirect methods.

6.2.3 Improved Numerical Stability for DAEs

In ICRM, the differential algebraic equations in the boundary value problem are differentiated to obtain differential equations for the algebraic variables. While this works effectively in many cases as illustrated in the examples in this dissertation, there are many problems in which numerical instabilities arise when using this approach to solve DAEs. The original work that ICRM is based on [111] overcomes this by developing a custom numerical solver based on collocation that can incorporate the algebraic conditions directly into the numerical solution process. In a similar fashion, one strategy to solve DAEs without differentiation would be to incorporate the algebraic constraints directly into the QCPI solution process. If the algebraic conditions can be incorporated into the QCPI iteration process for calculating Chebyshev coefficients, the numerical instabilities that arise from taking derivatives of the algebraic variables can be avoided.

6.2.4 Adaptive Grid & Mesh Refinement for QCPI

One of the major limitations of QCPI in its current form is that it uses a fixed-size mesh for representing the solution which has nodes clustered at the beginning and end of the trajectory. While this may be ideal in some cases, those problems that have highly sensitive dynamics towards the middle of the trajectory may be difficult to solve using the current implementation. This limitation was explored before in more detail in Section 4.3.4. This is very similar to the challenges encountered when using by collocation-based numerical methods such as *bvp4c* or direct solvers like GPOPS.

These solvers use an adaptive mesh-sizing and mesh refinement strategy that adjusts the points where the nodes are clustered based on the sensitivity of the problem.

Some strategies for overcoming this limitation are described in Refs 88, 153, and 154. A similar strategy can be developed for QCPI that uses multiple sets of Chebyshev polynomial series starting and ending at points of high numerical sensitivity. Such an adaptive mesh refinement strategy would help greatly improve the numerical stability of QCPI and make it applicable to a wider range of nonlinear optimal control problems. Ref. 181 describes a method for choosing the order of the Chebyshev series based on the desired accuracy of the solution. This is another strategy that could be implemented in QCPI to make it more robust.

6.2.5 Parallel Implementation of QCPI

The work in this dissertation demonstrated the inherent parallelism of the QCPI algorithm using benchmarks on a multi-core computer. The various components of QCPI – independent evaluation of dynamic equations, matrix multiplication operations, and linear algebra, are all operations that can be very efficiently implemented on parallel processors. The MCPI algorithm that this method was built on was demonstrated to be very efficient at leveraging GPU processors [31, 146]. Similarly the QCPI can also greatly benefit from the highly parallel computing environment offered by GPUs and significantly accelerate the numerical solution of large-scale nonlinear boundary value problems.

6.2.6 On-board Trajectory Optimization and Model Predictive Control

Model predictive control is an advanced method of process control where the control is generated by repeatedly solving a numerical optimization problem to a finite time-horizon. Since the optimal control problem is solved for successive starting points in a short time-span, the prior solutions can be used as a very good initial guess for the next iteration. The performance of QCPI with an eight-core processor was demonstrated in Section 3.5. The run-time values illustrated in that section was for starting from a trivial initial guess and evolving the problem to the desired parameters. When starting with a very good initial guess, as in the case of MPC, such a continuation process is not required. This makes QCPI a very good candidate for implementing model predictive control or onboard trajectory optimization.

The NVIDIA Jetson is a credit-card sized system-on-a-chip (SoC) that contains a suite of input, output, and processing hardware including an NVIDIA GPU [182–184]. Ref. 185 shows that the form factor of the Jetson TX1 computer allows its use in a small satellite such as a Cubesat for image processing tasks such as image recognition, object detection and localization, and image segmentation. The newer version of this platform, the NVIDIA Jetson TX2 [186] has even more processing power. Combined with a highly parallel implementation as outlined in the previous section, QCPI would be capable of achieving real-time performance on such a computing platform even for scenarios with moderately complex dynamic systems. This could be used for performing real-time trajectory planning to account for perturbations form the reference trajectory during flight or planning optimal maneuvers in-flight. In military applications, such a system could be used for real-time trajectory planning for missile avoidance, optimal adversary engagement strategies, and countermeasure maneuvers for pilot assistance or fully autonomous vehicles.

6.2.7 Expansion of *beluga*

Being a free, open source software project allows for wide adoption by the design community as well as faster growth using community contributions. In order to fulfill this objective, *beluga* is designed to be highly customizable with a rich API that allows implementation of new numerical methods, new optimal control algorithms, continuation strategies, etc. For example, the current version of *beluga* includes fixed-step and bisection strategies for performing homotopy continuation while solving optimal control problems. *beluga* also provides an interface for the implementation of more advanced continuation strategies such as those that leverage graph search algorithms [80] which can automate the continuation process even further putting less of a burden on the designer. Another possible improvement would be support for multi-phase optimal control problems as described in Ref. 187.

beluga currently does not treat multi-vehicle problems any differently from single vehicle problems. Every state is considered to have a unique dynamic equation, and they are not grouped together in any way. This results in inefficiencies in both run-time and compile-time in some cases as shown in Chapter 3. The necessary conditions for multiple vehicles with the same dynamic models could be formulated to more efficiently to reuse code and leverage parallelization. This would further improve performance when solving multi-vehicle problems.

REFERENCES

REFERENCES

[1] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 10 Jan, 2018].

[2] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[3] Open Source Initiative and others. The MIT license, 2006.

[4] Timothy R Jorris, Christopher S Schulz, Franklin R Friedl, and Anil V Rao. Constrained trajectory optimization using pseudospectral methods. In *Proceedings of the AIAA atmospheric flight mechanics conference and exhibit*, 2008.

[5] Isaac M Ross, Christopher D'Souza, Fariba Fahroo, and JB Ross. A fast approach to multi-stage launch vehicle trajectory optimization. In *aiaa Guidance, Navigation, and control conference and Exhibit*, volume 11, page 14, 2003.

[6] Michael A. Patterson and Anil 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 Trans. Math. Softw.*, 41(1):1:1–1:37, October 2014.

[7] I Michael Ross. A beginner's guide to DIDO: a MATLAB application package for solving optimal control problems. 2007.

[8] John T Betts. Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control and Dynamics*, 21(2):193–207, 1998.

