New Optimal-Control-Based Techniques for Midcourse Guidance of Gun-Launched Guided Projectiles

Emmanuel E. Skamangas

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Aerospace Engineering

Jonathan Black, Chair
Craig Woolsey
John Lawton
Mazen Farhood
Cornel Sultan

February 22, 2021
Blacksburg, Virginia

Keywords: Optimal Control, Costate Estimation, Constraints, etc.

New Optimal-Control-Based Techniques for Midcourse Guidance of Gun-Launched Guided Projectiles

Emmanuel E. Skamangas

(ABSTRACT)

The following is an exploration into the optimal guidance and control of gun-launched guided projectiles. Unlike their early counterparts, modern-day gun-launched projectiles are capable of considerable accuracy. This ability is enabled through the use of control surfaces, such as fins or wings, which allow the projectile to maneuver towards a target. These aerodynamic features are part of a control system which lets the projectile achieve some effect at the target. With the advent of very high velocity guns, such as the Navy's electromagnetic railgun, these systems are a necessary part of the projectile design. This research focuses on a control scheme that uses the projectile's angle of attack as the single control in the development of an optimal control methodology that maximizes impact velocity, which is directly related to the amount of damage inflicted on the target. This novel approach, which utilizes a reference trajectory as a seed for an iterative optimization scheme, results in an optimal control history for a projectile. The investigation is geared towards examining how poor an approximation of the true optimal solution that reference trajectory can be and still lead to the determination of an optimal control history. Several different types of trajectories are examined for their applicability as a reference trajectory. Although the use of aerodynamic control surfaces enables control of the projectile, there is a potential down side. With steady development of guns with longer ranges and higher launch velocities, it becomes increasingly likely that a projectile will fly into a region of the atmosphere (and beyond) in which there is not sufficient airflow over the control surfaces to maintain projectile control. This research is extended to include a minimum dynamic pressure constraint in the problem; the imposition of such a constraint is not examined in the literature. Several methods of

adding the constraint are discussed and a number of cases with varying dynamic pressure limits are evaluated. As a result of this research, a robust methodology exists to quickly obtain an optimal control history, with or without constraints, based on a rough reference trajectory as input. This methodology finds its applicability not only for gun-launched weapons, but also for missiles and hypersonic vehicles.

New Optimal-Control-Based Techniques for Midcourse Guidance of Gun-Launched Guided Projectiles

Emmanuel E. Skamangas

(GENERAL AUDIENCE ABSTRACT)

As the name implies, optimal control problems involve determining a control history for a system that optimizes some aspect of the system's behavior. In aerospace applications, optimal control problems often involve finding a control history that minimizes time of flight, uses the least amount of fuel, maximizes final velocity, or meets some constraint imposed by the designer or user. For very simple problems, this optimal control history can be analytically derived; for more practical problems, such as the ones considered here, numerical methods are required to determine a solution.

This research focuses on the optimal control problem of a gun-launched guided projectile. Guided projectiles have the potential to be significantly more accurate than their unguided counterparts; this improvement is achieved through the use of a control mechanism. For this research, the projectile is modeled using a single control approach, namely using the angle of attack as the only control for the projectile. The angle of attack is the angle formed between the direction the projectile is pointing and the direction it is moving (i.e., between the main body axis and the velocity vector of the projectile). An approach is then developed to determine an optimal angle of attack history that maximizes the projectile's final impact velocity. While this problem has been extensively examined by other researchers, the current approach results in the analytical determination of the costate estimates that eliminates the need to iterate on their solutions.

Subsequently, a minimum dynamic pressure constraint is added to the problem. While ex-

tensive investigation has been conducted in the examination of a maximum dynamic pressure constraint for aerospace applications, the imposition of a *minimum* represents a novel body of work. For an aerodynamically-controlled projectile, (i.e., one controlled with movable surfaces that interact with the air stream), dropping below a minimum dynamic pressure may result in loss of sufficient control. As such, developing a control history that accommodates this constraint and prevents the loss of aerodynamic control is critical to the ongoing development of very long range, gun-launched guided projectiles. This new methodology is applied with the minimum dynamic pressure constraint imposed and the resulting optimal control histories are then examined. In addition, the possibility of implementing other constraints is also discussed.

This dissertation is dedicated to my parents, who fully understood the value of an education, supported me in everything I did, and encouraged me to never stop learning.

Acknowledgments

The author would like to acknowledge the support of the Naval Surface Warfare Center in Dahlgren, Virginia for the time and money invested in the pursuit of this degree.

