REAL-TIME TRAJECTORY OPTIMIZATION BY SEQUENTIAL CONVEX PROGRAMMING FOR ONBOARD OPTIMAL CONTROL

by

Benjamin M. Tackett

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science

School of Aeronautics and Astronautics

West Lafayette, Indiana

August 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Michael Grant, Chair

School of Aeronautics and Astronautics

Dr. William Crossley

School of Aeronautics and Astronautics

Dr. Ran Dai

School of Aeronautics and Astronautics

Approved by:

Dr. Gregory A. Blaisdell

To my parents, Chistopher and Jeanne.

The contemplation of celestial things will make a man both speak and think more sublimely and magnificently when he descends to human affairs. - Marcus Tullius Cicero

To my brother, Jacob.

Most people see what they expect to see, what they want to see, what they've been told to see, what conventional wisdom tells them to see - not what is right in front of them in its pristine condition. - Vincent Bugliosi

ACKNOWLEDGMENTS

Throughout the writing of this dissertation I have received a great deal of support and assistance.

First, I would like to thank my graduate advisor, Professor Michael Grant, whose knowledge and commitment to excellence proved vital in formulating my research methodology and overcoming obstacles in pursuit of my research goals. Your encouragement to pursue advances in state-of-the-art techniques gave me the motivation to follow research questions I would not have thought to otherwise.

I would like to acknowledge my NASA Ames internship advisor, Dr. David Hash, who motivated me to pursue a graduate education to further my knowledge of aeronautics and astronautics. When my first instinct was to immediately find a job in industry following my undergraduate education, you provided me with some much needed vision.

In addition, I would like to thank my parents for their continued support of my education. I would not be where I am today without your encouragement, wisdom, and love. Lastly, I would like to thank my friends who provided me with energizing conversation and much needed fun while working towards my degree.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

R_0	radius of Earth, km
r	radial distance from Earth's center to point of interest, km
m	mass, kg
V	velocity, km/s
θ	longitude, deg
ϕ	latitude, deg
ψ	heading angle, deg
h	altitude, km
h_s	atmospheric scale height, km
s	downrange, km
g_0	approximate Earth gravity, $\mathrm{m/s^2}$
γ	flight-path angle, deg
L	lift force, N
C_L	coefficient of lift
D	drag force, N
C_D	coefficient of drag
L/D	lift to drag ratio
ρ	atmospheric density, $\mathrm{kg/m^3}$
ρ_0	atmospheric density at sea-level, $\mathrm{kg/m^3}$
σ	bank angle, deg
α	angle of attack, deg
A_{ref}	vehicle reference area, $\mathrm{m^2}$
x	state vector
u	control vector
x_f	desired terminal state
$x_{\min}$	lower bound
$x_{\max}$	upper bound
t_0	start time

t_f final time

ABBREVIATIONS

AIAA	American Institute of Aeronautics and Astronautics
AoA	Angle of Attack
CPU	Central Processing Unit
DOF	Degrees of Freedom
ECOS	Embedded Conic Solver
EOMs	Equations of Motion
GN&C	Guidance, Navigation, and Control
KKT	Karush-Kuhn-Tucker
NP	Non-deterministic Polynomial-time
RLV	Reusable Launch Vehicle
SCP	Sequential Convex Programming
SOCP	Second Order Cone Programming

ABSTRACT

Optimization of atmospheric flight control has long been performed on the ground, prior to mission flight due to large computational requirements used to solve non-linear programming problems. Onboard trajectory optimization enables the creation of new reference trajectories and updates to guidance coefficients in real time. This thesis summarizes the methods involved in solving optimal control problems in real time using convexification and Sequential Convex Programming (SCP). The following investigation provided insight in assessing the use of state of the art SCP optimization architectures and convexification of the hypersonic equations of motion[1]–[3] with different control schemes for the purposes of enabling on-board trajectory optimization capabilities.

An architecture was constructed to solve convexified optimal control problems using direct population of sparse matrices in triplet form and an embedded conic solver to enable rapid turn around of optimized trajectories. The results of this show that convexified optimal control problems can be solved quickly and efficiently which holds promise in autonomous trajectory design to better overcome unexpected environments and mission parameter changes. It was observed that angle of attack control problems can be successfully convexified and solved using SCP methods. However, the use of multiple coupled controls is not guaranteed to be successful with this method when they act in the same plane as one another. The results of this thesis demonstrate that state of the art SCP methods have the capacity to enable onboard trajectory optimization with both angle of attack control and bank angle control schemes.

1. INTRODUCTION

As autonomous systems are becoming more and more prevalent, increasing the run-time efficiency of optimization techniques has been of major interest in modern research with significant effort being concentrated on methods which speed up both the formulation and the computational optimization process. One goal of state of the art techniques is to create algorithms which can perform real-time onboard trajectory optimization leading to new capabilities for autonomous systems and enabling rapid retargeting of aerospace vehicle trajectories without human interaction. The two characteristics which define such systems are a short run-time and a small data footprint. A good onboard algorithm needs to run fast, demonstrate enough accuracy to be realistically usable, and consume a minimal amount of processing power. The first and last of these characteristics are often at odds with each other where speeding up an algorithm requires increased resources or vice-versa leading to nearly all optimization being performed on the ground where more computational power is available. This lack of adequate onboard capabilities leads to the creation of reference trajectories on the ground which are then either relayed to a vehicle or stored onboard to be later followed by the guidance scheme. This process also uses the reference trajectory to design the sensitivity coefficients of the guidance algorithm and this has been used successfully for many aerospace related missions like Mars Science Laboratory, Mars 2020, and Apollo Entry Guidance. However, this method does not provide the flexibility of a real-time onboard system which can rapidly generate new trajectories to further tune the guidance system in flight as the mission parameters and environments are updated. While it is unlikely that the use of pre-flight generated reference trajectories will cease due to the vast resources which can be used on the ground, a real-time onboard system creates an opportunity to alter the reference trajectory, responding to mission changes on the fly.

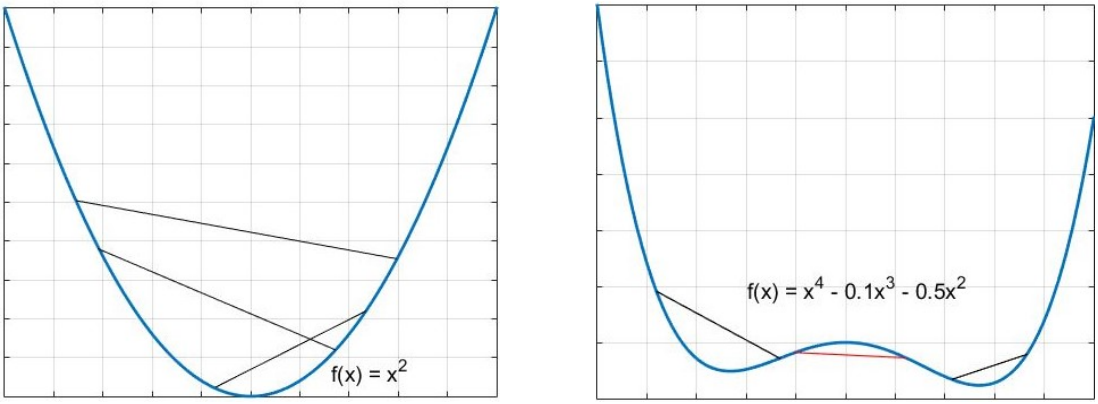
One method which solves problems quickly with high computational efficiency is convex optimization which utilizes the aspects of a convex solution space to solve problems with the speed of linear programming techniques. Two aspects in particular which make convex optimization particularly attractive are that any local minimum found within the convex solution space is by definition also the global minimum, and the sufficient conditions for

optimality are just the first-order conditions. Minimizing convex functions over convex sets also makes the process of maximizing a concave function f trivial as it can be reformulated as a minimization problem -f, which is convex.

Over the years convex optimization has had a major impact in the areas of signal processing[4], [5], electronic circuit design[6], and finance[7] leading to significant advances in these fields. These areas of study have a lot of naturally convex problems which require robust, high speed optimization to solve effectively in order to take advantage of the solutions. For example, quickly analyzing financial risk preferences naturally arises as a convex problem based on trading protocols that result in convex trading costs.[8] This allows for rapid analysis of financial assets and provides a significant advantage to those who can utilize these techniques.[7], [9] While convex optimization is indeed an extremely effective method for solving convex problems, many flight mechanics problems are by nature not convex. The equations of motion for hypersonic flight are considered highly non-linear even for three degree-of-freedom (DOF) flight, and these problems often have multiple local minima within the non-convex solution space making optimization difficult. These non-linear, non-convex hypersonic trajectory optimization problems are unable to take advantage of the characteristics of linear problems with regards to efficiency of solution solving and are unable to easily find global minima like with a convex solution space. As such, there have been several attempts made in recent years to reformulate the hypersonic trajectory equations into a convex system of equations in order to take advantage of convex optimization, which shares many characteristics with optimization of linear problems, for autonomous trajectory re-optimization. The objective of this thesis is to demonstrate a software architecture by which sequential convex programming can be used for real-time optimization of atmospheric flight trajectories as well as provide evidence that convexification and sequential convex programming can be used to solve previously unstudied angle of attack control problems.

1.1 A Historical Perspective

Convex optimization is defined as a technique for minimizing convex functions over a convex set. As such a convex objective function must lie below all of its secants and conversely, lie above all of its tangents.



(a) Convex Obj. Function (b) Non-convex Obj. Function.

Figure 1.1. Example Objective Functions.

In Figure 1.1, it is clear that (a) follows the secant and tangent rules for a convex objective function while (b) has an area where the secant can be defined *below* the objective function and the tangent *above* the objective function. Additionally, it is obvious that (a) has a single local minimum which is the global minimum and (b) has multiple local minima, where only one of them is the global minimum. These criteria show visually that (a) is convex while (b) is not. The major difference between convex and non-convex functions is that a convex function will have exactly one optimal solution which is globally optimal while a non-convex function may have several local extrema and it can take significant time to find a global optima. This makes the efficiency in time much better when solving convex problems and avoids the problem of local extrema. In addition, convex problems maintain many of the characteristics of linear problems while non-convex problems share many of the properties of general non-linear problems. These characteristics of convex and non-convex functions lead to the notion that converting a non-linear, non-convex problem into a non-linear, convex problem can allow for many of the same advantages as solving linear problems.

Convexity is often observed in linear programming, geometric programming, second order cone programming [10], and several more either as an already existing convex optimization problem or one that is reduced such that it becomes convex using simple transformation techniques. If a problem is not initially convex, it can be difficult to determine the needed transformation, should one exist, particularly when the solution space of the problem can not be observed visually. In 2013, Amir Ali Ahmadi et al. [11] attempted to determine if an arbitrary polynomial function could be checked for convexity and how programmatically hard such a process would be would be.

It was determined in this article, after 20 years of research, that determining the convexity of a particular polynomial function was non-deterministic polynomial-time (NP) hard. In computer science, a problem is considered "easy" if there is a polynomial algorithm to solve the problem, otherwise the problem is considered "hard." A NP-hard problem is defined as the set of problems whose solutions can be verified in polynomial time, but take exponential time to solve. The distinction between polynomial-time and NP is expressed as the difference between solution solving for polynomial expressions, where the generic number N is raised to a power, and expressions where a number is raised to the power of N. "If an algorithm whose execution time is proportional to N takes a second to perform a computation involving 100 elements, an algorithm whose execution time is proportional to N^3 takes almost three hours. But an algorithm whose execution time is proportional to 2^N takes 300 quintillion years. And that discrepancy gets much, much worse the larger N grows."[12] In other words, NP-hard problems are "hard" to solve computationally and cannot be solved in a reasonable amount of time unless one can show an *efficient*, polynomial-time algorithm for finding the solution. As such, alternative solutions to NP-hard problems which can be solved in polynomial-time are a constant point of research.

Ahmadi et al.[11] postulated that determining the convexity of a problem is NP-hard, but he also showed that sum-of-squares convexity can be determined efficiently [13], which is a viable substitute for determining convexity in many cases. This property specifically applies to problems where the polynomial exponents are small and there are few variables, "...less than 5 or 6." [13] Due to some of these difficulties, trajectory optimization has been largely done on the ground where significant computing power can be used to speed up the time

requirements of solving such problems. These techniques are often formulated as optimal control problems which take advantage of functionals using the calculus of variations. [14]

1.1.1 Optimal Control Problems in Aerospace

Optimal control theory is defined as a type of mathematical optimization problem that is used to find a control of a dynamical system such that a given objective functional is optimized. An example of this could be an aerospace vehicle, acting as the dynamical system, with a given set of controls and an objective to land on the moon using a minimum quantity of fuel. The optimal control set of differential equations have historically been derived by solving the Hamiltonian-Jacobi-Bellman equation thereby satisfying a sufficient condition or by using the Pontryagin's Minimum Principle satisfying a necessary condition of optimality. [15] Additionally, the optimal control problem may have non-unique solutions which are locally minimizing to the cost function. The general form of an optimal control problem is set up using the following formulation: [16]

Minimize the continuous-time, cost functional

$$J[x(\cdot), u(\cdot), t_0, t_f] := E[x(t_0), t_0, x(t_f), t_f] + \int_{t_0}^{t_f} F[x(t), u(t), t]dt \tag{1.1}$$

subject to the state equation

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

the path constraints

$$h[x(t), u(t), t] <= 0, \tag{1.3}$$

and the endpoint conditions

$$\mathrm{e}[x(t_0), t_0, x(t_f), t_f] = 0, \tag{1.4}$$

where $x(t)$ denotes the state, $u(t)$ is the control, t is the independent variable time, t_0 is the initial time, and t_f is the terminal time. E and F are the endpoint cost (Mayer term) and the running cost (Lagragian) respectively.

Often, optimal control problems are difficult to solve, especially as the problem starts to represent reality. Once an optimal control problem becomes too complex to solve analytically, numerical methods are used. One such method leverages the calculus of variations to obtain first order optimality conditions, then discretize the trajectory to solve a boundary value problem as it arises from taking the derivative of the Hamiltonian.[17], [18] This is known as the *indirect* approach to solving optimal control problems and solves problems quickly, but is hard to guarantee solution convergence. Conversely, the *direct* method discretises the trajectory first, then uses non-linear programming to solve for a local solution to the optimal control problem. This is also known as a pattern search and uses the Karush-Kuhn-Tucker (KKT) conditions to check optimality of a solution by first derivative tests (also called the first-order necessary conditions of optimality).[19]

In essence, this creates a mathematical programming problem which is very simple to program for any general trajectory optimization problem and allows for the application of ordinary calculus.[20] In trajectory optimization, a collocation problem is created when a two point boundary value problem with an infinite number of points between the endpoints is discretized into a finite number of points (nodes) between the endpoints.[21] The trajectory can then be optimized by moving the nodes to optimal locations using discrete optimization techniques as shown in Figure 1.2.

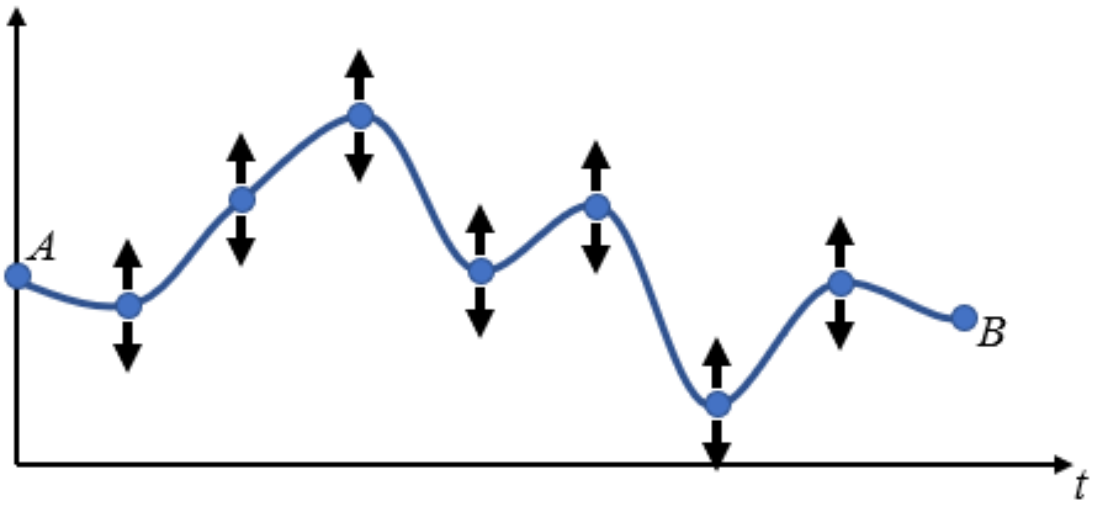


Figure 1.2. Direct Method Collocation Visualized.