[9] F Fahroo, DB Doman, and AD Ngo. Modeling issues in footprint generation for reusable launch vehicles. In *Proceedings of the 2003 IEEE Aerospace Conference*, volume 6, 2003.

[10] J.T. Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*. Advances in Design and Control. Society for Industrial and Applied Mathematics, 2001.

[11] LS Pontryagin, VG Boltyanskii, RV Gamkrelidze, and EF Mishchenko. Mathematical Theory of Optimal Processes. *NY Google Scholar*, 1962.

[12] A. E. Bryson and Y. C. Ho. *Applied Optimal Control — Optimization, Estimation, and Control*. Taylor & Francis, 1975.

[13] Michael J Grant and Robert D Braun. Rapid Indirect Trajectory Optimization for Conceptual Design of Hypersonic Missions. *AIAA Journal of Spacecraft and Rockets*, pages 1–6, 2014.

[14] Michael J Grant and Thomas Antony. Rapid Indirect Trajectory Optimization of a Hypothetical Long Range Weapon System. In *AIAA Atmospheric Flight Mechanics Conference*, page 0276, Jan 2016.

[15] Michael J Grant and Michael A Bolender. Minimum terminal energy optimizations of hypersonic vehicles using indirect methods. In *AIAA Atmospheric Flight Mechanics Conference*, page 2402, 2015.

[16] Michael Grant, Ian Clark, and Robert Braun. Rapid Simultaneous Hypersonic Aerodynamic and Trajectory Optimization Using Variational Methods. Guidance, Navigation, and Control and Co-located Conferences. American Institute of Aeronautics and Astronautics, Aug 2011. 0.

[17] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[18] Paolo Falcone. *Nonlinear model predictive control for autonomous vehicles*. PhD thesis, Università del Sannio, 2007.

[19] Craig Earl Beal and J Christian Gerdes. Model predictive control for vehicle stabilization at the limits of handling. *IEEE Transactions on Control Systems Technology*, 21(4):1258–1269, 2013.

[20] Yiqi Gao, Theresa Lin, Francesco Borrelli, Eric Tseng, and Davor Hrovat. Predictive control of autonomous ground vehicles with obstacle avoidance on slippery roads. In *ASME 2010 dynamic systems and control conference*, volume 1, pages 265–272. American Society of Mechanical Engineers, 2010.

[21] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *Intelligent Vehicles Symposium (IV), 2015 IEEE*, pages 1094–1099. IEEE, 2015.

[22] Nazareth Bedrossian, Sagar Bhatt, Mike Lammers, Louis Nguyen, and Yin Zhang. First ever flight demonstration of zero propellant maneuver attitude control concept. In *Proceedings of the 2007 Guidance, Navigation and Control Conference*, pages 1–12, 2007.

[23] Nguyen X Vinh, Adolf Busemann, and Robert D Culp. Hypersonic and planetary entry flight mechanics. *NASA STI/Recon Technical Report A*, 81, 1980.

[24] Qin Weiwei, He Bing, Liu Gang, and Zhao Pengtao. Robust model predictive tracking control of hypersonic vehicles in the presence of actuator constraints and input delays. *Journal of the Franklin Institute*, 353(17):4351 – 4367, 2016.

[25] Xiangyuan Tao, Ning Li, and Shaoyuan Li. Multiple model predictive control for large envelope flight of hypersonic vehicle systems. *Information Sciences*, 328(Supplement C):115 – 126, 2016.

[26] Weiqiang Tang, Wenkun Long, and Haiyan Gao. Model predictive control of hypersonic vehicles accommodating constraints. *IET Control Theory & Applications*, 11(15):2599–2606, 2017.

[27] Stephen W. Keckler, William J. Dally, Brucek Khailany, Michael Garland, and David Glasco. GPUs and the Future of Parallel Computing. *IEEE Micro*, 31(5):7–17, 2011.

[28] Andre R Brodtkorb, Christopher Dyken, Trond R Hagen, Jon M Hjelmervik, and Olaf O Storaasli. State-of-the-art in heterogeneous computing. *Scientific Programming*, 18(1):1–33, 2010.

[29] Vikas Agarwal, M. S. Hrishikesh, Stephen W. Keckler, and Doug Burger. Clock Rate Versus IPC: The End of the Road for Conventional Microarchitectures. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ISCA '00, pages 248–259, New York, NY, USA, 2000. ACM.

[30] Thomas Antony and Michael J Grant. Rapid Indirect Trajectory Optimization on Highly Parallel Computing Architectures. *Journal of Spacecraft and Rockets*, 54(5):1081–1091.

[31] Xiaoli Bai. *Modified Chebyshev-Picard Iteration Methods for Solution of Initial Value and Boundary Value Problems*. PhD thesis, Texas A&M University, 2010.

[32] Michael Sparapany. Towards the Real-Time Application of Indirect Methods for Hypersonic Missions. Master's thesis, Purdue University, West Lafayette, 2015.

[33] Derek F. Lawden. *Optimal trajectories for space navigation*. Butterworths, 1963.

[34] Oskar Von Stryk and Roland Bulirsch. Direct and indirect methods for trajectory optimization. *Annals of operations research*, 37(1):357–373, 1992.

[35] Michael J. Grant. *Rapid Simultaneous Hypersonic Aerodynamic and Trajectory Optimization for Conceptual Design*. PhD thesis, Georgia Institute of Technology, 2012.

[36] David Hull. Conversion of optimal control problems into parameter optimization problems. Guidance, Navigation, and Control and Co-located Conferences. American Institute of Aeronautics and Astronautics, Jul 1996. 0.

[37] Dieter Kraft. On converting optimal control problems into nonlinear programming problems. In *Computational mathematical programming*, pages 261–280. Springer, 1985.

[38] Philip E. Gill, Walter Murray, and Michael A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM journal on optimization*, 12(4):979–1006, 2002.

[39] Charles R Hargraves and Stephen W Paris. Direct trajectory optimization using nonlinear programming and collocation. *Journal of Guidance, Control, and Dynamics*, 10(4):338–342, 1987.

[40] von Stryk, Oskar and Bulirsch, R. and Miele, A. and Stoer, J. and Well, K. *Numerical Solution of Optimal Control Problems by Direct Collocation*, pages 129–143. Birkhäuser Basel, Basel, 1993.