I would also like to thank Dr. John Lawton for his guidance, advice, expertise and patience during the long road traveled to arrive at this point. It has been a genuine pleasure to work with him.

Contents

List of Figures

List of Tables

Chapter 1

Preliminaries

1.1 Ballistics

In general, the term *ballistics* refers to the study of objects flying under the influence of gravity and without any sustaining thrust. Typically a *ballistic* object has a motivating force applied to it initially (e.g., a bullet fired from a gun or a rock from a sling-shot); after launch, only aerodynamic forces and gravity are shaping the flight path of the object. In some cases, an object can transition from powered to ballistic flight; long-range missiles and rockets are examples of vehicles that fall into this category. Regardless of the initial propulsive force, all ballistic objects can be studied using the same general methodology.

The study of ballistics, however, also refers to the examination of the trajectories of ballistic objects with some preferred outcome in mind. For centuries, whether on land or at sea, the general use of ballistic weapons (i.e., cannons or hand-held weapons) revolved around the notion of inflicting heavy damage at relatively short range without a great deal of accuracy involved. This led to the strategy of employing a large number of weapons fired en masse to

achieve a broad swath of damage. While often effective, this approach generally resulted in heavy expenditures of ammunition or gunpowder and the potential for significant collateral damage. Attempts at hitting specific targets involved a trial-and-error approach for variables (aiming in both vertical and horizontal directions, amount of powder and type of projectile used) which also resulted in the expenditure of more ammunition and powder than necessary and often proved to be ineffective. In general, there was a fairly universal lack of any scientific or methodical approach to determining the correct combination of the variables mentioned above.

The first serious examination of ballistics from a military perspective was motivated as much by safety as it was by accuracy [1]. On 28 February 1844, a public relations cruise of the first screw-powered warship in naval history, the USS *Princeton*, was being carried out on the Potomac river south of Washington, D.C. The intent of the cruise was to showcase the ship, and it's newly developed, wrought iron gun, known as the "Peacemaker", to a crowd of 350 dignitaries and guests including President Tyler, various senators and congressmen, and several cabinet members. The Peacemaker was fired twice for the guests without incident on the southbound journey; on the return trip it was fired once and disaster struck: the cannon exploded killing a number of government officials, including the Secretary of State and the Secretary of the Navy, and wounding several other people.

A number of errors were responsible for the calamity, all of which could have been avoided. The Peacemaker prototype, the "Oregon", suffered a longitudinal crack during its first test firing. The barrel was reinforced with iron bands and successfully fired an additional 150 times before the crack widened until the barrel was unusable. This experience led to the belief that wrought-iron guns would merely crack when over-stressed and not explode as was commonly the case with cast iron guns. The Peacemaker was constructed following the same design as the Oregon but with additional material around the breech to prevent cracks.

The Peacemaker was initially tested with an even greater amount of powder (49 pounds) than had caused the Oregon to crack (35 pounds). Subsequently, the Peacemaker was fired with increasingly larger charges and since no damage was detected, it was declared ready for service.

The subsequent investigation revealed that the procurement of both the ship and gun had happened without the involvement of the Navy and as such there was no government regulation or testing of either. The severity of the incident strengthened the Bureau of Ordinance's role in the procurement and testing of guns. As a result, the U.S. Navy initiated an effort to examine naval gunnery from a methodical and scientific perspective. The objectives of this initiative were to develop an understanding of ballistics that would result in improved accuracy, safety and repeatability. A Naval base at Indian Head, Maryland was established to conduct ordinance testing using an open stretch of the Potomac River; ironically this location was only a few miles away from the *Princeton* incident. When the requirement for a longer-range testing capability became apparent, and as the area became more developed, the ordinance testing was relocated to a new base at Dahlgren, Virginia. Up until the early 2000s, every gun that found its way onto a US Navy vessel was proof tested at the Dahlgren facility. Because of this initiative, the use of naval gunnery (and land-based artillery) evolved from a mere blast-and-pray approach at relatively short ranges to the sophisticated gunnery capability present on gunboats and battleships at the beginning of the 1900s[1]. Despite the advent of missiles and other more sophisticated weapons, US Navy vessels still carry some type of cannon or gun.