However, optimality conditions cannot be reached by just moving the points around arbitrarily; they must be arranged such that the Equations of Motion (EOMs), $\dot{x} = f(x, u)$ are satisfied. The original approach to collocation uses cubic polynomials to represent state variables on each segment using values of the states and state time derivatives at the boundary nodes of each segment [21], but state of the art approaches also use other methods. [22]–[24] After discretizing the trajectory into nodes at time t_i, the state and time derivatives are constrained to match at the node intersections of the segments as shown in Figure 1.3. Figure 1.4 shows a combined visual of collocation and the discussed characteristics.

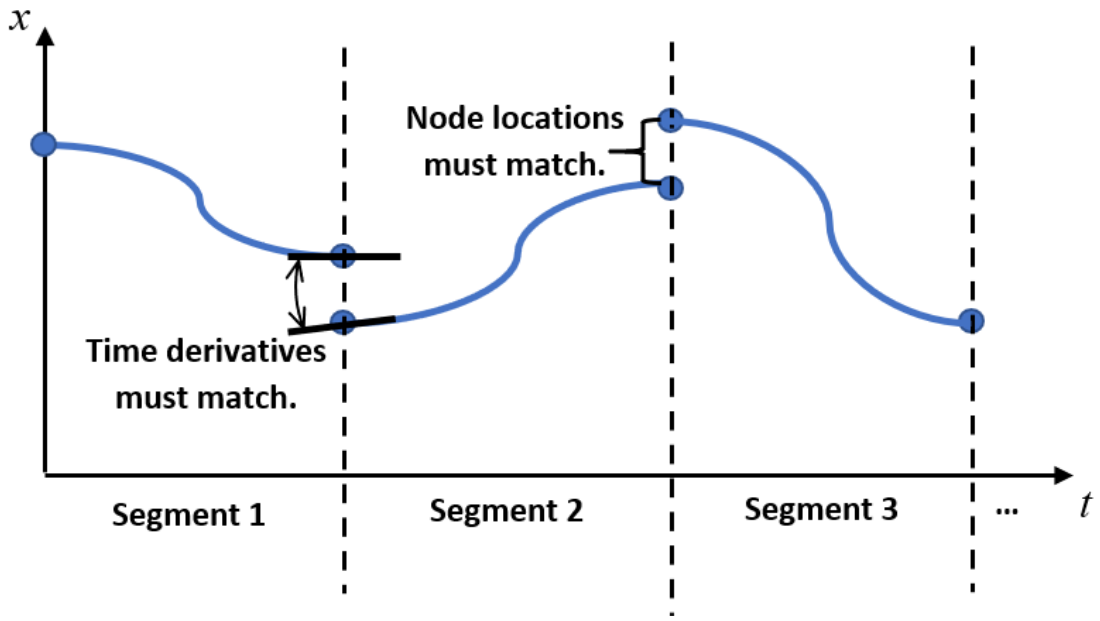


Figure 1.3. Collocation constraints.

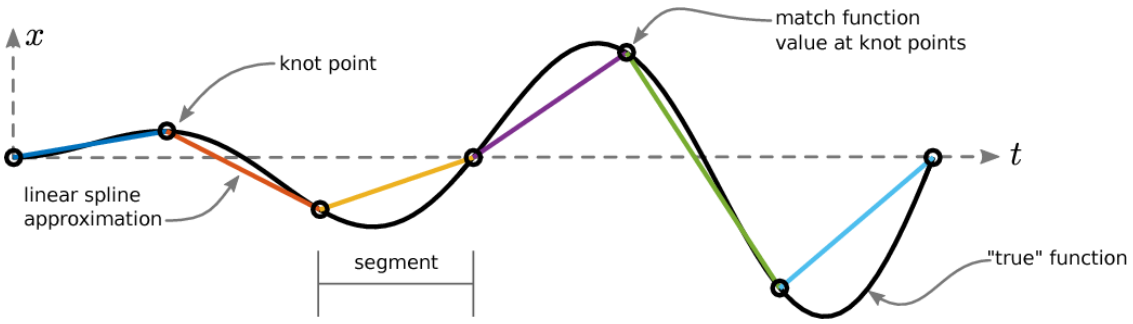


Figure 1.4. Generalized collocation diagram.[25]

The accuracy of direct collocation methods is tied to the tolerance with which the optimization of the nonlinear programming problem is set; however, these methods provide local optimal solutions and do not guarantee global optimality.

In addition to general *indirect* and *direct* methods, researchers have been investigating more specific methods by which to solve non-linear problems using alternative approaches to take advantage of the properties associated with particular problem types to overcome the NP-time requirements of direct methods and the need for a good initial guess to guarantee convergence for indirect methods. One such method uses convex optimization to expand upon the direct method and gain the benefits of convex solution spaces, one of which is solving for solutions in polynomial-time as discussed in Section 1.1. These methods are generally derived using the structure of collocation and altering the problem formulation such that it can be solved using a convex solver. In using collocation, trapezoidal methods were incorporated to keep the number of control points down and avoid introducing additional non-convex equations into the problem. This convexification process reformulates the non-linear, non-convex system of equations into a form which is naturally convex. Conic optimization, quadratic minimization with convex quadratic constraints, and semi-definite programming are a few examples of problem types which can be naturally convex, all of which can be solved using existing methods.[1], [26]–[28]

1.2 Aerospace Applications of Convexified Systems

The emergence of convex optimization in the field of aerospace has led to research into convex solutions to the powered descent and landing problem as well as drag-energy optimized entry guidance. One such approach was postulated by Açikmeşe et al. [26] for a method of lossless convexification of a soft landing optimal control problem. Specifically, the parts of the problem being convexified were the thrust pointing constraints which enabled a convex solution space for a soft landing optimal control problem.

Another popular point of study is the convexification of onboard guidance systems. This application of convex optimization is focused on speeding up already onboard systems such that the guidance can respond faster to flight data allowing for more rapid guidance re-

sponses or reducing computational load of onboard guidance systems. At AIAA SciTech 2018, Sagliano et al. presented a method for applying convex optimization to optimal drag-energy entry guidance. [28] In this work, Sagliano proposes introducing a new set of variables associated with the inverse of drag acceleration to achieve a loss-less convexified guidance to enable the possibility of solving the guidance algorithm in real-time.

Recently, researchers have also been attempting to apply the principles of convex optimization to hypersonic optimal control problems which are highly non-linear and non-convex by transforming the problem into a form which is naturally convex. Non-convex problems generally result in difficulty for a numerical solver to find global minima due to the search direction being defined by the gradient of the function of interest. Solving non-linear optimal control problems by direct methods, like collocation, is also considered NP-hard leading to long runtimes or high computational requirements. Both of these properties are solved by the nature of a convex solution space and using convex optimization to solve for solutions in polynomial-time as opposed to mathematical optimization which is generally considered NP-hard as defined previously in Section 1.1. One such method uses successive linear approximations to transform the non-linear, non-convex system of equations into second-order cone programming (SOCP) problems and solving them as a sequence of convex problems with each iteration resulting in a solution closer to the original problem of interest similar to sequential linear programming and sequential quadratic programming methods. [1]–[3] However, this method was unproven for angle of attack control trajectory optimization problems which have non-linear coupling between the control variable and the vehicle aerodynamics. This characteristic is a difficult obstacle for convexification due to the linearizations required for problem transformation to a convex form and will be explored in Section 3.2.

1.3 Convexification of a Non-Linear, Non-convex Optimal Control Problem

In order to use convex optimization techniques to solve highly non-linear, non-convex optimal control problems, the original problem must be reformulated into a form with a convex solution space through a process called "convexification." The base method used in

this thesis involves a series of successive linear approximations to linearize the non-linear EOMs about the fixed state history, $x_*(t)$, [1] as follows:

$$\dot{x} \approx f(x_*) + A(x_*)(x - x_*) + Bu \tag{1.5}$$

where

$$A(x_*) = \partial f(x)/\partial x|_{x=x_*} \tag{1.6}$$

and B represents the control vector which is constant and independent of the states. Similarly, non-convex objective functions of integral form can be linearized by a first-order Taylor serious expansion about the state history $x_*(t)$ as follows.

$$g(x) \approx g(x_*) + \partial g(x)/\partial x|_{x=x_*}(x - x_*) \tag{1.7}$$

This process results in a new problem with a convex objective function as well as convex state and control constraints. Non-linear path constraints can also be convexified, but these will not be of focus of this investigation. A trust region constraint is also enforced to ensure convergence of these successive linear approximations using a second order cone constraint, which is convex:

$$||x - x_*|| \preceq \delta \tag{1.8}$$

This trust region is difficult to define initially, but becomes more apparent after implementation and using engineering judgment; the trust region constraint can also be discovered by finding the required tolerance for the convex solution to represent the original problem of interest similar to convergence tolerance for methods like Sequential Linear Programming and Sequential Quadratic Programming. The combination of these methods seeks to transform the optimal control problem into a form with a linear objective function subject to second-order cone constraints, creating a SOCP problem which is naturally convex.

It should be noted however that a method to determine the success of convexification prior to implementation is entirely non-trivial and proving a convexified system is strictly convex is not guaranteed for complex systems. The approach most commonly implemented

to determine if a convexified system contains a convex solution space, is to attempt to solve the system as a convex problem.

1.4 Sequential Convex Programming

In order to solve the convexified problems described in Section 1.3, an iterative process is used which solves a sequence of convex optimal control problems each of which is initialized using the states from the solution to the previous iteration. Using this technique, an initial guess is provided to be used as dynamical state of the vehicle for the first optimization in the sequence. The convex optimal control problem is discretized into a number of nodes to convert the optimal control problem to one of finite dimensions. This involves discretizing the time domain into N equal intervals with constraints enforced at $N+1$ nodes. The resulting time intervals are defined by $\Delta t = (t_f - t_0)/N$ and nodes are denoted as follows, $t_0, t_1, t_2, ..., t_N$ with the state and control nodes, x and u, discretized along the same index similar to the collocation method. The dynamics can then be numerically integrated using the following formula for solution pair (x^k, u^k), where k is the trajectory number and i is the node number [1]:

$$\begin{aligned} x_{\mathrm{i}} = x_{\mathrm{i}-1} + \Delta t/2[&(A_{\mathrm{i}-1}^{k-1} x_{\mathrm{i}-1} + B u_{\mathrm{i}-1} + f_{\mathrm{i}-1}^{k-1} - A_{\mathrm{i}-1}^{k-1} x_{\mathrm{i}-1}^{k-1}) \\ &+(A_{\mathrm{i}}^{k-1} x_{\mathrm{i}} + B u_{\mathrm{i}} + f_{\mathrm{i}}^{k-1} - A_{\mathrm{i}}^{k-1} x_{\mathrm{i}}^{k-1})] \end{aligned} \tag{1.9}$$

Initial and terminal state constraints are defined as linear equality constraints while state and control bounds are enforced by second order cone inequality constraints. Linear equality constraint can also be expressed by two linear inequality constraints which is a special case of the second order cone constraint. This results in the discretized problem taking the form of an second order cone problem. Solving this discretized problem for a sufficiently large number of nodes results in an approximate numerical solution to the original optimal control problem.

At present, a complete proof of convergence for the sequential method is still unreached, but prior research in sequential methods can provide confidence of convergence using some aspects of the problem.[29]–[31] Convergence has also been studied in other research related

to sequential SOCP approximations [27], but a complete proof has not been found in this area of study yet either.

2. ONBOARD OPTIMIZATION ARCHITECTURE

Onboard trajectory optimization has become a point of research interest as the computational power which can be used for onboard applications has increased. While this advancement tends to lag behind commercial computation metrics, onboard capabilities continue to increase leading to interest in previously ground only systems being incorporated into onboard autonomous guidance, navigation, and control (GN&C) systems. Key factors in autonomous GN&C systems for all applications are a minimal program footprint and the ability to run real-time software on minimal computational hardware. These characteristics are important to autonomous systems due to their iterative nature and the requirements which are used to evaluate the effectiveness of potential onboard software. The hardware requirements for onboard systems are directly dependent upon the efficiency of the software being used. Reducing the data footprint of an onboard software can allow for additional data storage or create hard drive space for additional programs. Creating software which can run faster on a given set of hardware creates a situation where iterative processes can be run at a significantly increased rate and enables the use of realtime trajectory optimization with onboard GN&C systems. For the purposes of this thesis, all computational work was performed within the MATLAB R2015a programming environment due to availability and runtime results are considered to be comparative. [32]

2.1 Research Architecture

The implementation of sequential convex optimization postulated by Z. Wang [1] uses an architecture made up of a simulation, an equation modeler and a solver. The simulation is used to create an initial trajectory which satisfies the EOMs associated with the optimal control problem. This creates an initial guess to the optimal control problem which satisfies the dynamic constraints as each node location in the discretized trajectory. This initial guess trajectory is used as the starting point for the first iteration of the sequential method and is used by the equation modeler to populate the convexified equations before passing them to the solver during each iteration of the sequential method. The equation modeler is a computational program which allows a user to define an optimization problem using

symbolic mathematical expressions which are then interpreted by the modeler, reformatted into a solver specific form, and then fed to the solver for optimization. The solver takes the output from the modeler and finds an optimal solution to the defined objective function, given the constraints, state equations, and options output by the modeling program. This architecture is extremely effective for research purposes because it takes the computational formatting out of the hands of the engineer and allows them to work from a symbolic equation viewpoint, which is much quicker to experiment with. However, this architecture creates slowdowns at several run-time points and uses a large amount of data compared to a dedicated architecture. This has led to all onboard software being purpose coded for maximum speed and minimal program data footprint in an effort to reduce computational complexity, increase programmatic effectiveness, and reduce cost.

2.1.1 Simulation, Modeler, and Solver

The simulation used to generate the initial guess trajectory uses the dimensionless atmospheric relative EOMs for an unpowered entry vehicle with three degrees of freedom about a spherical, non-rotating planet to model the motion a hypersonic vehicle from initial conditions to a given terminal state. [33] Numerical integration was used to propagate the trajectory forward in time from the initial conditions to the defined terminal state. This trajectory was then discretized into the equally spaced nodes at t_n to create the initial guess for the convexified trajectory optimization problem associated with the first iteration of the sequential method In Z. Wang's implementation of sequential convex programming, a mathematical modeler is used to rewrite the convexified EOMs in a form that a given solver can understand. For sequential numeric problems, the modeler must be rerun during each iteration of the sequential process to convert the numerical problem data into a form recognized by the convex solver. For the work presented in this document, the interfacing program YALMIP [34] (not an acronym) was used as the symbolic equation modeler. YALMIP is a multi-functional optimization interfacing tool which facilitates modeling and interfacing with the solver to reduce the human work required to solve optimization problems. YALMIP is able to drastically reduce the time required to set up an optimization problem for a par-

ticular solver by allowing the user to define their problem by writing algebraic equations. It then translates the algebraic form of the problem to a format which is used by a particular solver. While this is very helpful when solving a big optimization problem with a low time investment upfront from the engineer, an equation modeler adds too much computational overhead to be useful when used in a sequential method or for onboard purposes which need to have as fast of a runtime as possible to enable autonomous applications. Additionally, YALMIP is currently only supported as a MATLAB program, which would not be an acceptable coding language for autonomous onboard software applications.

In the research level architecture of the sequential convex method, the SeDuMi solver [35] is used to perform the SOCP optimization for each sequential iteration. SeDuMi is a program used to perform optimization over symmetric cones using convex optimization, linear inequality matrices, second-order cone constraints, and linear equations. It is a very effective software package, but it has not been optimized to minimize central-processing-unit (CPU) time and is better suited to experimental applications where the nature of the solver needs to be understood and perhaps even modified by an engineer. A solver suited for onboard aerospace applications is expected to be software optimized to reduce CPU time and does not need to be read by an engineer, but rather run continuously by a computer.