[41] David A Benson, Geoffrey T Huntington, Tom P Thorvaldsen, and Anil V Rao. Direct trajectory optimization and costate estimation via an orthogonal collocation method. *Journal of Guidance Control and Dynamics*, 29(6):1435, 2006.

[42] B. A. Conway and K. M. Larson. Collocation Versus Differential Inclusion in Direct Optimization. *Journal of Guidance, Control, and Dynamics*, 21(5):780–785, Sep 1998.

[43] Fariba Fahroo and I Michael Ross. Direct trajectory optimization by a chebyshev pseudospectral method. *Journal of Guidance, Control, and Dynamics*, 25(1):160–166, 2002.

[44] David Benson. *A Gauss pseudospectral transcription for optimal control*. PhD thesis, Massachusetts Institute of Technology, 2005.

[45] Timothy R Jorris and Richard G Cobb. Multiple method 2-d trajectory optimization satisfying waypoints and no-fly zone constraints. *Journal of Guidance, Control, and Dynamics*, 31(3):543, 2008.

[46] Timothy R Jorris and Richard G Cobb. Three-dimensional trajectory optimization satisfying waypoint and no-fly zone constraints. *Journal of Guidance, Control, and Dynamics*, 32(2):551, 2009.

[47] Scott Josselyn and I Michael Ross. Rapid verification method for the trajectory optimization of reentry vehicles. *Journal of Guidance Control and Dynamics*, 26(3):505–507, 2003.

[48] Qi Gong, I Michael Ross, Wei Kang, and Fariba Fahroo. On the pseudospectral covector mapping theorem for nonlinear optimal control. In *Decision and Control, 2006 45th IEEE Conference on*, pages 2679–2686. IEEE, 2006.

[49] Qi Gong, I Michael Ross, Wei Kang, and Fariba Fahroo. Connections between the covector mapping theorem and convergence of pseudospectral methods for optimal control. *Computational Optimization and Applications*, 41(3):307–335, 2008.

[50] I Michael Ross. A Historical Introduction to the Convector Mapping Principle. In *Proceedings of Astrodynamics Specialists Conference*. Naval Postgraduate School (US), 2005.

[51] H.G. Bock and K.J. Plitt. A multiple shooting algorithm for direct solution of optimal control problems*. *IFAC Proceedings Volumes*, 17(2):1603 – 1608, 1984. 9th IFAC World Congress: A Bridge Between Control Science and Technology, Budapest, Hungary, 2-6 July 1984.

[52] Gan Chen, Zi-ming Wan, Min Xu, and Si-lu Chen. Genetic Algorithm Optimization of RLV Reentry Trajectory. International Space Planes and Hypersonic Systems and Technologies Conferences. American Institute of Aeronautics and Astronautics, May 2005. 0.

[53] Nobuhiro Yokoyama and Shinji Suzuki. Modified genetic algorithm for constrained trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 28(1):139–144, Jan 2005.

[54] Michael J Grant and Gavin F Mendeck. Mars science laboratory entry optimization using particle swarm methodology. In *AIAA atmospheric flight mechanics conference and exhibit*, page 6393, 2007.

[55] Mauro Pontani and Brace A Conway. Particle swarm optimization applied to space trajectories. *Journal of Guidance, Control and Dynamics*, 33(5):1429–1441, 2010.

[56] Xiuqiang Jiang and Shuang Li. Mars atmospheric entry trajectory optimization via particle swarm optimization and gauss pseudo-spectral method. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 230(12):2320–2329, 2016.

[57] SW Paris, BK Joosten, and LE Fink. Application of an advanced trajectory optimization method to ramjet propelled missiles. *Optimal Control Applications and Methods*, 1(4):319–334, 1980.

[58] Charles Hargraves, Forrester Johnson, Stephen Paris, and Ian Retties. Numerical computation of optimal atmospheric trajectories. *Journal of Guidance, Control, and Dynamics*, 4(4):406–414, 1981.

[59] C Hargraves. Numerical computation of optimal atmospheric trajectories involving staged vehicles. In *20th Aerospace Sciences Meeting*, page 360, 1982.

[60] Jane Cullum. Finite-dimensional approximations of state-constrained continuous optimal control problems. *SIAM Journal on Control*, 10(4):649–670, 1972.

[61] Renjith R Kumar and Hans Seywald. Should controls be eliminated while solving optimal control problems via direct methods? *Journal of Guidance Control and Dynamics*, 19:418–423, 1996.

[62] Kevin P Bollino. High-fidelity real-time trajectory optimization for reusable launch vehicles. 2006.

[63] I. Michael Ross, Qi Gong, and Pooya Sekhavat. Low-thrust, high-accuracy trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 30(4):921–933, Jul 2007.

[64] Qi Gong, LR Lewis, and I Michael Ross. Pseudospectral motion planning for autonomous vehicles. *Journal of Guidance, Control, and Dynamics*, 32(3):1039–1045, 2009.

[65] Ian D Cowling, Oleg A Yakimenko, James F Whidborne, and Alastair K Cooke. Direct method based control system for an autonomous quadrotor. *Journal of Intelligent & Robotic Systems*, 60(2):285–316, 2010.

[66] Fariba Fahroo and I Michael Ross. Costate estimation by a legendre pseudospectral method. *Journal of Guidance, Control, and Dynamics*, 24(2):270–277, 2001.

[67] Divya Garg, Michael A Patterson, Camila Francolin, Christopher L Darby, Geoffrey T Huntington, William W Hager, and Anil 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.

[68] James Ferguson. A Brief Survey of the History of the Calculus of Variations and its Applications. *arXiv preprint math/0402357*, 2004.

[69] Miguel de Icaza Herrera. Galileo, Bernoulli, Leibniz and Newton around the Brachistochrone Problem. *Rev Mexicana Fis*, 40(3):459–475, 1994.

[70] Cornelius Lanczos. *The variational principles of mechanics*, volume 4. Courier Dover Publications, 1970.

[71] James M. Longuski, José J. Guzmán, and John E. Prussing. *Optimal Control with Aerospace Applications*. Springer, 2014.

[72] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[73] Wolfram Research, Inc. Mathematica, Version 11.2. Champaign, IL, 2017.

[74] The MathWorks Inc. *Symbolic Math Toolbox version 8.0.0.* Natick, Massachusetts, 2017.

[75] Michael J Grant and Michael A Bolender. Rapid, Robust Trajectory Design Using Indirect Optimization Methods. In *AIAA Atmospheric Flight Mechanics Conference*, page 2401, 2015.

[76] Geethu Lisba Jacob, Geethu Neeler, and RV Ramanan. Mars entry mission bank profile optimization. *Journal of Guidance, Control, and Dynamics*, 2014.

[77] Tieding Guo, Fanghua Jiang, and Junfeng Li. Homotopic approach and pseudospectral method applied jointly to low thrust trajectory optimization. *Acta Astronautica*, 71(Supplement C):38 – 50, 2012.

[78] H. Julian Allen and Alfred J. Eggers. *A study of the motion and aerodynamic heating of missiles entering the earth's atmosphere at high supersonic speeds.* National Advisory Committee for Aeronautics, 1957.

[79] Thomas Antony. Rapid Indirect Trajectory Optimization on Highly Parallel Computing Architectures. Master's thesis, Purdue University, West Lafayette, 2014.

[80] Justin R Mansell. Adaptive continuation strategies for indirect trajectory optimization. Master's thesis, Purdue University, 2017.

[81] Benjamin R Saunders. *Optimal Trajectory Design Under Uncertainty.* PhD thesis, Massachusetts Institute of Technology, 2012.

[82] Owens, John D and Houston, Mike and Luebke, David and Green, Simon and Stone, John E and Phillips, James C. GPU computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[83] John H Ahlberg and T Ito. A collocation method for two-point boundary value problems. *Mathematics of Computation*, 29(131):761–776, 1975.

[84] R. D. Russell and L. F. Shampine. A collocation method for boundary value problems. *Numerische Mathematik*, 19(1):1–28, Feb 1972.

[85] R. D'Ambrosio, M. Ferro, Z. Jackiewicz, and B. Paternoster. Two-step almost collocation methods for ordinary differential equations. *Numerical Algorithms*, 53(2):195–217, Mar 2010.

[86] James M Varah. A comparison of some numerical methods for two-point boundary value problems. *Mathematics of Computation*, 28(127):743–755, 1974.

[87] Jacek Kierzenka and Lawrence F Shampine. A bvp solver based on residual control and the maltab pse. *ACM Transactions on Mathematical Software (TOMS)*, 27(3):299–316, 2001.

[88] Lawrence F Shampine, Jacek Kierzenka, and Mark W Reichelt. Solving boundary value problems for ordinary differential equations in matlab with bvp4c. *Tutorial notes*, 2000:1–27, 2000.

[89] Jacek Kierzenka and Lawrence F Shampine. A bvp solver that controls residual and error. *Journal of Numerical Analysis, Industrial and Applied Mathematics*, 3:27–41, 2008.

[90] RD Russell and J Christiansen. Adaptive mesh selection strategies for solving boundary value problems. *SIAM Journal on Numerical Analysis*, 15(1):59–80, 1978.

[91] Xiaoli Bai and John L Junkins. Modified Chebyshev-Picard iteration Methods for Solution of Boundary Value Problems. *The Journal of the Astronautical Sciences*, 58(4):615–642, 2011.

[92] Robyn M Woollands, Julie L Read, Brent Macomber, Austin Probe, Ahmad Bani Younes, and John L Junkins. Method of particular solutions and kustaanheimo-stiefel regularized picard iteration for solving two-point boundary value problems. In *AAS/AIAA Spce Flight Meeting, Williamsburg, VA*, 2015.

[93] Earl A Coddington and Norman Levinson. *Theory of Ordinary Differential Equations*. Tata McGraw-Hill Education, 1955.

[94] JP Riehl, SW Paris, and WK Sjauw. Comparison of implicit integration methods for solving aerospace trajectory optimization problems. In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, pages 21–24, 2006.

[95] GL Brauer, DEj Cornick, and R_ Stevenson. Capabilities and applications of the program to optimize simulated trajectories (post). program summary document. 1977.

[96] The MathWorks Inc. MATLAB, 2013.

[97] G Basset, Yunjun Xu, and OA Yakimenko. Computing short-time aircraft maneuvers using direct methods. *Journal of Computer and Systems Sciences International*, 49(3):481–513, 2010.

[98] Geoffrey T Huntington and Anil V Rao. Optimal reconfiguration of spacecraft formations using the gauss pseudospectral method. *Journal of Guidance Control and Dynamics*, 31(3):689–698, 2008.

[99] Georges S Aoude, Jonathan P How, and Ian M Garcia. Two-stage path planning approach for solving multiple spacecraft reconfiguration maneuvers. *The Journal of the Astronautical Sciences*, 56(4):515–544, 2008.

[100] Kathryn F Graham and Anil V Rao. Minimum-time trajectory optimization of multiple revolution low-thrust earth-orbit transfers. *Journal of Spacecraft and Rockets*, 2015.

[101] Yanning Guo, Matt Hawkins, and Bong Wie. Waypoint-optimized zero-effort-miss/zero-effort-velocity feedback guidance for mars landing. *Journal of Guidance, Control, and Dynamics*, 2013.

[102] Joel Benito and Robert Shotwell. Trajectory design for a mars ascent vehicle concept terrestrial demonstration. In *Aerospace Conference, 2017 IEEE*, pages 1–7. IEEE, 2017.

[103] Victor M Becerra. Solving complex optimal control problems at no cost with psopt. In *Computer-Aided Control System Design (CACSD), 2010 IEEE International Symposium on*, pages 1391–1396. IEEE, 2010.

[104] Michael J Grant, Ian G Clark, and Robert D Braun. Rapid Design Space Exploration for Conceptual Design of Hypersonic Missions. In *AIAA Atmospheric Flight Mechanics Conference*, 2011.

[105] Guido Rossum. Python reference manual. Technical report, Amsterdam, The Netherlands, The Netherlands, 1995.

[106] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, pages 7:1–7:6, New York, NY, USA, 2015. ACM.

[107] Shunsaku Arita and Seiya Ueno. Optimal feedback guidance for nonlinear missile model with impact time and angle constraints. In *AIAA Guidance, Navigation, and Control (GNC) Conference*, volume 4785, 2013.