2.2 Onboard Architecture

Initial testing revealed that modeling the problem symbolically using YALMIP is responsible for a sizable portion of the run time for a sequential SOCP problem due to multiple calls of the YALMIP modeler and is not conducive to a fast, iterative approach. In the sequential approach, YALMIP is called at each point in the iterative process as it is needed to assign values to variables and convert the symbolic convexified equations of the dynamics, constraints, and objective into numerical form and then pass them to the solver. For any onboard applications, a solver is specified and the general aspects of the problem being solved are well defined and understood before any software is incorporated into a vehicle. As such, solver agnostic symbolic modeling of the convexified equations of motions are not necessary. Following this logic, replacing YALMIP with a faster and more direct modeling method

was hypothesized to provide a clear improvement for onboard applications of optimizing the convexified flight EOMs.

It is often required that flight software use as little resources as possible in order to conserve both processing power and data storage. While SeDuMi is a very useful solver, it has a large data storage footprint and it is common for a solver to use significant processing power due to the shear number of iterations taking place within the solver. It was found that SeDuMi was not very processor efficient, though not horrible, and replacing it with a more purpose built solver may provide an additional point of improvement when attempting to reduce the onboard software data footprint and, at minimum, not slow down runtime compared to SeDuMi. In addition to these two criteria, another constraint on solvers for onboard applications is being either open-source or developed in-house. While in-house development is usually the most optimal because it can be designed for exactly the purpose it is being proposed for, time and cost of development are a significant driver that lead many to use open-source software. Since development of an SOCP solver is not the objective of this thesis, several off-the-shelf solvers were investigated for potential use in onboard applications. Initial research showed that Gurobi, ECOS, MOSEK, and SDPT3 were available solvers with interfaces in YALMIP, and FORCES was an additional solver which was thought to be of use once the YALMIP modeler was removed. Each of the listed solvers has the capability built in to solve naturally convex SOCP problems. For the purposes of this thesis, licensed solvers were eliminated to avoid the need for a specific license to replicate and use the methods discussed therein. Gurobi, Mosek, and FORCES were all removed from the trade space for this reason, but these are all proven solvers with proven performance that should be considered if license is not an issue. SDPT3 and ECOS were both tested against SeDuMi using simple example problems provided for each solver using YALMIP as the consistent modeler. SDPT3 was unable to produce in-family results with SeDuMi while ECOS was able to achieve similar results within an average of 0.001% of the solutions observed in SeDuMi. ECOS is defined as a "lightweight conic solver for second-order cone-programming" which was developed for the express purpose of solving second-order cone-programming problems with zero adjunct functionality. Additionally, ECOS is advertised as an open-source, embedded solver designed using only 750 lines of code and produces virtually no variation in run

time.[36] ECOS is available in several languages including MATLAB, C, and C++. As such, it fits the criteria for onboard applications better than other solvers or solver packages like Gurobi which are very easy to use, but come loaded down with a plethora of options and extraneous functionality as well as the need for a license.

2.2.1 Upgrading the Solver

Replacing the SeDuMi solver with ECOS was exceptionally simple while YALMIP was still being used for problem setup; YALMIP includes ECOS as a solver option. As YALMIP advertised, it took only changing some options in the call and pointing to the ECOS solver location in order to swap out solvers. With this simple change, the solver was successfully switched to ECOS, and the results of preliminary testing showed optimization results in family with those achieved with SeDuMi which will be demonstrated in Section 3.1. One advantage to ECOS which was not considered originally, but became evident when removing the YALMIP modeler was the ability for ECOS to handle sparse constraint matrices. This will be discussed further in the following subsection.

2.2.2 Removing Equation Modeler: Direct Population Method

Unlike the algebraic equation inputs to the YALMIP modeler, the ECOS solver requires a SOCP to be set up in standard optimization form,

$$\begin{aligned} &minimize\ c^T x \\ &subject\ to\ Ax = b \\ &\qquad Gx \preceq h\ or\ Gx + s = h, \quad s \in\ K, \end{aligned} \tag{2.1}$$

where x is a vector of the primal variables, s is a vector of slack variables, $c \in \mathbf{R}^n$, $A \in \mathbf{R}^{(p \times n)}$, $b \in \mathbf{R}^p$, $G \in \mathbf{R}^{(M \times n)}$, $h \in \mathbf{R}^M$ are the problem data definitions and K denotes the cone.[36] In this format, the initial and terminal state constraints and the dynamic state constraints are used to create $Ax = b$ while the SOCP state bounds, control bounds, inequality path constraints, and trust regions constraints are used to create $Gx \preceq h$. Recreating the opti-

mization problem in this form requires converting the algebraic equations to matrix optimization form. A and G are matrices made up of sub-matrices which correspond to vectors of sub-vectors b and h respectively.

$$A_{(p,n)} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{p,1} & a_{p,2} & \cdots & a_{p,n} \end{bmatrix}, \; b_{(p)} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{bmatrix} \tag{2.2}$$

$$G_{(m,n)} = \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,n} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{m,n} & g_{m,n} & \cdots & g_{m,n} \end{bmatrix}, \; h_{(m)} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_m \end{bmatrix} \tag{2.3}$$

where $a_{p,n}$ and $g_{m,n}$ are sub-matrices of problem data aligned with primal variables x_n of node n and constraint sub-vectors b_p and h_m of constraint numbers p and m respectively. In order to solve a trajectory problem using this type of solver input, a collocation method was used to apply constraints to the dynamics between the nodes. The process of forming the convex dynamical constraints will be discussed in greater detail in Sections 3.1 and 3.2, but an important factor for modeling these constraints in a discretized fashion is that each node in the trajectory has motion constraints only with the adjacent node on either side of it. This suggests that the equality constraint matrix for a trajectory problem will be largely populated by zero matrices, problem data for each node, n, problem data at the following adjacent node, $n+1$, and equality problem constraints such as initial and final conditions in Eq. 2.4. Additionally, as the number of nodes increases, the number of zero sub-matrices increases at a much faster rate than the number of non-zero sub-matrices. In this implementation, the number of problem data sub-matrices corresponding to trajectory

dynamics at the nodes within A is equal to $2(n-1)$ while the number of zero matrices within A is equal to $(n-2)*(n-1)$.

$$A_{dyn} = \begin{bmatrix} a_{1,1} & a_{1,2} & 0 & \cdots & 0 \\ 0 & a_{1,2} & a_{1,3} & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & a_{p-1,n-2} & a_{p-1,n-1} & 0 \\ 0 & \cdots & 0 & a_{p,n-1} & a_{p,n} \end{bmatrix} \tag{2.4}$$

Looking at the growth of the A constraint matrix in this way, it is evident that the number of zero matrices introduced as the number of nodes increases outpaces the number of problem data matrices.

When a matrix form of this nature emerges in optimization problems, the matrix is known as *sparse* and it is advantageous to be able to model that matrix using triplet representation. [37], [38] Triplet representation is created by noting the position and value of each non-zero number in a matrix and all other positions are assumed to be zero and do not need to be stored in memory. This store data as an $m \times 3$ matrix instead of a $p \times n$ matrix where the three columns of the triplet denote the rows, columns, and values of non-zero entries in the original matrix. Figure 2.1 shows equivalent sparse matrices, one in standard matrix representation and the other in triplet representation.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \end{bmatrix} = \begin{array}{c} \text{Row, Column, Value} \\ \left[\begin{array}{c:c:c} 0 & 0 & 1 \\ 1 & 4 & 5 \\ 2 & 2 & 3 \\ 3 & 1 & 2 \\ 4 & 3 & 4 \end{array}\right] \end{array}$$

Figure 2.1. Equivalent *standard(left)* and *triplet(right)* matrix representation.

Using triplet representation to model sparse matrices is expected to remove the memory allocation of the zero elements of a standard represented matrix and programs which can interact directly with triplet formatted matrices are generally faster at performing operations using those matrices. Efficient use of triplet representation requires a matrix to be sparse and a matrix is considered sparse when its sparcity (1 - the density of the matrix) is greater than 0.5. [39] In the case of the dynamic constraints only (excluding other equality constraints) the sparcity of the A matrix is equal to $1-2/n$ and is considered to be sparse for all problems with $n > 4$ by the sparcity function above. Due to the sparcity of the A matrix, only two sub-matrices, $a_{\mathrm{i,j}}$ are added to the structure when an additional node is introduced instead of $2(n-1)$ sub-matrices where $2(n-2)$ of the additional sub-matrices are zero matrices. Adding more equality constraints can affect the sparcity of the A matrix, but for a trajectory style problem where the number of nodes is likely greater than 50, the additional constraints are unlikely to break sparcity. For the purposes of this thesis, the only additional equality constraints correspond to the initial conditions and the fixed terminal conditions. For an A matrix which is used to represent 300 nodes, the resulting sparcity would be 0.993 suggesting that a 300 node collocated trajectory problem could be represented in triplet representation much more efficiently than standard notation. As a check of this high sparcity value, a 300 node problem with seven states and one control ($a_{\mathrm{i,j}}$+control is 7×8) was used to generate an A matrix and resulted in 33,544 non-zero elements and 4,998,056 zero elements when assuming all values of $a_{\mathrm{i,j}}$ are non-zero. The problem used for this example is the bank angle control problem detailed in Section 3.1. Representing the non-zero elements in triplet notation allows for a 97.986% reduction in the element volume of the A matrix allocated in memory.

The G matrix is also observed to be sparse and can be formatted using triplet representation as well. G uses +1, -1, and 0 value elements to correlate states and nodes to the inequality constraint values in vector h, and a significant number of these are state and control constraints which act on one node and one state at a time. Similar to the A matrix, this causes the sparsity of G to increase as the number of nodes increases continuing to make the process more efficient by using triplet representation than standard matrix form. For the same 300 node problem, G was found to have only 8,986 non-zero elements and 21,591,014

zero elements with triplet notation resulting in a 99.875% reduction in the number of elements required to fully represent G in triplet format. Formatting the constraint matrices in triplet representation and showing that the sparcity of these matrices increases as the number of nodes increases, it can be inferred that as the number of nodes is increased to improve the accuracy of the collocated solution, this modeling technique continues to increase in efficiency in comparison to standard matrix representation methods.

ECOS is equipped to handle sparse matrices in triplet representation without converting them back to standard matrices making operations very efficient; significantly decreasing runtime per iteration and reducing memory storage. Equation modelers, like YALMIP, are generally not equipped to populate sparse matrices using triplet representation unless built specifically to do so. As such, the combined architecture of direct population using triplet representation and ECOS as the embedded solver is expected to provide speed ups in runtime as well as decrease data footprint for an onboard trajectory optimization system. Verification of this hypothesis can be performed by doing a direct comparison to the research architecture previously used for the sequential convex optimization method, YALMIP and SeDuMi. Once verified, new control schemes that have use in onboard applications can also be investigated using this architecture.

3. VERIFICATION, VALIDATION, and APPLICATION

When taking a concept or method of solving a problem in a particular way and changing the structure in which that concept or method operates, it is of the utmost importance to verify that the solution obtained does not change outside of the hypothesized parameters. When changing the solver and architecture from one originally only meant for lab demonstration to one which is expected to show the potential for onboard applications, verifying that the new method solves the same problem as the previous method within a given specification is an essential part of the engineering design process. For the application of real-time onboard trajectory optimization, speed and a small data footprint are of paramount importance. As such, this implementation of the convexified, hypersonic EOMs for planetary entry over a spherical planet was investigated with the intent to both decrease runtime of the optimal control problem for hypersonic atmospheric flight and the data footprint associated with the optimization architecture.

In order to verify, validate, and demonstrate the capabilities involved with the changes made in Chapter 2, two hypersonic flight problems were chosen. The first of which is a bank angle control, hypersonic entry starting at the edge of the atmosphere with the optimal control problem of minimizing the impact velocity of a given vehicle. As a well studied scenario, this problem will provide a testing ground for verifying that the updated architecture is performing as specified and providing a platform for testing how well the updated method achieves the original hypothesis to speed up optimization time and reduce total software data footprint for an onboard trajectory optimization system. An angle of attack control, maximum impact velocity at impact problem was selected in order to demonstrate the application of the convexified hypersonic equations to an additional control scheme and determine the steps required when the linearity of the problem is altered by the choice of control.

3.1 Bank Angle Control: Hypersonic Re-entry

Bank angle control is a method of flight control which involves rotating the lift vector of a vehicle in order to control downrange and crossrange error during flight. [40] This flight

control is characterized by a vehicle with a non-zero trim angle of attack, a non-zero lift-to-drag ratio (L/D), and a physical control which rotates the vehicle, thereby rotating the lift vector and providing control orientation of the aerodynamic lifting force vector as shown in Figure 3.1.

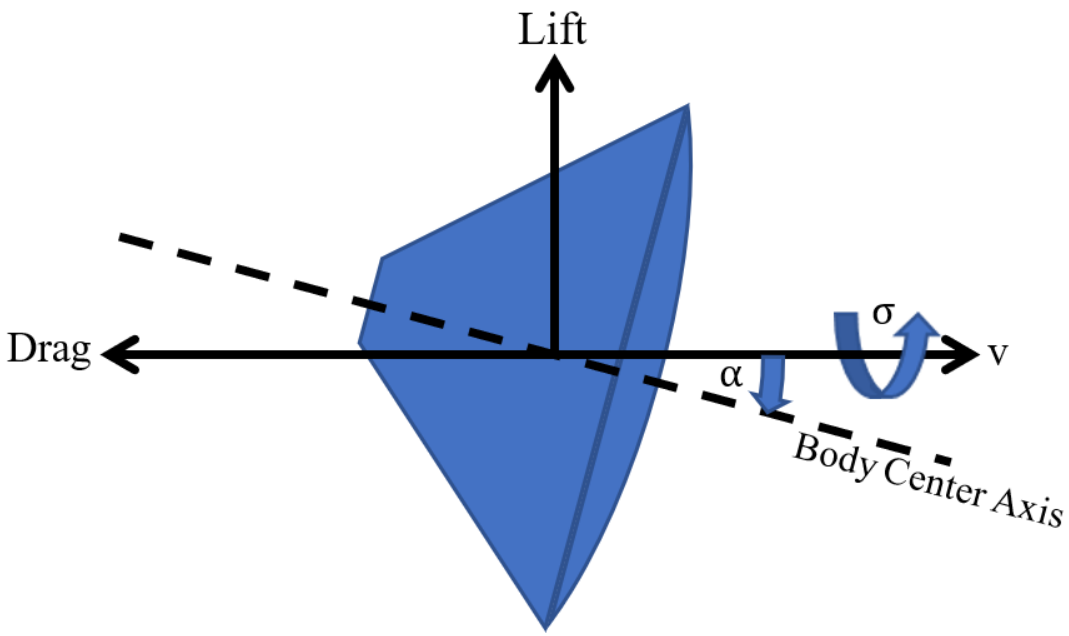


Figure 3.1. Aerodynamic Force and Control Angle Diagram of a Capsule-like Vehicle.

A major benefit of using bank angle as the control mechanism is that it requires only a single physical control in order to manage both downrange and crossrange errors. While this can make the design of the vehicle simpler, it requires an increase in guidance complexity. With bank angle control, the out of plane and vertical components of the aerodynamic force vector are coupled in the guidance which leads to increased complexity when solving for optimal trajectories. Bank angle guidance generally results in a trajectory defined by S-curves where the lift vector is cyclically rotated left and right from vertical to control the crossrange error. The guidance algorithm is expected to manage this rotation of the lift vector to also control the downrange error leading to two errors coupled to a single control with constraints on the control based on physical limitations of the vehicle's physical control mechanism.

The main benefit of bank angle control is the decreased mechanical complexity in vehicle design required to initiate a bank angle control maneuver. Bank angle control is often used in applications where the time of flight is long to allow for correction of out of plane errors over time or for vehicles where decreasing mechanical complexity of the control system is required. One such application which satisfies both of these types of trajectories is a hypersonic entry problem for a capsule-like vehicle. Controlled entry trajectory problems have been assessed using bank angle control many times in the past and and bank angle control continues to be a staple in re-entry trajectory simulations where complexity of the entry vehicle is highly constrained. Bank angle control is also a common choice for guided entry of a blunt body vehicle due to the necessity for low complexity control mechanisms onboard.[41] Bank angle was the control chosen in [1], the solution method of which will provide a baseline for verification of the direct population and ECOS combination using the same convexification approach to the hypersonic equations of motion.