[108] A. Miele, A. Mangiavacchi, and A. K. Aggarwal. Modified quasilinearization algorithm for optimal control problems with nondifferential constraints. *Journal of Optimization Theory and Applications*, 14(5):529–556, Nov 1974.

[109] Cristiana Silva and Emmanuel Trélat. Smooth regularization of bang-bang optimal control problems. *IEEE Transactions on Automatic Control*, 55(11):2488–2499, 2010.

[110] Kshitij Mall and Michael J. Grant. Trigonomerization of optimal control problems with bounded controls. AIAA AVIATION Forum. American Institute of Aeronautics and Astronautics, Jun 2016. 0.

[111] Knut Graichen, Andreas Kugi, Nicolas Petit, and Francois Chaplais. Handling constraints in optimal control with saturation functions and system extension. *Systems & Control Letters*, 59(11):671–679, 2010.

[112] Stephen L Campbell and B Leimkuhler. Differentiation of constraints in differential-algebraic equations. *Journal of Structural Mechanics*, 19(1):19–39, 1991.

[113] Kathryn Eleda Brenan, Stephen L Campbell, and Linda Ruth Petzold. *Numerical solution of initial-value problems in differential-algebraic equations*. SIAM, 1995.

[114] Charles William Gear and Linda Ruth Petzold. Ode methods for the solution of differential/algebraic systems. *SIAM Journal on Numerical Analysis*, 21(4):716–728, 1984.

[115] Stephen L Campbell. The numerical solution of higher index linear time varying singular systems of differential equations. *SIAM journal on scientific and statistical computing*, 6(2):334–348, 1985.

[116] Kathryn E Brenan and Björn E Engquist. Backward differentiation approximations of nonlinear differential/algebraic systems. *Mathematics of Computation*, 51(184):659–676, 1988.

[117] Charles William Gear. Maintaining solution invariants in the numerical solution of odes. *SIAM journal on scientific and statistical computing*, 7(3):734–743, 1986.

[118] Rafael de P Soares and Argimiro R Secchi. Direct initialisation and solution of high-index dae systems. *Computer Aided Chemical Engineering*, 20:157–162, 2005.

[119] Claus Führer and BJ Leimkuhler. Numerical solution of differential-algebraic equations for constrained mechanical motion. *Numerische Mathematik*, 59(1):55–69, 1991.

[120] C. W. Gear. Differential-algebraic equation index transformations. *SIAM Journal on Scientific and Statistical Computing*, 9(1):39–47, 1988.

[121] K Brenan. Stability and convergence of difference approximations for higher index differential/algebraic equations with applications to trajectory control. *PhD Disertation, Math. Dept. UCLA, Los Angeles*, 1983.

[122] Per Lötstedt and Linda Petzold. Numerical solution of nonlinear differential equations with algebraic constraints. i. convergence results for backward differentiation formulas. *Mathematics of computation*, 46(174):491–516, 1986.

[123] Linda Petzold and Per Lötstedt. Numerical solution of nonlinear differential equations with algebraic constraints ii: Practical implications. *SIAM Journal on Scientific and Statistical Computing*, 7(3):720–733, 1986.

[124] Joachim Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer methods in applied mechanics and engineering*, 1(1):1–16, 1972.

[125] LF Shampine. Conservation laws and the numerical solution of odes. *Computers & Mathematics with Applications*, 12(5-6):1287–1296, 1986.

[126] LF Shampine. Conservation laws and the numerical solution of odes, ii. *Computers & Mathematics with Applications*, 38(2):61–72, 1999.

[127] Kenneth Sutton and Randolph A. Graves Jr. A general stagnation-point convective heating equation for arbitrary gas mixtures. Technical report, National Aeronautics and Space Administration, 1971.

[128] Yun Fei, Guodong Rong, Bin Wang, and Wenping Wang. Parallel L-BFGS-B Algorithm on GPU. *Computers & Graphics*, 40:1–9, 2014.

[129] NVIDIA Corporation. Cuda C Programming Guide, 2014.

[130] Pafnuti Lvovitch Tchebychev. *Théorie des mécanismes connus sous le nom de parallélogrammes*. Imprimerie de l'Académie impériale des sciences, 1853.

[131] L Fox and IB Parker. Chebyshev polynomials in numerical analysis. 1968.

[132] John C Mason and David C Handscomb. *Chebyshev polynomials*. CRC Press, 2002.

[133] CW Clenshaw. The numerical solution of linear differential equations in chebyshev series. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 53, pages 134–149. Cambridge University Press, 1957.

[134] CW Clenshaw and HJ Norton. The solution of nonlinear ordinary differential equations in chebyshev series. *The Computer Journal*, 6(1):88–92, 1963.

[135] K Wright. Chebyshev collocation methods for ordinary differential equations. *The Computer Journal*, 6(4):358–365, 1964.

[136] David Gottlieb and Steven A Orszag. *Numerical analysis of spectral methods: theory and applications*. SIAM, 1977.

[137] Mehmet Sezer and Mehmet Kaynak. Chebyshev polynomial solutions of linear differential equations. *International Journal of Mathematical Education in Science and Technology*, 27(4):607–618, 1996.

[138] John P Boyd. *Chebyshev and Fourier spectral methods*. Courier Corporation, 2001.

[139] Minoru Urabe. Numerical solution of multi-point boundary value problems in chebyshev series theory of the method. *Numerische Mathematik*, 9(4):341–366, 1967.

[140] Jacques Vlassenbroeck and Rene Van Dooren. A chebyshev technique for solving nonlinear optimal control problems. *IEEE transactions on automatic control*, 33(4):333–340, 1988.

[141] Gamal N Elnagar and Mohammad A Kazemi. Pseudospectral chebyshev optimal control of constrained nonlinear dynamical systems. *Computational Optimization and Applications*, 11(2):195–217, 1998.

[142] T Feagin. *The numerical solution of two point boundary value problems using chebyshev series*. PhD thesis, Ph. D. dissertation, The Universtiy of Texas at Austin, Austin, TX, 1973.

[143] J Shaver. *Formulation and evaluation of parallel algorithms for the orbit determination problem*. PhD thesis, 1980.

[144] Terry Feagin and Paul Nacozy. Matrix formulation of the picard method for parallel computation. *Celestial Mechanics and Dynamical Astronomy*, 29(2):107–115, 1983.

[145] Toshio Fukushima. Vector integration of dynamical motions by the picard-chebyshev method. *The Astronomical Journal*, 113:2325, 1997.

[146] Darin Koblick, Mark Poole, and Praveen Shankar. Parallel High-Precision Orbit Propagation using the Modified Picard-Chebyshev Method. In *ASME 2012 International Mechanical Engineering Congress and Exposition*, pages 587–605. American Society of Mechanical Engineers, 2012.

[147] Thomas Antony and Michael J. Grant. A Generalized Adaptive Chebyshev–Picard Iteration Method for Solution to Two–Point Boundary Value Problems. In *3rd Annual Meeting of the AFRL Mathematical Modeling and Optimization Institute*, 2015.

[148] Angelo Miele. Method of particular solutions for linear, two-point boundary-value problems. *Journal of Optimization Theory and Applications*, 2(4):260–273, Jul 1968.

[149] Angelo Miele and RR Iyer. General technique for solving nonlinear, two-point boundary-value problems via the method of particular solutions. *Journal of Optimization Theory and Applications*, 5(5):382–399, 1970.

[150] A. Miele, R. R. Iyer, and K. H. Well. Modified quasilinearization and optimal initial choice of the multipliers part 2—optimal control problems. *Journal of Optimization Theory and Applications*, 6(5):381–409, Nov 1970.

[151] S. Gonzalez and S. Rodriguez. Modified quasilinearization algorithm for optimal control problems with nondifferential constraints and general boundary conditions. *Journal of Optimization Theory and Applications*, 50(1):109–128, Jul 1986.

[152] E Hairer, S.P. Norsett, and G. Wanner. Solving ordinary, differential equations i, nonstiff problems. 2000.

[153] Christopher L Darby, William W Hager, and Anil V Rao. An hp-adaptive pseudospectral method for solving optimal control problems. *Optimal Control Applications and Methods*, 32(4):476–502, 2011.

[154] Christopher L Darby, William W Hager, Anil V Rao, et al. Direct trajectory optimization using a variable low-order adaptive pseudospectral method. *Journal of Spacecraft and Rockets*, 48(3):433, 2011.

[155] Michael J Grant. *Rapid Simultaneous Hypersonic Aerodynamic and Trajectory Optimization for Conceptual Design*. PhD thesis, Georgia Institute of Technology, 2012.

[156] Thomas Antony, Michael J. Grant, and Michael A. Bolender. Optimization of Interior Point Cost Functions Using Indirect Methods. AIAA AVIATION Forum. American Institute of Aeronautics and Astronautics, Jun 2015.

[157] Michael J. Grant and Michael A. Bolender. Rapid, Robust Trajectory Design Using Indirect Optimization Methods. AIAA AVIATION Forum. American Institute of Aeronautics and Astronautics, Jun 2015. 0.

[158] Kshitij Mall and Michael J. Grant. High mass mars exploration using slender entry vehicles. AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, Jan 2016. 0.

[159] Kshitij Mall and Michael J. Grant. Epsilon-trig regularization method for bang-bang optimal control problems. AIAA AVIATION Forum. American Institute of Aeronautics and Astronautics, Jun 2016. 0.

[160] Janav P. Udani, Kshitij Mall, Michael J. Grant, and Dengfeng Sun. Optimal flight trajectory to minimize noise during landing. AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, Jan 2017. 0.

[161] Joseph Williams, Kshitij Mall, and Michael J. Grant. Trajectory optimization using indirect methods and parametric scramjet cycle analysis. AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, Jan 2017. 0.

[162] Pearu Peterson. F2py: a tool for connecting fortran and python programs. *International Journal of Computational Science and Engineering*, 4(4):296–305, 2009.

[163] Travis E Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3), 2007.

[164] John D Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[165] Luke Barnard and Matej Mertik. Usability of visualization libraries for web browsers for use in scientific analysis. *International Journal of Computer Applications*, 121(1), 2015.

[166] Christoph Weniger. A tentative gamma-ray line from dark matter annihilation at the fermi large area telescope. *Journal of Cosmology and Astroparticle Physics*, 2012(08):007, 2012.

[167] Helge Eichhorn and Reiner Anderl. Plyades: A Python Library for Space Mission Design. *arXiv preprint arXiv:1607.00849*, 2016.

[168] Andy Buckley, Hendrik Hoeth, Heiko Lacker, Holger Schulz, and Jan Eike von Seggern. Systematic event generator tuning for the lhc. *The European Physical Journal C-Particles and Fields*, 65(1):331–357, 2010.

[169] JR Johansson, PD Nation, and Franco Nori. Qutip: An open-source python framework for the dynamics of open quantum systems. *Computer Physics Communications*, 183(8):1760–1772, 2012.

[170] Michael J Keiser, Bryan L Roth, Blaine N Armbruster, Paul Ernsberger, John J Irwin, and Brian K Shoichet. Relating protein pharmacology by ligand chemistry. *Nature biotechnology*, 25(2):197–206, 2007.

[171] Edward Anderson, Zhaojun Bai, Christian Bischof, L Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, et al. *LAPACK Users' guide*. SIAM, 1999.

[172] Alan C Hindmarsh. Odepack, a systematized collection of ode solvers, rs stepleman et al.(eds.), north-holland, amsterdam,(vol. 1 of), pp. 55-64. *IMACS transactions on scientific computation*, 1:55–64, 1983.

[173] Bokeh Development Team. *Bokeh: Python library for interactive visualization*, 2014.

[174] Timothy M Shead. toyplot. Technical report, Sandia National Laboratory, 2014.

[175] Audris Mockus, Roy T Fielding, and James D Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3):309–346, 2002.

[176] Karim R Lakhani and Eric Von Hippel. How open source software works: "free" user-to-user assistance. *Research policy*, 32(6):923–943, 2003.

[177] Eric von Hippel and Georg von Krogh. Open source software and the "private-collective" innovation model: Issues for organization science. *Organization science*, 14(2):209–223, 2003.

[178] Karim R Lakhani, Robert G Wolf, et al. Why hackers do what they do: Understanding motivation and effort in free/open source software projects. *Perspectives on free and open source software*, 1:3–22, 2005.