3.1.1 Bank Angle Controlled Solution with Direct Population and ECOS Solver

In order to verify the changes to the problem formulation process discussed in Chapter 2, a bank angle control entry problem was used. This problem was defined as a controlled re-entry problem with the following characteristics for a Reusable Launch Vehicle (RLV) with a mass of 104305 kg and a reference area of 391.22 m^2; and is subject to the constraints defined in Tables 3.1-3.2. The initial conditions and terminal constraints defined in Table 3.1 are added to the equality matrix A and equality vector b while the inequality constraints in Table 3.2 are included in the inequality matrix G and inequality vector h.

In addition to the parameters defined above, an angle of attack profile has been specified for this entry problem:

$$\alpha = \begin{cases} 40, \text{if } V > 4570 \text{ m/s} \\ 40 - 0.20705 * ((V - 4570)/340)^2, \text{if } V \leq 4570 \text{ m/s} \end{cases} \tag{3.1}$$

Table 3.1.
Initial Conditions and Terminal Constraints

Variable	**Initial**	**Target**
Altitude	100 km	25 km
Velocity	7.45 km/s	Unconstrained
Longitude	0 deg	12 deg
Latitude	0 deg	70 deg
Flight Path Angle	-0.5 deg	-10 deg
Heading Angle	0 deg	90 deg
Bank Angle	0 deg	Unconstrained
Bank Angle rate	0 deg/s	Unconstrained

Table 3.2.
State and Control Constraints

Variable	**Minimum**	**Maximum**
Altitude	25 km	100 km
Velocity	10 m/s	7.45 km/s
Longitude	0 deg	180 deg
Latitude	0 deg	90 deg
Flight Path Angle	-89 deg	89 deg
Heading Angle	0 deg	180 deg
Bank Angle	-80 deg	80 deg
Bank Angle rate	Unconstrained	10 deg/s

where α is defined in degrees with respect to the velocity vector. For this vehicle and the angle of attack profile defined above, the lift and drag coefficients are defined as:

$$C_L = -0.041065 + 0.016292 * \alpha + 0.00026024 * \alpha^2 \tag{3.2}$$

$$C_D = 0.080505 - 0.03026 * C_L + 0.86495 * C_L^2 \tag{3.3}$$

To create a proper verification comparison, Z. Wang's method [1] and the direct population method, detailed in Chapter 2, were each used to set up an optimal control problem where the control is bank angle rate, $\dot{\sigma} = u$:

$$\underset{x,u}{\text{Min}}\, J = \varphi[x(t_f)] + \int_{t_0}^{t_f} g(x)dt \tag{3.4}$$

$$\text{Subject to} : \dot{x} = f(x) + Bu \tag{3.5}$$

$$x(t_0) = x_0,\ x(t_f) = x_f \tag{3.6}$$

$$x \in [x_{min},\ x_{max}] \tag{3.7}$$

$$|u| \leq u_{max} \tag{3.8}$$

where x_0 and x_f are the initial and terminal constraints respectively, x_{min} and x_{max} are the state variable bounds, and u_{max} is the maximum control bound of bank angle rate, $\dot{\sigma}_{max}$. $x = [r, \theta, \phi, V, \gamma, \psi]$ is the state vector defined by the dimensionless atmospheric-relative EOMs for an unpowered entry vehicle with three degrees of freedom about a spherical, non-rotating planet to model the motion a hypersonic vehicle from initial conditions to a given terminal state:

$$\dot{r} = V \sin\gamma \tag{3.9}$$

$$\dot{\theta} = V \cos\gamma \sin\psi/(r\cos\phi) \tag{3.10}$$

$$\dot{\phi} = V \cos\gamma \cos\psi/r \tag{3.11}$$

$$\dot{V} = -D - \sin\gamma/r^2 \tag{3.12}$$

$$\dot{\gamma} = L\cos\sigma/V + (V^2 - 1/r)\cos\gamma/(Vr) \tag{3.13}$$

$$\dot{\psi} = L\sin\sigma/V\cos\gamma + V\cos\gamma\sin\psi\tan\phi/r \tag{3.14}$$

where r is the distance from the planet's center to the center of the vehicle which has been normalized by the radius of the planet, R_0, and V is the planet relative velocity normalized by $\sqrt{R_0 g_0}$. θ and ϕ are the longitude and latitude respectively while γ and ψ are the flight path angle and the heading angle respectively; ψ is measured clockwise from North in the local horizontal plane. Additional variables σ, L and D are the bank angle, lift acceleration

and drag acceleration. The lift and drag acceleration have been normalized with respect to g_0 to complete the dimensionless EOMs.

$$L = R_0 \rho V^2 A_{ref} C_L / (2m) \tag{3.15}$$

$$D = R_0 \rho V^2 A_{ref} C_D / (2m) \tag{3.16}$$

where A_{ref} and m are the dimensional reference area and mass of the vehicle respectively. ρ is the atmospheric density and for the purposes of this simulation, density is calculated using an exponential model, $\rho = \rho_0 \mathrm{e}^{-h/H_s}$, where H_s is the scale height, h is the altitude of the vehicle and ρ_0 is the sea-level surface density. C_L and C_D are the aerodynamics coefficients of lift and drag respectively and are a function of angle of attack, α, for this simulation.

Defining the control as the bank angle rate and creating bounds on this rate helps to reduce the high frequency behavior and decouple the control from the state vector. In this formulation, the bank angle rate control, u, is defined as an additional zero state in $f(x)$ and is normalized by $\sqrt{R_0/g_0}$. Using bank angle rate, the control can then be decoupled from the states using the constant control matrix B:

$$f(x) = \begin{bmatrix} (3.9) & (3.10) & (3.11) & (3.12) & (3.13) & (3.14) & 0 \end{bmatrix}^T \tag{3.17}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T \tag{3.18}$$

Additionally, the optimization problem can be simplified under the following assumptions:

- The atmosphere rotates with the Earth.
- This vehicle is not subject to constraints on heat rate, $\dot{Q}$, dynamic pressure, q, and normal load, n.
- The objective function J is defined to minimize the terminal velocity of the vehicle, $V(t_f) = V_f$ at an altitude of 25km.

These assumptions result in the simplified optimization problem:

$$\underset{x,u}{\text{Min}}\ J = V(t_f) = V_f \tag{3.19}$$

$$\text{Subject to}: \dot{x} = f(x) + Bu \tag{3.20}$$

$$x(t_0) = x_0,\ x(t_f) = x_f \tag{3.21}$$

$$x \in [x_{min},\ x_{max}] \tag{3.22}$$

$$|u| \leq u_{max} \tag{3.23}$$

With this formulation of the optimization problem where the control is decoupled from the state vector, the non-linear term in the dynamic constraint, $f(x)$, can be linearized using the fixed time history, $x_*(t)$:

$$\dot{x} = f(x_*) + a(x_*)(x - x_*) + Bu \tag{3.24}$$

where

$$a(x_*) = \left.\frac{\partial f(x)}{\partial x}\right|_{x=x_*} = \begin{bmatrix} \frac{\partial \dot{r}}{\partial r} & \frac{\partial \dot{r}}{\partial \theta} & \frac{\partial \dot{r}}{\partial \phi} & \frac{\partial \dot{r}}{\partial V} & \frac{\partial \dot{r}}{\partial \gamma} & \frac{\partial \dot{r}}{\partial \psi} & \frac{\partial \dot{r}}{\partial \sigma} \\ \frac{\partial \dot{\theta}}{\partial r} & \frac{\partial \dot{\theta}}{\partial \theta} & \frac{\partial \dot{\theta}}{\partial \phi} & \frac{\partial \dot{\theta}}{\partial V} & \frac{\partial \dot{\theta}}{\partial \gamma} & \frac{\partial \dot{\theta}}{\partial \psi} & \frac{\partial \dot{\theta}}{\partial \sigma} \\ \frac{\partial \dot{\phi}}{\partial r} & \frac{\partial \dot{\phi}}{\partial \theta} & \frac{\partial \dot{\phi}}{\partial \phi} & \frac{\partial \dot{\phi}}{\partial V} & \frac{\partial \dot{\phi}}{\partial \gamma} & \frac{\partial \dot{\phi}}{\partial \psi} & \frac{\partial \dot{\phi}}{\partial \sigma} \\ \frac{\partial \dot{V}}{\partial r} & \frac{\partial \dot{V}}{\partial \theta} & \frac{\partial \dot{V}}{\partial \phi} & \frac{\partial \dot{V}}{\partial V} & \frac{\partial \dot{V}}{\partial \gamma} & \frac{\partial \dot{V}}{\partial \psi} & \frac{\partial \dot{V}}{\partial \sigma} \\ \frac{\partial \dot{\gamma}}{\partial r} & \frac{\partial \dot{\gamma}}{\partial \theta} & \frac{\partial \dot{\gamma}}{\partial \phi} & \frac{\partial \dot{\gamma}}{\partial V} & \frac{\partial \dot{\gamma}}{\partial \gamma} & \frac{\partial \dot{\gamma}}{\partial \psi} & \frac{\partial \dot{\gamma}}{\partial \sigma} \\ \frac{\partial \dot{\psi}}{\partial r} & \frac{\partial \dot{\psi}}{\partial \theta} & \frac{\partial \dot{\psi}}{\partial \phi} & \frac{\partial \dot{\psi}}{\partial V} & \frac{\partial \dot{\psi}}{\partial \gamma} & \frac{\partial \dot{\psi}}{\partial \psi} & \frac{\partial \dot{\psi}}{\partial \sigma} \\ \frac{\partial \dot{\sigma}}{\partial r} & \frac{\partial \dot{\sigma}}{\partial \theta} & \frac{\partial \dot{\sigma}}{\partial \phi} & \frac{\partial \dot{\sigma}}{\partial V} & \frac{\partial \dot{\sigma}}{\partial \gamma} & \frac{\partial \dot{\sigma}}{\partial \psi} & \frac{\partial \dot{\sigma}}{\partial \sigma} \end{bmatrix} \tag{3.25}$$

where a is the sub-matrix populated with problem data at each node as mentioned in Chapter 2. With this formulation, the original, highly non-linear optimal control problem has been converted to a continuous-time optimal control problem with a terminal point objective function, convex state constraints, convex control constraints, and a control variable

decoupled from the state variables. In addition, a trust region constraint, δ, has been added in order to ensure convergence of successive linear approximations in Eq. 3.27.

$$\underset{x,u}{\text{Min}}\ J = V_f \tag{3.26}$$

$$\text{Subject to} : \dot{x} = f(x_*) + a(x_*) * (x - x_*) + Bu \tag{3.27}$$

$$x(t_0) = x_0,\ x(t_f) = x_f \tag{3.28}$$

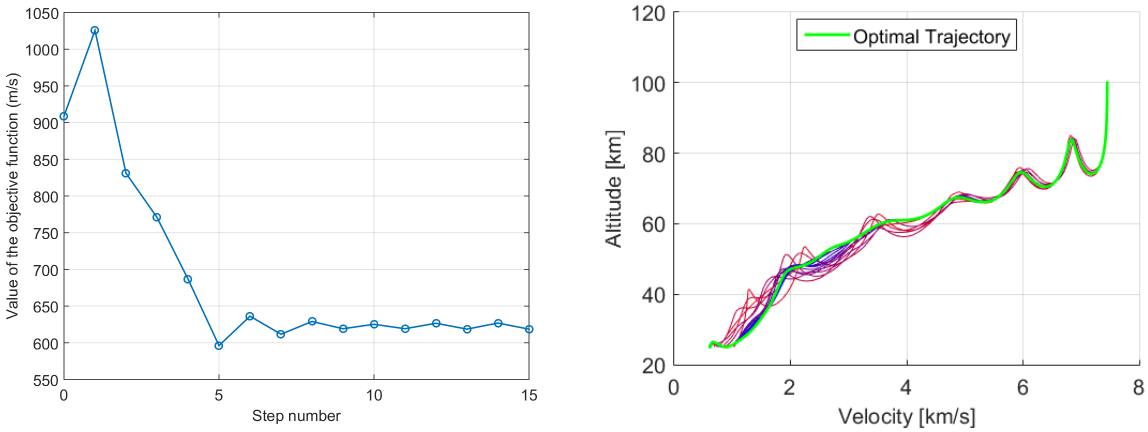
$$x \in [x_{min},\ x_{max}] \tag{3.29}$$

$$|u| \leq u_{max}\|x - x_*\| \leq \delta \tag{3.30}$$

where the trust region constraint Eq. 3.30 is defined as a second-order cone constraint. From here, the optimal control problem can be formulated into the discretized format where the A matrix is populated by sub-matrices, $a_{\text{i,j}}$. In this application, the rows, i, determine the dynamic constraint grouping number between two nodes and the columns,j determine the node numbers associated with each constraint. The initial conditions and the terminal state equality constraints are then appended to the A matrix. The b vector is created to store the right side of the equality constraint while A and x create the left side of the equality constraint. Using the standard optimization form, Eq. 2.1, all equality constraints in the problem can be represented using the A matrix and b vector. Conversely, G holds the inequality constraints of the bank angle control problem where in G, much like A, the columns of G represent the state groupings for each node. In addition, h is populated by the state variable bounds, control bounds, and trust region constraints for all nodes where G determines for which variable in x the constraint is active and at which node. In effect, this formulation sets up individual equality and inequality constraints as a collocation problem where the sub-matrix rows of A and G represent the equality and inequality constraints, respectively, in order to pass them to the ECOS solver in standard optimization form.

From here the problem can be transformed, as in Chapter 2, into a set of sparse matrices of the form depicted in Eq. 2.4 and the associated state and constraint vectors. This set of matrices and vectors is then passed into ECOS several times during the sequential process, moving towards an optimal solution to this convex problem. Figure 3.2 shows the

solution steps towards an optimal solution using sequential convex programming. One thing to note is that not all of the sequential solution steps satisfy the problem constraints or trust-region constraints, but each successive iteration marches towards the final solution by decreasing the objective function, satisfying constraints, and converging under the trust region constraint. These intermediate solutions generated during the sequential process can have a lower objective function value, but do not trigger convergence due to violated constraints.



(a) Sequential Convex Obj. Function (b) Sequential Trajectory Solutions.

Figure 3.2. Steps towards Optimal Trajectory Solution.

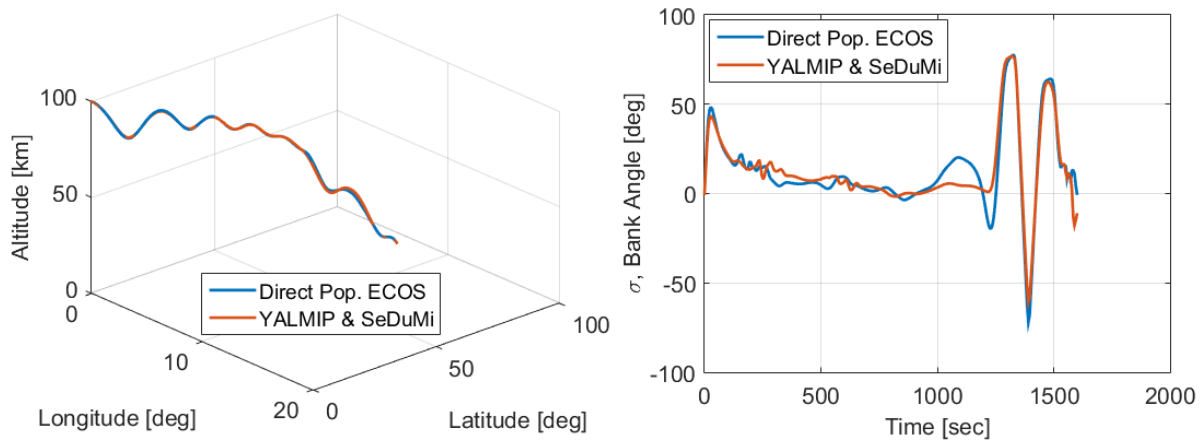
Since this sequential method uses collocation to solve for optimal trajectories, an initial guess is required for the optimizer to start with. While this can simply be an uncontrolled trajectory generated using numerical integration, a better initial guess can significantly reduce the number of iterations before an optimal solution is solved. In addition to generally fast run times, this is a key characteristic for any onboard trajectory optimization tool which is expected to re-run regularly with small variations to parameters to account for environment changes or alterations in terminal point constraints. So using the previously optimized trajectory as the new initial guess for the next recalculated optimal path has the potential to decrease solving time dramatically by reducing the number of iteration required to reach solution convergence; this behavior will be explored in Section 3.2.4.

In order to verify that the direct population method using sparse matrices is working as intended while interfaced with ECOS, the solution can be directly compared to the solution of the same problem using the previous architecture of YALMIP and SeDuMi.

The comparative trajectories are shown in Figures 3.3-3.5 and the progressive runtime decrease is shown in Table 3.3. In general, these trajectories follow the same trend even with some variance in the resulting bank angle control profile. Both solutions satisfy the terminal constraints and all state limit constraints, but are likely in different local minima of the solution space. The differences observed could be due to a variety of reasons, but is most likely the result of differences in the convex optimization method used in ECOS compared to SeDuMi or equation construction differences between the YALMIP modeler and the direct population method since YALMIP has to work with a large number of problem and solver types. The resulting optimized control is fairly smooth due to the updated control from bank angle to bank angle rate. This is especially true over bank reversal zones, which are often jittery with traditional trajectory control optimization. The jitters observed could be reduced further by decreasing the maximum bank angle rate limit. Table 3.3 shows the per iteration run-time for both setups solving the same bank angle control entry problem. This table shows that the ECOS solver was able to solve iteration steps in the sequential programming method 2.33x faster than the SeDuMi solver for this problem and the direct population method proved to be 6.2x faster than the YALMIP equation modeling program. An additional benefit, which was not explored, is that the direct population method is expected to decrease run-time even further when incorporated into a faster onboard programming language, like C or C++.

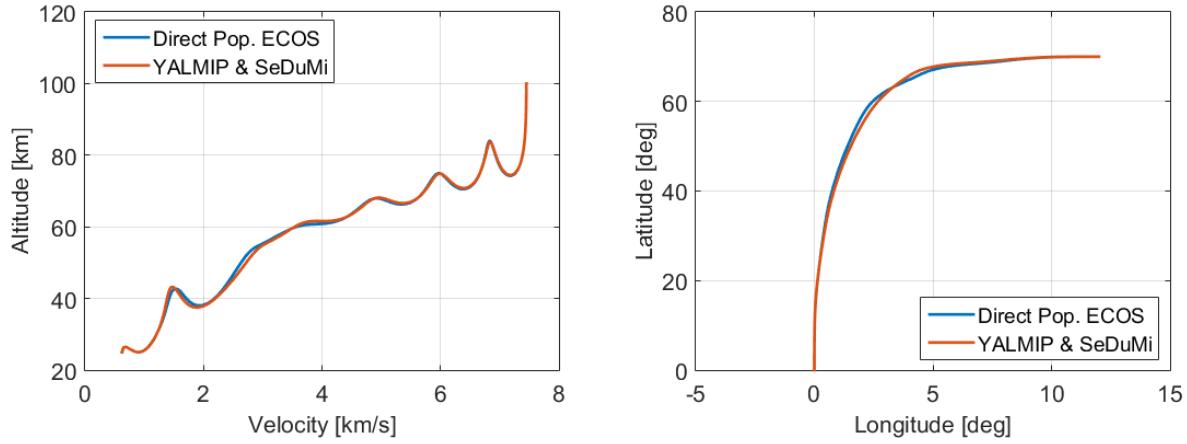
Table 3.3.
Average Per Iteration Run-time MATLAB R2015a, Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz

MATLAB	**YALMIP/SeDuMi**	**Direct Pop./ECOS**
Total	3.8 sec	0.8 sec
Modeling	3.1 sec	0.5 sec
Solver	0.7 sec	0.3 sec
Terminal Velocity	618.37 m/s	619.95 m/s



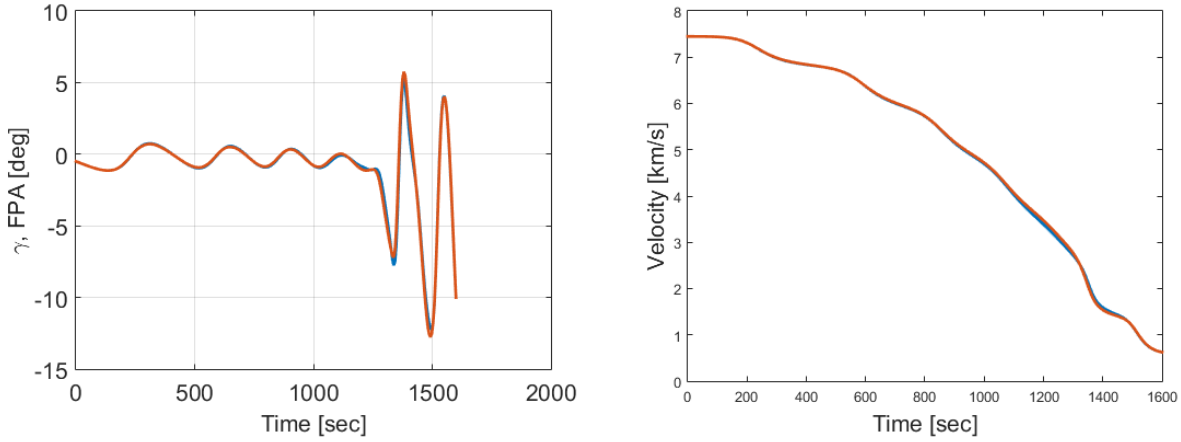
(a) 3D Trajectory. (b) Control.

Figure 3.3. Comparison of Flight Path Trajectory and Control profile.



(a) Trajectory Profile. (b) Ground Track.

Figure 3.4. Comparison of Trajectory profile and Ground Track.



(a) Flight Path Angle.

(b) Velocity Profile.

Figure 3.5. Comparison of Flight Path Angle profile and Velocity profile.

It was observed that the per iteration runtime showed very little variation over the sequential process; this was expected with a consistent number of nodes being allocated using the modeler and the ECOS solver is designed for low runtime variance. The direct population method currently is isolated to a 10 KB file and ECOS uses only 4 KB of hard drive storage. Conversely, the YALMIP application is a 2.62 MB package which can be pared down to a minimum of 480 KB for this bank angle entry problem and SeDuMi requires 34 KB of disk space. For onboard data footprint, the direct population method and ECOS solver represent 2.7% of the disk space that would be used by YALMIP and SeDuMi and both methods result in the same level of data output. This decrease in required disk space coupled with the run-time speed up demonstrated in Table 3.3 satisfies the hypothesis that the direct population method combined with the ECOS solver would create a significant reduction in software data footprint and considerably speed up the per-iteration run-time of this onboard trajectory optimization system.

3.2 Angle of Attack Control

Angle of attack control is a method of atmospheric flight guidance which modulates the magnitude of the perpendicular lift and drag force vectors of a vehicle in order to control downrange error. This flight control is characterized by a vehicle which can induce and adjust a non-zero L/D in order to control the in-plane components of the aerodynamic force vector. This control method is particularly effective for controlling downrange trajectory errors, but does not directly control crossrange errors.

Angle of attack control can be used to augment bank angle control schemes to reduce the severity of S-curves and create additional guidance control authority of the vehicle than bank angle control could achieve on its own. This combination is used effectively on many atmospheric flight vehicles and is a sister to the angle of attack and side-slip control method, both of which are incredibly versatile in atmospheric flight. As a standalone control, angle of attack has proven to be quite effective for trajectories where crossrange errors are expected to be minimal, such as trajectories those with characteristics like steep flight path angles or short time of flight from release to impact. Such trajectories often use only angle of attack control as a means to reduce computational complexity and due to limitations in mechanical control of such vehicles.

In addition to being an often used control method for hypersonic entry guidance, angle of attack control often exhibits non-linear aerodynamics coupled to the control. This can pose a challenge to convexification methods which use successive linearization to convexify the vehicle equations of motion. In this section, convexification of an angle of attack problem was explored, applied, and finally demonstrated in an onboard scenario.

A trajectory which has these characteristics is a maximum impact velocity problem with a downrange target from a downward facing initial flight path angle. Angle of attack is chosen as the control and the trajectory is assumed to operate in the two dimensional plane of altitude and latitude. This problem provided an initial platform to convexify a general angle of attack control problem with the ability to compare to an optimal control theory

solution. The vehicle chosen for this problem has a mass of 100 kg, an aerodynamic reference area of 0.3 m^2, and the following aerodynamic properties:

$$C_L = 1.5658 * \alpha \tag{3.31}$$

$$C_D = 1.6537 * \alpha^2 + 0.0612 \tag{3.32}$$

Table 3.4.
Initial Conditions and Terminal Constraints

Variable	**Initial**	**Target**
Altitude	50 km	0 km
Latitude	0 deg	0.01 deg (Δ1 km)
Velocity	4 km/s	Unconstrained
Flight Path Angle	-90 deg	Unconstrained
Angle of Attack	0 deg	Unconstrained
Angle of Attack rate	0 deg/s	Unconstrained

Table 3.5.
State and Control Constraints

Variable	**Minimum**	**Maximum**
Altitude	25 km	100 km
Latitude	-90 deg	90 deg
Velocity	10 m/s	5 km/s
Flight Path Angle	-90 deg	90 deg
Angle of Attack	-30 deg	30 deg
Angle of Attack rate	-3 deg/s	3 deg/s

3.2.1 Convexification of Angle of Attack Control Problem

For this problem, the assumptions from Section 3.1 relating to the atmosphere, non-rotating planet, and path constraints are also assumed and this problem also uses the exponential atmosphere model noted in Section 3.1.1. The dimensionless EOMs discuss in the prior section are adjusted for a two dimensional trajectory problem with angle of attack rate control using the non-dimensionalized form of Vinh's hypersonic EOMs [33] as the base dynamics with an assumed bank angle of 0, constant heading angle, and constant longitude. "Small angle" was not an assumption required to convexify the angle of attack control problem and allows for larger bounds on angle of attack such that the C_L and C_D equations remain valid valid.

$$\dot{r} = V * sin(\gamma) \tag{3.33}$$

$$\dot{\phi} = V * cos(\gamma)/r \tag{3.34}$$

$$\dot{V} = -D - sin(\gamma)/(r^2) - g * sin(\gamma) \tag{3.35}$$

$$\dot{\gamma} = L/V + (V^2 - 1/r) * cos(\gamma)/(V * r) \tag{3.36}$$

$$\dot{\alpha} = u \tag{3.37}$$

where r is the distance from the planet center nomralized by the radius of the planet, R_0, ϕ is the latitude, V is the planet relative velocity normalized by $\sqrt{R_0 g_0}$, γ is the flight path angle and α is the angle of attack. For the control states, u denotes the angle of attack rate normalized by $\sqrt{R_0/g_0}$ and was chosen to reduce the effects of high frequency oscillations within the angle of attack control.

These dynamic state equations and the dynamic control equation can then be set up as an optimal control problem of the same form as Eq. 3.19-3.23 for a maximum impact velocity optimization:

$$\underset{x,u}{\text{Min}}\ J = -V(t_f) = -V_f \tag{3.38}$$

$$\text{Subject to}: \dot{x} = f(x_*) + a(x_*)(x - x_*) + Bu \tag{3.39}$$

$$x(t_0) = x_0,\ x(t_f) = x_f \tag{3.40}$$

$$x \in [x_{min},\ x_{max}] \tag{3.41}$$

$$|u| \leq u_{max} \tag{3.42}$$

where

$$a(x_*) = \left.\frac{\partial f(x)}{\partial x}\right|_{x=x_*} = \begin{bmatrix} \frac{\partial \dot{r}}{\partial r} & \frac{\partial \dot{r}}{\partial \phi} & \frac{\partial \dot{r}}{\partial V} & \frac{\partial \dot{r}}{\partial \gamma} & \frac{\partial \dot{r}}{\partial \alpha} \\ \frac{\partial \dot{\phi}}{\partial r} & \frac{\partial \dot{\phi}}{\partial \phi} & \frac{\partial \dot{\phi}}{\partial V} & \frac{\partial \dot{\phi}}{\partial \gamma} & \frac{\partial \dot{\phi}}{\partial \alpha} \\ \frac{\partial \dot{V}}{\partial r} & \frac{\partial \dot{V}}{\partial \phi} & \frac{\partial \dot{V}}{\partial V} & \frac{\partial \dot{V}}{\partial \gamma} & \frac{\partial \dot{V}}{\partial \alpha} \\ \frac{\partial \dot{\gamma}}{\partial r} & \frac{\partial \dot{\gamma}}{\partial \phi} & \frac{\partial \dot{\gamma}}{\partial V} & \frac{\partial \dot{\gamma}}{\partial \gamma} & \frac{\partial \dot{\gamma}}{\partial \alpha} \\ \frac{\partial \dot{\alpha}}{\partial r} & \frac{\partial \dot{\alpha}}{\partial \phi} & \frac{\partial \dot{\alpha}}{\partial V} & \frac{\partial \dot{\alpha}}{\partial \gamma} & \frac{\partial \dot{\alpha}}{\partial \alpha} \end{bmatrix} \tag{3.43}$$

$$\frac{\partial \dot{r}}{\partial r} = 0,\ \ \frac{\partial \dot{r}}{\partial \phi} = 0,\ \ \frac{\partial \dot{r}}{\partial V} = \sin\gamma,\ \ \frac{\partial \dot{r}}{\partial \gamma} = V * \cos\gamma,\ \ \frac{\partial \dot{r}}{\partial \alpha} = 0,$$

$$\frac{\partial \dot{\phi}}{\partial r} = -\frac{V * \cos\gamma}{r^2},\ \ \frac{\partial \dot{\phi}}{\partial \phi} = 0,\ \ \frac{\partial \dot{\phi}}{\partial V} = \frac{\cos\gamma}{r},\ \ \frac{\partial \dot{\phi}}{\partial \gamma} = -\frac{V * \sin\gamma}{r},\ \ \frac{\partial \dot{\phi}}{\partial \alpha} = 0,$$

$$\frac{\partial \dot{V}}{\partial r} = -\frac{\partial D}{\partial r} + \frac{2 * \sin\gamma}{r^3},\ \ \frac{\partial \dot{V}}{\partial \phi} = 0,\ \ \frac{\partial \dot{V}}{\partial V} = -\frac{\partial D}{\partial V},\ \ \frac{\partial \dot{V}}{\partial \gamma} = -\frac{\cos\gamma}{r^2},\ \ \frac{\partial \dot{V}}{\partial \alpha} = -\frac{\partial D}{\partial \alpha},$$

$$\frac{\partial \dot{\gamma}}{\partial r} = \frac{\partial L}{\partial r} - \frac{V * \cos\gamma}{r^2} + \frac{2 * \cos\gamma}{V * r^3},\ \ \frac{\partial \dot{\gamma}}{\partial \phi} = 0,$$

$$\frac{\partial \dot{\gamma}}{\partial V} = \frac{\frac{\partial L}{\partial V}}{V} - \frac{L}{V^2} + \frac{\cos\gamma}{r} + \frac{\cos\gamma}{V^2 * r^2},\ \ \frac{\partial \dot{\gamma}}{\partial \gamma} = \frac{1}{V * r^2 - \frac{V}{r}} * \sin\gamma,\ \ \frac{\partial \dot{\gamma}}{\partial \alpha} = \frac{\frac{\partial L}{\partial \alpha}}{V},$$

$$\frac{\partial \dot{\alpha}}{\partial r} = 0,\ \ \frac{\partial \dot{\alpha}}{\partial \phi} = 0,\ \ \frac{\partial \dot{\alpha}}{\partial V} = 0,\ \ \frac{\partial \dot{\alpha}}{\partial \gamma} = 0,\ \ \frac{\partial \dot{\alpha}}{\partial \alpha} = 0,$$

From this set of equations, the optimal control problem can be reformatted into standard optimal control form as discussed in Section 2.2.2 using direct population of sparse matrices in triplet representation.