[179] Andreas Griewank, David Juedes, and Jean Utke. Algorithm 755: Adol-c: a package for the automatic differentiation of algorithms written in c/c++. *ACM Transactions on Mathematical Software (TOMS)*, 22(2):131–167, 1996.

[180] Joel Andersson, Johan Åkesson, and Moritz Diehl. Casadi: A symbolic package for automatic differentiation and optimal control. In *Recent Advances in Algorithmic Differentiation*, pages 297–307. Springer, 2012.

[181] P. E. Nacozy and T. Feagin. Approximations of interplanetary trajectories by chebyshev series. *AIAA Journal*, 10(3):243–244, 1972.

[182] Nvidia: Embedded systems developer kits.

[183] Augusto Vega, Chung-Ching Lin, Karthik Swaminathan, Alper Buyuktosunoglu, Sharathchandra Pankanti, and Pradip Bose. Resilient, uav-embedded real-time computing. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on*, pages 736–739. IEEE, 2015.

[184] Yash Ukidave, David Kaeli, Umesh Gupta, and Kurt Keville. Performance of the NVIDIA Jetson TK1 in HPC. In *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*, pages 533–534. IEEE, 2015.

[185] Nick Buonaiuto, Mark Louie, Jim Aarestad, Rohit Mital, Dennis Mateik, Robert Sivilli, Apoorva Bhopale, Craig Kief, and Brian Zufelt. Satellite Identification Imaging for Small Satellites Using NVIDIA. In *Small Satellite Conference*, 2017.

[186] Tanya Amert, Nathan Otterness, Ming Yang, James H Anderson, and F Donelson Smith. GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed. In *Real Time Systems Conference*, 2017.

[187] Saranathan, Harish and Grant, Michael J. Relaxed Autonomously Switched Hybrid System Approach to Indirect Multiphase Aerospace Trajectory Optimization. *Journal of Spacecraft and Rockets*, pages 1–11, 2017.

APPENDICES

A. MCPI Matrices

There are two matrices used in formulating the matrix-vector form of the Chebyshev-Picard iteration used by QCPI. The first, C_a, is used to compute the coefficients, F, for an N-th order Chebyshev Polynomial series to a given function, $g(x)$, as follows:

$$F = 2\chi_0 + C_a \times g(x) \tag{A.1}$$

where $g(x)$ is $g(x)$ evaluated on an N-th order Chebyshev mesh as:

$$g = [g(\tau_0), g(\tau_1), ..., g(\tau_N)]^T \tag{A.2}$$

and χ_0 is defined as:

$$\chi_0 = \quad 2x_0 \quad 0 \quad 0 \quad \cdots \quad 0 \quad 0 \quad ^T \tag{A.3}$$

with x_0 being the initial value of x. C_a is defined as:

$$C_a \equiv RSTV \tag{A.4}$$

where

$$R = diag \quad 1, \frac{1}{2}, \frac{1}{4}, ..., \frac{1}{2(N-1)}, \frac{1}{2N} \tag{A.5a}$$

$$S = \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{2}{3} & \frac{1}{4} & \frac{-2}{15} & \cdots & (-1)^{N+1}\frac{1}{N-1} & \\ 1 & 0 & -1 & 0 & 0 & \cdots & 0 & \\ 0 & 1 & 0 & -1 & 0 & \cdots & 0 & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & -1 & \\ 0 & 0 & 0 & 0 & \cdots & 1 & 0 & \end{bmatrix} \tag{A.5b}$$

$$T = \begin{bmatrix} T_0(\tau_0) & T_0(\tau_1) & \cdots & T_0(\tau_N) \\ T_1(\tau_0) & T_1(\tau_1) & \cdots & T_1(\tau_N) \\ T_2(\tau_0) & T_2(\tau_1) & \cdots & T_2(\tau_N) \\ \vdots & \vdots & \vdots & \vdots \\ T_N(\tau_0) & T_N(\tau_1) & \cdots & T_N(\tau_N) \end{bmatrix} \tag{A.5c}$$

$$T_k(\tau_j) = \cos\left(k \arccos \tau_j\right) \tag{A.5d}$$

$$\tau_j = \cos \quad \frac{j\pi}{N} \tag{A.5e}$$

$$V = diag \quad \frac{1}{N}, \frac{2}{N}, \frac{2}{N}, ..., \frac{2}{N}, \frac{1}{N} \qquad \text{with } N+1 \text{ elements} \tag{A.5f}$$

$$\tag{A.5g}$$

The second matrix is C_x which is used to evaluate an N-th order Chebyshev series represented by its coefficients, β on a Chebyshev grid.

$$x = C_x \times \beta \tag{A.6}$$

C_x is defined as:

$$C_x \equiv TW \tag{A.7}$$

where

$$W = diag \quad \frac{1}{2}, 1, 1, \cdots, 1, 1 \qquad \text{with } N+1 \text{ elements} \tag{A.8}$$

B. *beluga* Software Information

B.1 Obtaining & Installing *beluga*

Follow these steps to obtain and install *beluga*.

1. Install Python version 3.6 or newer. It is recommended that a package such as Anaconda be used (`https://www.anaconda.com/distribution/`) since it includes many popular scientific computing libraries such as NumPy and SciPy that are used by *beluga*
2. Download and install the latest version of *beluga* by following the instructions at `https://github.com/thomasantony/beluga`

B.2 Sample Input File

```
"""Brachistochrone example with path constraint."""

ocp = beluga.OCP('constrainedBrachistochrone')

# Define independent variables
ocp.independent('t', 's')

# Define equations of motion
ocp.state('x', 'v*cos(theta)', 'm')\
   .state('y', 'v*sin(theta)','m')\
   .state('v', 'g*sin(theta)','m/s')

# Define controls
```

```
ocp.control('theta','rad')

# Define constants
ocp.constant('g',-9.81,'m/s^2')

# Define costs
ocp.path_cost('1','s')

# Define constraints
ocp.constraints() \
    .initial('x-x_0','m') \
    .initial('y-y_0','m') \
    .initial('v-v_0','m/s')\
    .terminal('x-x_f','m') \
    .terminal('y-y_f','m')

ocp.constraints() \
    .path('constraint1','y+x','>',-1.0,'m',start_eps=1e-4)

ocp.scale(m='y', s='y/v', kg=1, rad=1, nd=1)

bvp_solver = beluga.bvp_algorithm('MultipleShooting',
                  tolerance=1e-4,
                  max_iterations=500,
                  verbose = True,
                  max_error=50,
)

guess_maker = beluga.guess_generator('auto',
                start=[0,0,1], # Starting values for states
                direction='forward',
                costate_guess = 0.1,
                control_guess = [-3.14*60/180, 0.0, 0.0],
)

continuation_steps = beluga.init_continuation()

continuation_steps.add_step('bisection') \
                .num_cases(21) \
                .terminal('x', 10) \
                .terminal('y',-10)

beluga.solve(ocp,
             method='icrm',
             bvp_algorithm=bvp_solver,
             steps=continuation_steps,
             guess_generator=guess_maker)
```

B.3 Sample Plotting Script

```
from beluga.visualization import BelugaPlot
from beluga.visualization.datasources import Dill, GPOPS

gpops_ds = GPOPS('./brachisto_eps5.mat',states=('x','y','v','xi','tf')
    → ,controls=('theta','ue1'))
plots = BelugaPlot('./data.dill',default_sol=-1,default_step=-1,
    → renderer='matplotlib')
```

```
plots.add_plot(mesh_size=None).line('x','y',label='ICRM␣Solution', sol
    → =-1, step=-1) \
                .line('x','y',label='GPOPS␣Solution', style='o',sol=-1,
                    →  step=-1, datasource=gpops_ds) \
                .line('x','-1.0-x',label='Constraint1',step=-1,sol=-1)
                    → \
                .xlabel('x(t)').ylabel('y(t)') \
                .title('Trajectory')

plots.add_plot(mesh_size=None).line('t','ue1',label='ICRM') \
                .line('t','ue1',label='GPOPS',datasource=gpops_ds,style
                    → ='o') \
                .xlabel('t␣(s)').ylabel('theta␣(degrees)') \
                .title('Control␣history')

plots.add_plot(mesh_size=None).line('t','theta*180/3.14',label='ICRM')
    →  \
                .line('t','theta*180/3.14',label='GPOPS',datasource=
                    → gpops_ds,style='o') \
                .xlabel('t␣(s)').ylabel('theta␣(degrees)') \
                .title('Control␣history')

plots.add_plot(mesh_size=None).line('t','lamX')\
                .line('t','lamY')\
                .line('t','lamV')\
                .line('t','lamXI11')\
                .line('t','lamX',datasource=gpops_ds,style='o') \
                .line('t','lamY',datasource=gpops_ds,style='o') \
                .line('t','lamV',datasource=gpops_ds,style='o') \
                .line('t','lamXI',datasource=gpops_ds,style='o') \
                .xlabel('t␣(s)').ylabel('lambda') \
                .title('lamX')
plots.render()
```

C. Publications

Upcoming Publications

1. Journal submission to Journal of Optimization Theory and Applications regarding the use of regularizing path constraints using saturation functions (ICRM)
2. Journal submission to JGCD regarding Quasilinear Chebyshev Picard Iteration
3. Journal submission to JSR about Multi-Vehicle Constrained Trajectory Optimization

Relevant Publications

1. Thomas Antony and Michael J. Grant. Path Constraint Regularization in Optimal Control Problems using Saturation Functions. In *AIAA Atmospheric Flight Mechanics Conference, AIAA SciTech 2018*. American Institute of Aeronautics and Astronautics, Jan 2018
2. Thomas Antony and Michael J Grant. Rapid Indirect Trajectory Optimization on Highly Parallel Computing Architectures. *Journal of Spacecraft and Rockets*, 54(5):1081–1091
3. Michael J Grant and Thomas Antony. Rapid Indirect Trajectory Optimization of a Hypothetical Long Range Weapon System. In *AIAA Atmospheric Flight Mechanics Conference*, page 0276, Jan 2016
4. Thomas Antony and Michael J. Grant. Rapid Indirect Trajectory Optimization on Highly Parallel Computing Architectures. *AIAA Atmospheric Flight Mechanics Conference, AIAA SciTech Forum 2016*. American Institute of Aeronautics and Astronautics, Jan 2016

Additional Publications

1. Thomas Antony, Michael J. Grant, and Michael A. Bolender. Optimization of Interior Point Cost Functions Using Indirect Methods. AIAA AVIATION Forum 2015. American Institute of Aeronautics and Astronautics, Jun 2015
2. Michael Sparapany, Thomas Antony, Harish Saranathan, Lorenz Klug, Ben Libben, Eiji Shibata, Joseph Williams, Michael J. Grant, and Sarag J. Saikia. Enabling Mars Exploration Using inflatable Purdue Aerodynamic Decelerator with Deployable Entry Systems (iPADDLES) Technology. In *13th International Planetary Probe Workshop*, June 2016.

VITA

VITA

Thomas Antony was born in Kochi, India in 1989. He earned a B.Tech in Mechanical Engineering from the Cochin University of Science & Technology in June 2011. He joined the School of Aeronautics & Astronautics at Purdue University in August 2012, working under the guidance of Prof. Michael J. Grant at the Rapid Design of Systems Laboratory (RDSL). At RDSL, he developed high performance numerical algorithms for trajectory optimization as part of a project supported by the Charles Stark Draper Laboratory (Boston, MA). Thomas earned his Masters in Aeronautics & Astronautics in December 2014 with the thesis titled, "Rapid Indirect Trajectory Optimization on Highly Parallel Computing Architectures".

In May 2015, he continued this work as a summer researcher at the Air Force Research Laboratory's Mathematical Modeling and Optimization Institute in Shalimar, FL where he developed novel numerical algorithms for trajectory optimization. Thomas worked on the design and architecture Mars entry systems, particularly Hypersonic Inflatable Aerodynamic Decelerator(HIAD) systems for the NASA Big Idea Challenge in 2016. He was part of the team that was a finalist after presenting the work at NASA Langley Research Center.

Thomas' research interests include trajectory optimization, high performance computing, guidance and control, autonomous vehicles, sensor fusion, robotics and machine learning.