3.2.2 Discussion

The initial guess trajectory for this problem was a straight down trajectory from the initial conditions generated using the numerically integrated simulation discussed in Section 3.1.1. The guess and convexified dynamics were then modeled using the direct population method, Section 2.2.2, and the optimized trajectory was solved using ECOS by sequential convex programming. The resulting trajectory is very well behaved with only a rapid change in the angle of attack at the start of the trajectory before smoothing out for the remainder of the flight. The resulting trajectory was able to meet the terminal constraint requirement of a 0.01 deg latitude downrange travel distance from the initial guess of a straight down trajectory. This problem was able to reach the optimal solution using sequential convex optimization, direct population, and ECOS in a single iteration of the method at which point, none of the trust region constraints were violated and all state and limit constraints were satisfied. The total run-time of this problem was extremely low at only 0.6 seconds for the single iteration required using the same computing setup as was listed in Section 3.1.1. This drastic speed up could be the result of the trajectory being much shorter and being constrained to two degrees of freedom as compared to three for the bank angle control problem analyzed previously. Another possibility is that the convex solution space of this problem may be easier to optimize. In addition to the convex optimization solution, an optimal control theory solution was also completed for comparison using the built in MATLAB function bvp4c.

One challenge faced while working with the convexified angle of attack control problem was that the initial angle of attack and the final time had to be defined as constraints. This difference is what caused the control to jump up at the start of the trajectory in order to compensate for the inability to change the initial angle of attack.

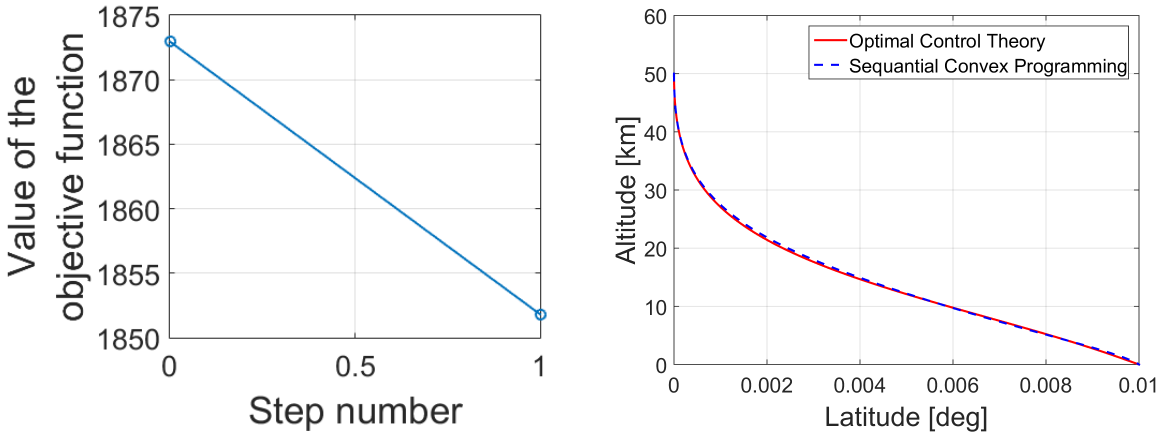
Table 3.6 shows the maximum impact velocity achieved by each optimization scheme. There is a observable difference between the maximum impact velocity for each trajectory, but the difference is small and could be attributed to the linearizations in the convexification process has introduced small errors into the vehicle dynamics which are not present in the optimal control theory solution or the initial difference in the control as discussed previously. An additional discovery found while deriving and solving for the solution to the angle of attack control problem was that it is extremely important to non-dimensionalize as much of the problem as possible. The first derivation of the convexified angle of attack control problem, which was derived from the dimensional equations of motion could not converge even when using the optimal control theory solution as the initial guess trajectory. For the problems solved in this paper, the solution was not only more likely to converge when fully non-dimensionalized, but generally converged in several fewer iterations.

Table 3.6.
Optimal Maximum Impact Velocity

Convex Solution	Optimal Control Theory
1851.81 m/s	1851.86 m/s

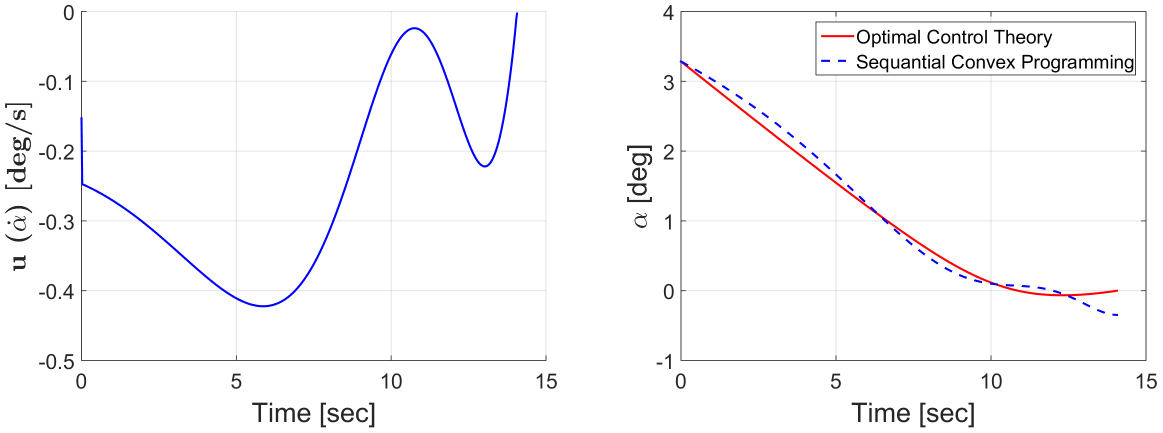
Figures 3.6-3.8 show the optimized solution for the convexified, angle of attack optimal control problem for a vehicle with an initial downward facing trajectory and a target on the ground at 0.01 deg latitude downrange (approx. 1 km) from the initial position. This problem was successfully solved by the method of sequential convex programming to find an optimal control profile to maximize the impact velocity at the target. Also included in these figures is an optimal control theory solution using Vihn's EOMs (not dimensionless) for the same two dimensional angle of attack problem. The optimal control solution uses angle of attack as the control method, but does so directly without using angle of attack rate as the decoupled control since optimal control theory is less prone to high frequency oscillations in the solution. This optimal control theory case was solved using bvp4c in MATLAB and provides a good comparison case for the convex solution. Both optimization methods resulted in trajectories which follow similar trends. The altitude and velocity compare favorably between the two optimized solutions while the flight path angle and angle of attack profiles show similar

trends, but have some differences. These differences could be due to the differences in the control utilized for each optimization and the linearizations present within the convexification process.



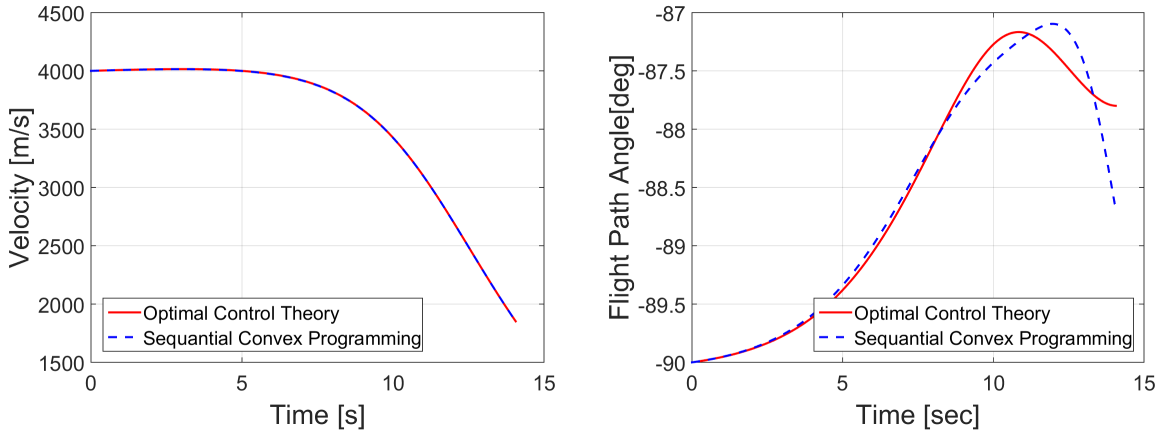
(a) Objective Function. (b) Trajectory.

Figure 3.6. Maximum impact velocity steps and trajectory comparison to optimal control theory.



(a) Control (b) Angle of Attack Profile

Figure 3.7. Control and comparison of angle of attack optimal control theory.



(a) Velocity Profile (b) Flight Path Angle

Figure 3.8. Comparison of velocity and flight path angle to optimal control theory.

3.2.3 Convexified Lofting Angle of Attack Control

Using the formulation derived in the previous section, other angle of attack control problems can be solved within the same assumptions and equations of motion. For example, a re-entry trajectory optimal control problem for which the objective is now to minimize velocity at a particular altitude rather than maximize it. In this re-entry problem, the vehicle expected to loft and reduce velocity in the atmosphere before the terminal point is reached. This vehicle is defined as a blunt-body vehicle with a mass of 86 kg and a base diameter of 0.64 m and the following aerodynamic coefficients:

$$C_L = 1.293 * \alpha \tag{3.44}$$

$$C_D = 3.071 * \alpha^2 + 0.0802 \tag{3.45}$$

with boundary conditions and constraints detailed in Tables 3.7-3.8

For this angle of attack optimal control problem, it is assumed that another EDL system (parachute, retro-propulsion, etc.) will be activated at the 49 km altitude mark. This second system is assumed to either be able to handle the cross-range error associated with low altitude winds or the vehicle is expected to land in a location where cross-range error is a negligible risk factor. As such, the optimal control trajectory through the hypersonic and

Table 3.7.
Initial Conditions and Terminal Constraints

Variable	**Initial**	**Target**
Altitude	100 km	49 km
Latitude	0 deg	40 deg
Velocity	4 km/s	Unconstrained
Flight Path Angle	-90 deg	Unconstrained
Angle of Attack	0 deg	Unconstrained
Angle of Attack rate	0 deg/s	Unconstrained

Table 3.8.
State and Control Constraints

Variable	**Minimum**	**Maximum**
Altitude	49 km	100 km
Latitude	-90 deg	90 deg
Velocity	10 m/s	5 km/s
Flight Path Angle	-90 deg	90 deg
Angle of Attack	-50 deg	50 deg
Angle of Attack rate	-3 deg/s	3 deg/s

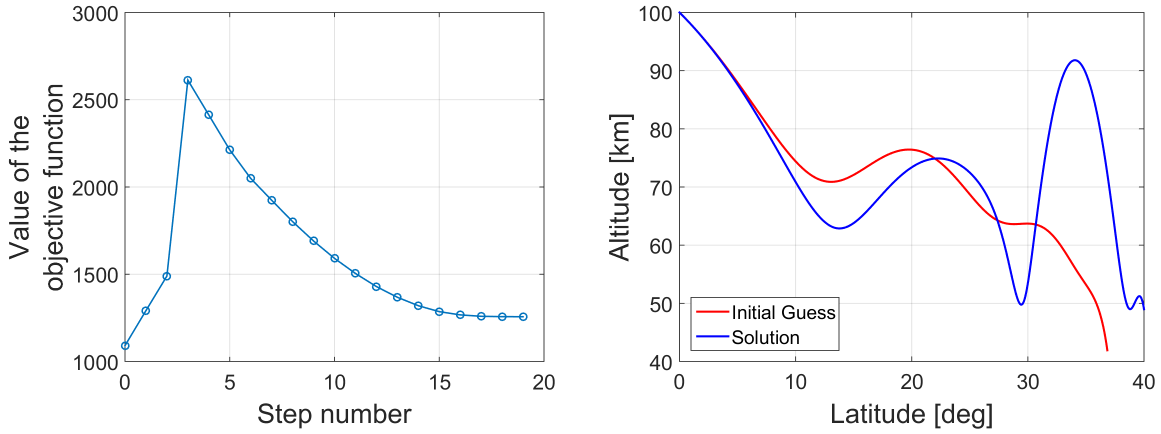
supersonic regimes can be assumed to have an optimal solution within the two dimensional plane and using angle of attack as the control mechanism. The problem is also constrained such that the vehicle cannot descend below an altitude of 49.5 km in order to avoid a secondary EDL system trigger and cannot loft above the initial altitude to avoid skipping out of the atmosphere.

The initial guess trajectory for this problem was generated using numerical integration of the EOMs, Eqs. 3.33 - 3.37, with a constant angle of attack of 40 deg. This initial guess has some basic lofting behavior, but does not meet the terminal point constraint and is not optimized to minimize velocity at the terminal point. Figures 3.9 - 3.11 represent the optimized trajectory obtained for the problem described using sequential convex programming.

In Figure 3.9a, the initial value of the objective function is below the final optimized value. For this problem, the vehicle has increased deceleration as lower altitudes are reached. However, the initial trajectory does not meet the terminal point constraints, so the sequential

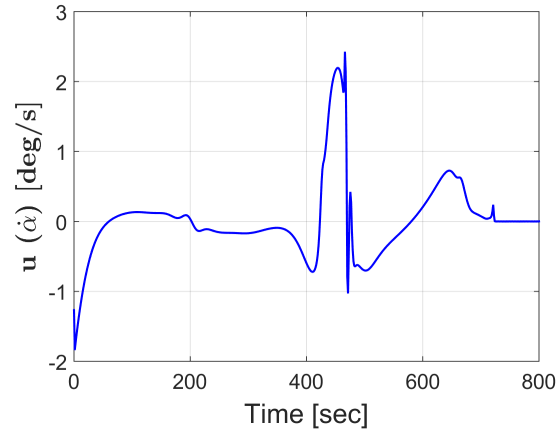
method steps through iterations working towards satisfying the problem constraints. Once these constraints have been satisfied, iterations start to march through further optimized values of the objective function.

Figure 3.9b shows the comparison of the initial guess and the optimized solution. For the optimized trajectory, there is significant lofting behavior observed which takes advantage of atmospheric deceleration to reduce velocity prior to reaching the terminal point. If this optimized trajectory was observed in an engineering design cycle, it is likely that the vehicle initial states would be re-targeted to a location which would allow for additional lofting over the course of the trajectory and allowing for longer latitude ground track travel. This provides additional versatility to quickly analyze initial trajectory design spaces in a rapid manner to determine if engineering changes are required in the design cycle, this case being a re-targeting of the initial state following the in-space delivery phase.

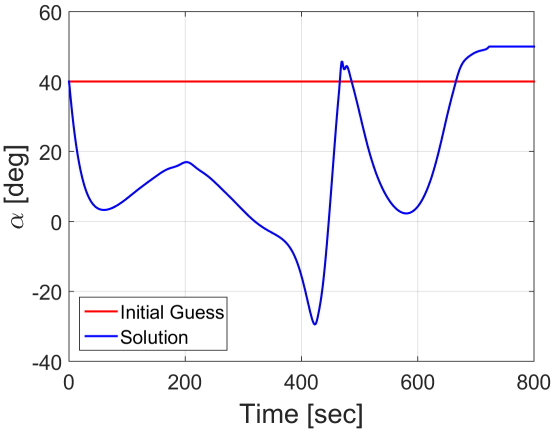


(a) Objective Function (b) Trajectory

Figure 3.9. Maximum impact velocity steps and trajectory comparison to initial guess.

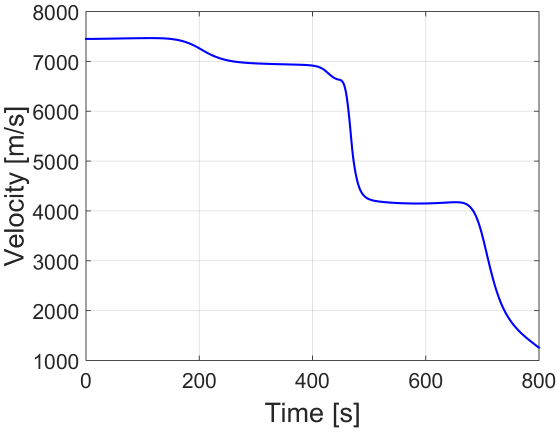


(a) Control

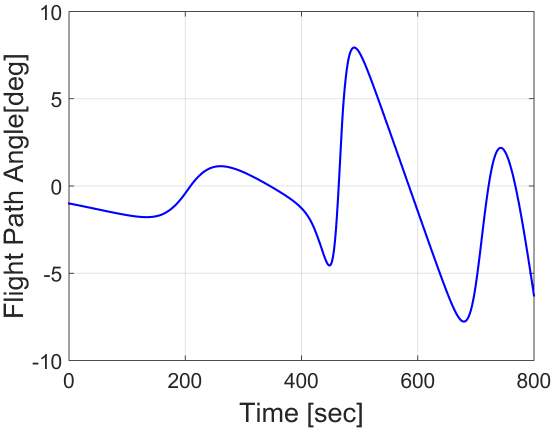


(b) Angle of Attack Profile

Figure 3.10. Control and resulting angle of attack profiles.



(a) Velocity Profile



(b) Flight Path Angle

Figure 3.11. Velocity and flight path angle profiles.

3.2.4 Onboard Application of Convexified Angle of Attack Control Optimization

In order to test the validity of using the convexified angle of attack control optimization scheme for onboard applications, the lofting angle of attack control problem described in Section 3.2.3 was used as a baseline reference trajectory for a simulated flight. At twenty regular intervals throughout the flight, the vehicle was forcibly moved off the path of the optimized reference to simulate sampling of imperfect reference following. Imperfect reference following is often the result of encountering unexpected atmospheres or anomalous events which were not predicted prior to flight. This optimization architecture is not being simulated as a guidance scheme, so it is unlikely that continuous state feedback would be available. As such, the optimization architecture was rerun at each of these twenty perturbed points using the previous optimized trajectory as the initial guess with fixed initial states at the new perturbed states. The perturbations were defined randomly at each point to simulate breakaways from the previous optimized reference trajectory such that a new reference trajectory becomes necessary using the boundaries on the state perturbations shown in Table 3.9:

Table 3.9.
Boundaries of State Variable Perturbations

Perturbed State Variable	**Minimum**	**Maximum**
Altitude	-1 km	1 km
Latitude (approx. 1km)	-0.01 deg	0.01 deg
Velocity	-50 m/s	50 m/s
Flight Path Angle	-1 deg	1 deg

For each new reference trajectory, the initial state conditions are fixed at the perturbed state values and the angle of attack rate control is free under the assumption that any rate change within the control boundary is achievable in a negligible amount of time. The consecutive objective function results are shown in Figure 3.12 and the optimized trajectories for this simulated flight are shown in Figures 3.13 - 3.17. Each perturbation location in is marked with the distinguishable color matching its associated line on the plot.

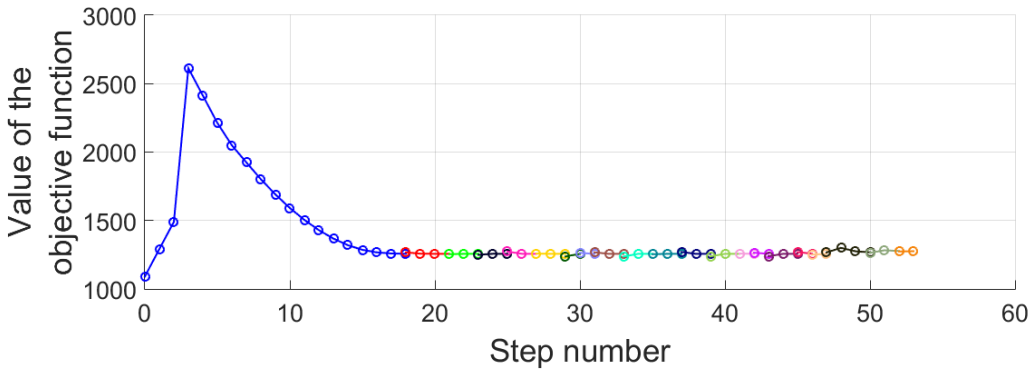


Figure 3.12. Consecutive objective function for each optimized reference trajectory.

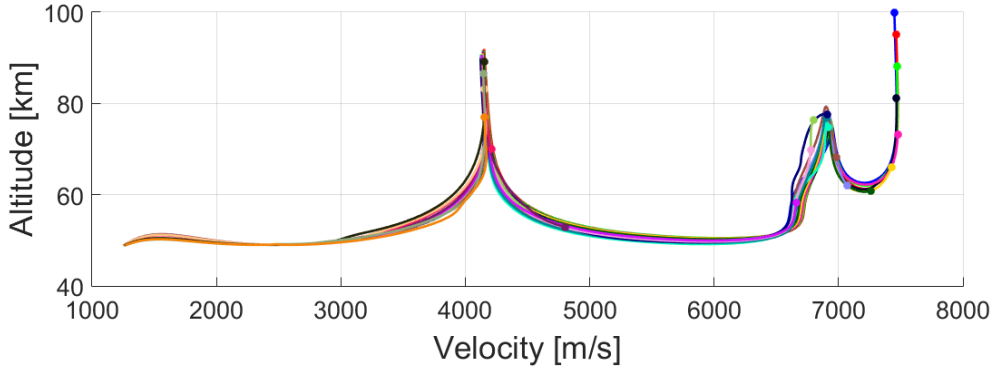


Figure 3.13. Consecutive velocity vs altitude profiles.

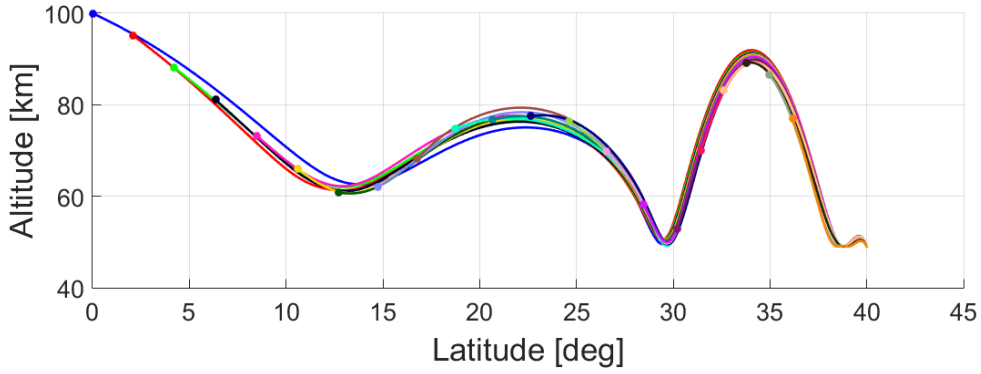


Figure 3.14. Consecutive trajectory profiles.

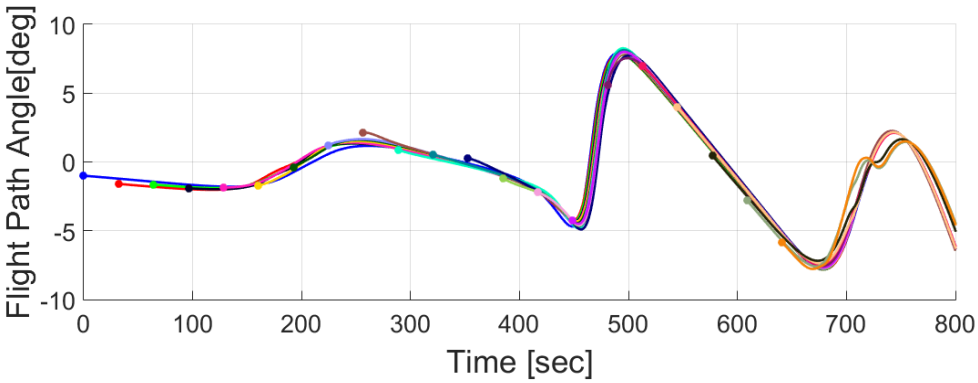


Figure 3.15. Consecutive flight path angle profiles.

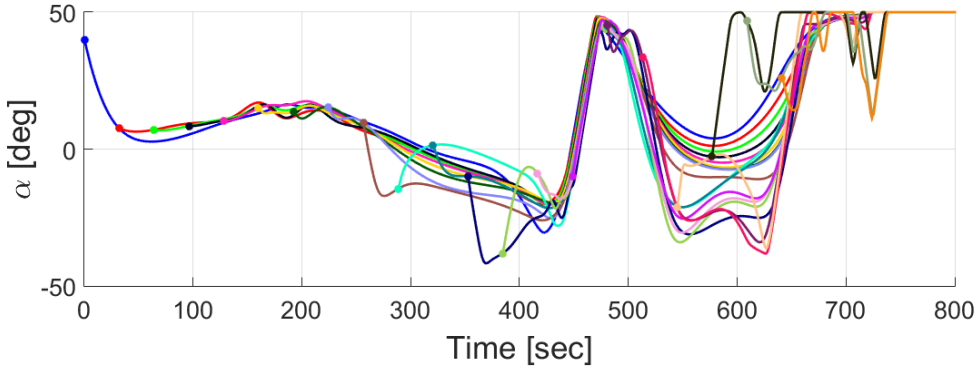


Figure 3.16. Consecutive angle of attack profiles.

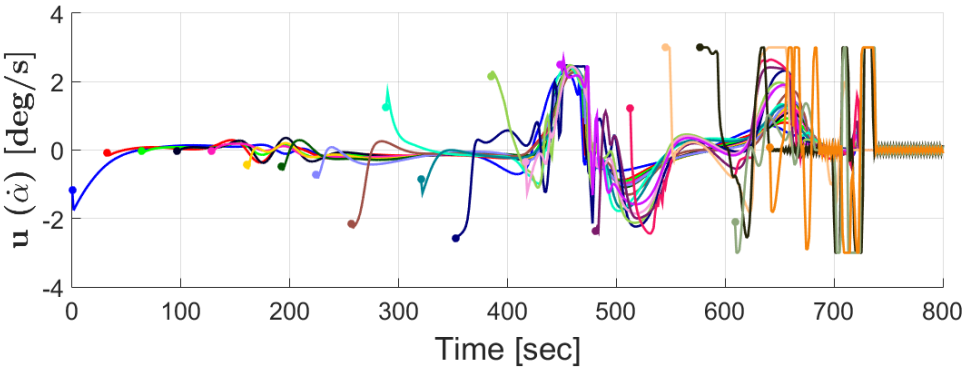


Figure 3.17. Consecutive control profiles.

It was observed that after the initial reference trajectory was generated from the initial guess, same as Section 3.2.3, that subsequent optimizations required less iterations to converge due using the previously optimized reference trajectory as the new initial guess. It was also observed that as the vehicle progressed through the flight, it was possible to reduce runtime further by eliminating nodes with each reduction in total flight time.

In this problem, six hundred nodes were used to generate the initial reference trajectory with 24 nodes being removed after reaching each perturbation point resulting in one-hundred twenty nodes used in the last optimized reference trajectory. This reduction was measured such that the number of nodes used to represent each optimized trajectory was equal to the number of nodes used to represent that same segment of the initial optimized trajectory resulting in zero loss to vehicle representation as the number of nodes decreased. The resulting 80% reduction in nodes over the flight resulted in appoximately an 80% reduction in per iteration runtime, which translates to a 5x faster optimization per iteration of the sequential convex programming problem showing a linear relationship between per iteration runtime and the number of nodes. This is an advantageous trait if faster adjustments to the reference trajectory are necessary as the vehicle approaches the target. While there is a likely a lower limit to the number of nodes necessary to represent a realistic trajectory, this reduction in nodes was able to reduce both the problem formulation time and optimizer runtime due to the reduction in total calculations of the equality and inequality matrices A and G, and vectors b and h. Additionally, it was observed that regular re-optimization of the trajectory required only two or three iterations to successfully solve for a new optimal control

as opposed to the initial optimization from a bad initial guess. This follows the hypothesis that better initial guesses result in faster convergence of the sequential method resulting in faster re-optimization of the trajectory when a previously optimized trajectory is used as the guess. However, it is expected that additional trajectory re-optimization requirements or more severe changes to the perturbed states could result in a larger number of expected iterations per re-optimization though this was not tested.

A lower number of nodes, 300, was also tested and resulted in a runtime reduction of 40% with convergence achieved in a similar number of iterations per optimization to achieve convergence. However, the solution to the 300 node problem did not match the 600 node solution and the control authority of a 50% reduction in nodes generally showed larger control changes to re-optimize the perturbed trajectories. It is expected that the initial node number will need to be determined by on-ground experimentation based on the expected level of control authority required to achieve mission success and expected trajectory time. It was also observed that the per iteration runtime is non-sensitive to changes in the initial conditions while the number of iterations required to converge is more affected by off nominal initial conditions as the guess trajectory diverges from the optimized trajectory.

With this effect of node numbers exhibiting a large influence over the per iteration runtime, the number of nodes required to represent a particular trajectory accurately also came into question. To investigate this phenomenon, a convergence study was run for an increasing number of nodes from 100 to 1000 in increments of 50. The result of the first convergence study resulted in 500 nodes being the minimum number of nodes required on a scale in 100 node increments. The same convergence study was run again at a finer increment of 5 nodes to better represent the minimum requirement. The result of this second study was that a minimum of 525 nodes were required to represent the trajectory such that additional nodes resulted in no major change in the control law with 600 nodes being the point at which additional nodes contributed negligible change to the optimal control law. This convergence study has been visually represented in Figures 3.18-3.20 and shows the convergence from the low number of nodes to larger numbers of nodes and the point of negligible returns as the profiles turn blue.

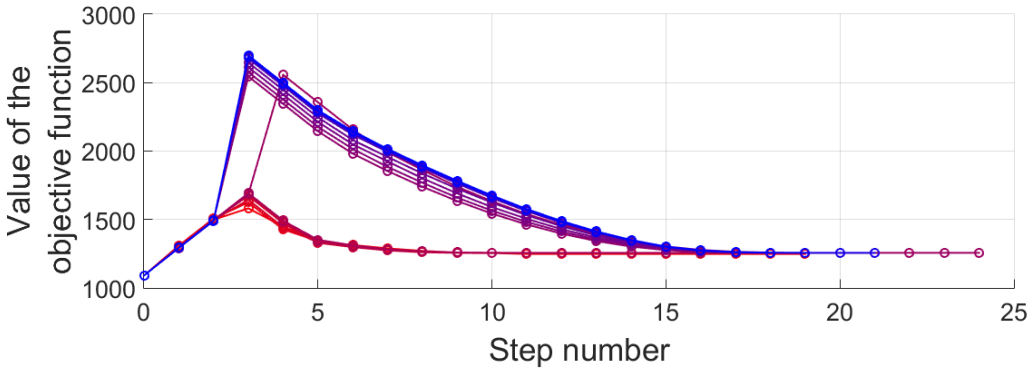


Figure 3.18. Required Nodes Convergence Study: Objective, 100 nodes *(red)* to 1000 nodes *(blue)*.

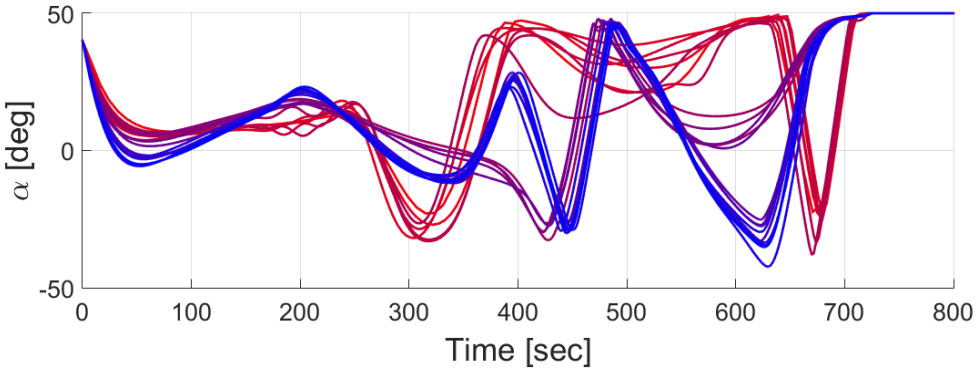


Figure 3.19. Required Nodes Convergence Study: Angle of Attack, 100 nodes *(red)* to 1000 nodes *(blue)*.

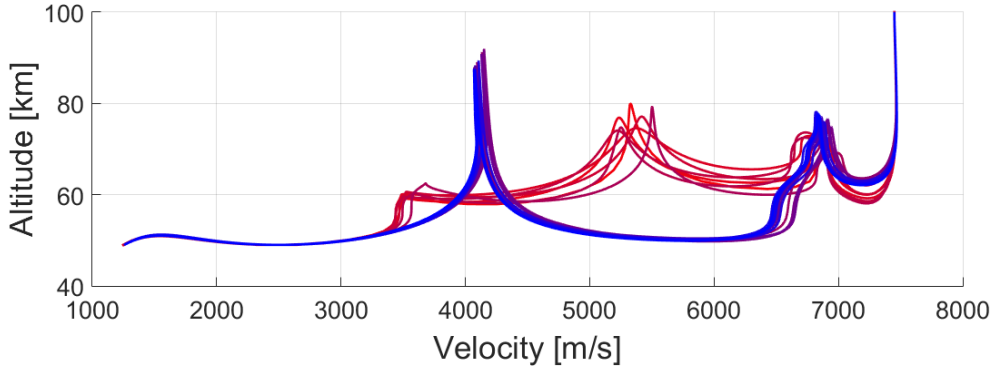


Figure 3.20. Required Nodes Convergence Study: Trajectory Profile, 100 nodes *(red)* to 1000 nodes *(blue)*.

4. SUMMARY

As the capability of hardware onboard aerospace vehicles increases, the interest in onboard trajectory optimization systems has increased with it. The purpose of these onboard optimization systems is to autonomously generate new reference trajectories for the onboard guidance algorithm such that trajectory re-planning can be done onboard. In order for a trajectory optimization system to be classified as "onboard ready," it must be computationally efficient, have a fast runtime for iterative use, and have a small data footprint to fit on limited hardware data storage. One method which generally satisfies these conditions is convex optimization. Convex optimization is a process to minimize convex functions over convex sets and solutions can be found in polynomial-time, which is significantly faster than standard mathematical optimization which is non-deterministic polynomial hard. While optimal control problems for atmospheric flight are not naturally convex, they can be reformulated into a convex form using successive linearizations and solving the resulting system of equations using sequential programming methods.

The convexified optimal control problem can be solved using a direct collocation method paired with an iterative process to converge to an optimal solution within the convex solution space. In this thesis, an architecture is proposed which directly populates sparse matrices with the convexified collocated optimal control problem and solved using an open-source SOCP solver to achieve computational efficiency and minimal data storage requirements. Using a bank angle control re-entry problem, this architecture is compared to a traditional research software architecture using an equation modeler and a standard solver package wherein the direct population method and embedded solver solution compared favorably to the solution generated using the modeler and solver combination. Additionally, the direct population method shows a 6x faster problem formulation time per iteration and a 2.3x faster solving time by sequential convex programming. It was also observed that the proposed software architecture required 2.7% of the software data allocation of the previously demonstrated, research oriented architecture depending on the size of the required software package. This comparison was completed using MATLAB R2015a and a standard desktop Intel CPU, and is expected to increase in computational efficiency further by implementing

the direct population method and an embedded SOCP solver into a C or C++ programming architecture and software optimization, though this was not tested in this thesis. This comparison successfully demonstrated that the proposed software architecture was able to drastically reduce iteration runtime and reduce the necessary onboard software data allocation to achieve the capabilities needed for an onboard trajectory optimization system.

Convexification was also used to transform angle of attack optimal control problems; a low time of flight, maximum velocity problem and a lofting minimal velocity problem. The convexified angle of attack optimal control problem was solved using the sequential convex programming method and was compared to an optimal control theory solution to the maximum impact velocity problem. The convex solution compared favorably with the optimal control theory solution with a 0.05 m/s difference observed in the maximized velocity at impact objective. The convexified angle of attack control was also applied to an entry trajectory with the objective of minimizing the velocity at a particular terminal point subject to state and control constraints. An initial guess was provided for this problem which did not satisfy the boundary constraints and the expected result of this trajectory was a highly lofting trajectory which would be required to reduce velocity in a limited downrange prior to reaching the terminal point. This trajectory was optimized successfully using sequential convex programming, resulting in the expected lofting trajectory prior to reaching the terminal point. As the first application of a convexified angle of attack control, these solutions provide evidence that a control method which is non-linearly coupled to the aerodynamics can be successfully convexified and solved using sequential convex programming.

It was also observed that a better initial guess can significantly reduce the number of iterations required to converge to a solution, enabling sequential convex programming for autonomous onboard systems which rapidly regenerate new trajectories for path alterations and unexpected environments with sub-second iteration runtimes. A final observation was that continuous re-optimizations of the trajectory can be solved using only the nodes previously used to represent the remaining trajectory segment, speeding up re-optimization as the trajectory approaches the terminal point resulting in sub-second optimal convergence times as the number of nodes is reduced.

5. RECOMMENDATIONS

A significant observation made during this investigation was that final time had to be specified within an extremely small tolerance to the actual final time of the optimal trajectory in order for the convexified problem to execute successfully. For example, specifying a time of flight which is too short may make the optimization infeasible over a given solution space; the optimizer would not be able to find any controls that allow the trajectory to meet the terminal point constraints. Adding functionality for an unspecified final time would greatly reduce the risk of failure to meet boundary constraints at the end of the trajectory. One option used in other direct collocation methods and indirect boundary value problems is to define an additional parameter which is a constant that defines the time step between nodes. The optimizer can then adapt the time of flight between nodes, altering the final time and allowing for solutions that satisfy the all trajectory constraints, though it is unclear if the addition of this time parameter would still result in a convex system of equations. Another option is to employ a Chebyshev-Gauss-Lobatto quadrature formula for optimal spacing of the timestep.[42]

Another point of future work is to incorporate the convexified trajectory optimization into an autonomous flight vehicle system and begin testing in conjunction with a guidance scheme. This coupled system could then be used to simulate trajectories where the sequential convex programming method is used to generate new reference trajectories throughout the trajectory which are then passed to the guidance for reference following. A coupled system of sequential convex programming for trajectory optimization and a reference following guidance scheme could also be run in shadow-mode on board a flight vehicle to assess efficacy of the scheme in real world conditions.

Further study should be done to investigate other collocation and direct method optimizations which can take advantage of sequential convex programming. Specifically, methods which can provide passive speed-ups in solver runtime and problem formulation time, or allow for a reduced number of control points to solve for an accurate optimized trajectory.

Lastly, angle of attack and bank angle are not the only controls used in flight vehicle control and there are also instances where multiple controls are used to achieve the desired

control authority. Sideslip angle, drag modulation, and shape altering vehicles are three examples of control which have not been investigated for trajectory optimization via sequential convex programming As of yet, there is no evidence that convexification can be used for a vehicle with multiple controls. Investigation into convexification of multi-controlled systems would be expected to include both coupled controls, like angle of attack and bank angle, and decoupled controls, like direct force control.

REFERENCES

[1] Z. Wang and M. J. Grant, "Constrained Trajectory Optimization for Planetary Entry via Sequential Convex Programming," *Journal of Guidance, Control, and Dynamics*, 2017. DOI: 10.2514/1.G002150.

[2] Z. Wang and M. J. Grant, "Hypersonic Trajectory Optimization by Sequential Semidefinite Programming," *AIAA Atmospheric Flight Mechanics Conference*, 2017. DOI: 10.2514/6.2017-0248.

[3] Z. Wang and M. J. Grant, "A Convex Approach to Minimum-Time Low-Thrust Trajectory Optimization," *2018 AIAA Guidance, Navigation, and Control Conference*, 2018. DOI: 10.2514/6.2018-1577.

[4] J. Mattingley and S. Boyd, "Real-Time Convex Optimization in Signal Processing," *IEEE Signal Processing Magazine*, vol. 27, no. 3, pp. 50–61, 2010. DOI: 10.1109/MSP.2010.936020.

[5] J. Mattingley and S. Boyd, "Automatic code generation for real-time convex optimization," in *Convex Optimization in Signal Processing and Communications*, D. P. Palomar and Y. C. Eldar, Eds. Cambridge University Press, 2009, pp. 1–41. DOI: 10.1017/CBO9780511804458.002.

[6] S. Boyd and L. Vandenberghe, "Convex Optimization - Lectures," *Stanford University*, 2014, ISSN: 02601826. DOI: 10.1039/b202577a.

[7] S. Boyd, E. Busseti, S. Diamond, R. N. Kahn, K. Koh, P. Nystrup, and J. Speth, *Multi-Period Trading via Convex Optimization*, 1. NOW Publishers, 2016, vol. 3, pp. 1–76. DOI: 10.1561/2400000023.

[8] C. Brandt, *Convexity: the perfect trade?* 2016. [Online]. Available: https://www.ipe.com/convexity-the-perfect-trade/10013565.article.

[9] T. Pennanen, "Introduction to convex optimization in financial markets," *Mathematical Programming*, vol. 134, no. 1, pp. 157–186, 2012. DOI: 10.1007/s10107-012-0573-4.

[10] X. Liu, Z. Shen, and P. Lu, "Entry Trajectory Optimization by Second-Order Cone Programming," *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 2, pp. 227–241, Aug. 2015, ISSN: 0731-5090. DOI: 10.2514/1.G001210.

[11] A. A. Ahmadi, A. Olshevsky, P. A. Parrilo, and J. N. Tsitsiklis, "NP-hardness of Deciding Convexity of Quartic Polynomials and Related Problems," *Mathematical Programming*, vol. 137, no. 1-2, pp. 453–476, Feb. 2013. DOI: 10.1007/s10107-011-0499-2.

[12] L. Hardesty, *Explained: P vs. NP*, 2016. [Online]. Available: https://news.mit.edu/2009/explainer-pnp.

[13] A. A. Ahmadi and P. A. Parrilo, "A Complete Characterization of the Gap between Convexity and SOS-Convexity," *SIAM Journal on Optimization*, vol. 23, no. 2, pp. 811–833, Jun. 2013. DOI: 10.1137/110856010.

[14] L. D. Elsgolc, *Calculus of Variations*. Pergamon Press, republished in 2007 by Dover Publications, 1961.

[15] H. P. Geering, *Optimal Control with Engineering Applications*. Springer-Verlag Berlin Heidelberg, 2007, pp. 23–74, ISBN: 978-3-540-69437-3. DOI: 10.1007/978-3-540-69438-0.

[16] O. von Stryk and R. Bulirsch, "Direct and indirect methods for trajectory optimization," *Ann Oper Res*, vol. 37, pp. 357–373, 1992. DOI: 10.1007/BF02071065.

[17] A. E. Bryson and Y. Ho, *Applied Optimal Control: Optimization, Estimation, and Control*. Washington: Hemisphere Pub. Corp., 1975. DOI: 10.1109/TSMC.1979.4310229.

[18] H. Schaub and J. L. Junkins, *Analytical Mechanics of Space Systems*. AIAA American Institute of Aeronautics and Ast.; 4th edition, 2018, ISBN: 978-1-62410-521-0. DOI: 10.2514/4.105210.

[19] J. Renes, *On the Use of Splines and Collocation in a Trajectory Optimization Algorithm Based on Mathematical Programming*, ser. NLR TR. NLR, 1978. [Online]. Available: https://books.google.com/books?id=2E-pYgEACAAJ.

[20] D. Kraft, "On Converting Optimal Control Problems into Nonlinear Programming Problems," *Computational Mathematical Programming*, vol. 15, pp. 261–280, 1985. DOI: .

[21] C. Hargraves and S. Paris, "Direct trajectory optimization using nonlinear programming and collocation," *Journal of Guidance, Control, and Dynamics*, vol. 10, no. 4, pp. 338–342, 1987. DOI: 10.2514/3.20223. [Online]. Available: https://doi.org/10.2514/3.20223.

[22] Y. M. Agamawi and A. V. Rao, "Exploiting Sparsity in Direct Orthogonal Collocation Methods for Solving Multiple-Phase Optimal Control Problems," in *2018 Space Flight Mechanics Meeting*, 2018. DOI: 10.2514/6.2018-0724.

[23] B. T. Burchett, "A Gauss Pseudospectral Collocation for Rapid Trajectory Prediction and Guidance," in *AIAA Atmospheric Flight Mechanics Conference*, 2017. DOI: 10.2514/6.2017-0246.

[24] E. R. Pager and A. V. Rao, "A Proximal Method for the Numerical Solution of Singular Optimal Control Problems Using a Modified Radau Collocation Method," in *AIAA Scitech 2020 Forum*, 2020. DOI: 10.2514/6.2020-0377.

[25] M. Kelly, "An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation," *SIAM Rev.*, vol. 59, pp. 849–904, 2017.

[26] B. Açikmeşe, J. M. I. Carson, and L. Blackmore, "Lossless Convexification of Nonconvex Control Bound and Pointing Constraints of the Soft Landing Optimal Control Problem," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 6, Nov. 2013. DOI: 10.2514/6.2018-1315.

[27] X. Liu and P. Lu, "Solving Nonconvex Optimal Control Problems by Convex Optimization," *Journal of Guidance, Control, and Dynamics*, vol. 37, no. 3, pp. 750–765, Feb. 2014, ISSN: 0731-5090. DOI: 10.2514/1.62110.

[28] M. Sagliano and E. Mooij, "Optimal Drag-Energy Entry Guidance via Pseudospectral Convex Optimization," *AIAA Guidance, Navigation, and Control Conference*, no. 1315, Jan. 2018. DOI: 10.1109/TCST.2012.2237346.

[29] F. Palacios-Gomez, L. Lasdon, and M. Engquist, "Nonlinear Optimization by Successive Linear Programming," *Manage. Sci.*, vol. 28, no. 10, pp. 1106–1120, Oct. 1982, ISSN: 0025-1909. DOI: 10.1287/mnsc.28.10.1106.

[30] S. P. Banks and K. Dinesh, "Approximate Optimal Control and Stability of Nonlinear Finite- and Infinite-Dimensional Systems," *Annals of Operations Research*, vol. 98, no. 1, pp. 19–44, 2000, ISSN: 1572-9338. DOI: 10.1023/A:1019279617898.

[31] M. Tomas-Rodriguez and S. P. Banks, *Linear, time-varying approximations to nonlinear dynamical systems. With applications in control and optimization*. Springer, Lecture Notes in Control and Information Sciences, Jan. 2010, vol. 400. DOI: 10.1007/978-1-84996-101-1.

[32] *MATLAB version 8.5.0.197613 (R2015a)*, The Mathworks, Inc., Natick, Massachusetts, 2015.

[33] N. Vinh, A. Busemann, and R. Culp, *Hypersonic and Planetary Entry Flight Mechanics*. The University of Michigan Press, Feb. 1980.

[34] J. Löfberg, "YALMIP : A Toolbox for Modeling and Optimization in MATLAB [software]," in *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[35] J. Sturm, "Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones [software]," *Optimization Methods and Software*, vol. 11–12, pp. 625–653, 1999, Version 1.05 available from `http://fewcal.kub.nl/sturm`.

[36] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP Solver for Embedded Systems [software]," *European Control Conference (ECC)*, Jul. 2013. DOI: 10.23919/ecc.2013.6669541.

[37] J. T. Betts and W. P. Huffman, "Application of sparse nonlinear programming to trajectory optimization," *Journal of Guidance, Control, and Dynamics*, vol. 15, no. 1, pp. 198–206, 1992. DOI: 10.2514/3.20819.

[38] M. Eding, *Data Structures - Sparse Matrices*, 2019. [Online]. Available: https://matteding.github.io/2019/04/25/sparse-matrices/.

[39] J. Brownlee, *A Gentle Introduction to Sparse Matrices for Machine Learning*, 2018. [Online]. Available: https://machinelearningmastery.com/sparse-matrices-for-machine-learning/.

[40] A. J. Roenneke and P. J. Cornwell, "Trajectory control for a low-lift re-entry vehicle," *Journal of Guidance, Control, and Dynamics*, vol. 16, no. 5, pp. 927–933, 1993. DOI: 10.2514/3.21103.

[41] G. F. Mendeck and L. Craig McGrew, "Entry Guidance Design and Postflight Performance for 2011 Mars Science Laboratory Mission," *Journal of Spacecraft and Rockets*, vol. 51, no. 4, pp. 1094–1105, 2014. DOI: 10.2514/1.A32737.

[42] N. Srinivas and K. Schroeder, "Collocation Method and Model Predictive Control for optimal control strategies of a system of EDL vehicles," in *ASCEND 2020*, 2020. DOI: 10.2514/6.2020-4121.