Despite these advances, gunnery was still at the mercy of forces beyond the control of the gunners, which still often necessitated the use of multiple shots to achieve a desired effect. Although gravity and aerodynamic forces could be modeled and accounted for, the

[1]The last American battleship, the USS *Missouri*, served until being decommissioned in 1992.

variability in these forces (e.g., wind direction and speed, irregular air density, etc.) could still potentially impact the accuracy of a ballistic weapon. As a result, weapon designers began examining ways of controlling a weapon in flight through the use of aerodynamic control surfaces. The addition of this capability, along with guidance and control mechanisms, began with the relatively simplistic (by today's standards) methods utilized by the Germans and their V-2 rockets during World War II. The subsequent advent of computers made it easy to model ballistic systems and make predictions about their effectiveness[2]. This, along with the addition of highly accurate positioning methods (such as the Global Positioning System, or GPS) has led to the development of gun-launched projectiles with aerodynamic control surfaces and guidance systems that can achieve levels of accuracy unheard of a hundred years ago. For this research, we will be investigating the optimal control of a gun-launched guided projectile controlled via aerodynamic surfaces.

With the advent of very long range guns, such as the Navy's electromagnetic railgun (RG), the need is even greater for a guided projectile control system capable of achieving high levels of accuracy. At long ranges (i.e., hundreds of miles), even small launch errors or deflections caused by wind or rain could result in large miss distances. RG projectiles are generally small ($\sim$ 1 m long) compared with shells fired from World War II battleships (each shell weighing up to 2200 lbs.). In addition to accuracy, control systems are needed to achieve some final result (such as impact angle) at the target, or for maximizing velocity. Another consideration is the potential for the projectile to fly out of the appreciable atmosphere and therefore lose control capability. As a result, the control system must be designed in such a way as to optimally address these often competing considerations. This research examines

[2]Because of the significant mathematical calculations required for Naval gunnery and ballistics, the Naval base at Dahlgren, Virginia became involved in the early development of computers. These computers were extremely large and contained numerous mechanical relays. During one failed operation, a physical inspection of the computer resulted in the discovery of a moth caught in one of the relays. As a result, the term computer "bug" became a part of the software development lexicon [2].

this type of optimal control capability.

1.2 Ballistic Modeling

Ballistic modeling involves the simulation (usually digitally based) of a projectile as it travels through its environment. A number of simplifying assumptions can be made which reduce the complexity of the problem. For example, one could model the projectile as a point mass with an initial velocity, v_0, imparted by the firing mechanism. The assumptions of no atmosphere and constant gravitational acceleration[3], g, result in relatively simple equations of motion. Figure 1.1 provides a description of the problem. The projectile is launched with an initial velocity at a launch angle of β_0 (measured from the horizontal). If we divide the problem into horizontal and vertical components (x and y, respectively) we can write the positions, velocities, and accelerations as:

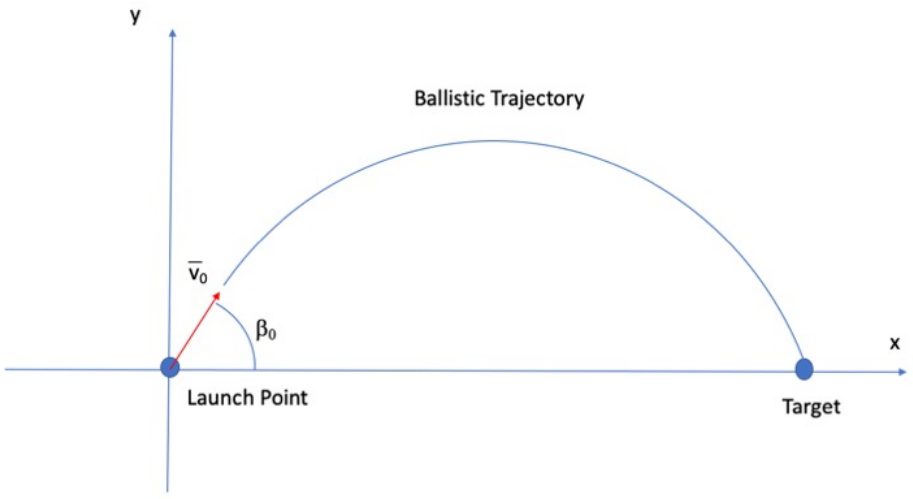

Figure 1.1: Notional Ballistic Trajectory

[3]Constant g is a fair approximation when operating near the Earth's surface.

$$x = x_0 + v_0 \cos \beta_0 t \tag{1.1}$$

$$y = y_0 + v_0 \sin \beta_0 t - \frac{1}{2} g t^2 \tag{1.2}$$

$$v_x = v_0 \cos \beta_0 \tag{1.3}$$

$$v_y = v_0 \sin \beta_0 - g t \tag{1.4}$$

$$a_x = 0 \tag{1.5}$$

$$a_y = -g \tag{1.6}$$

where x_0 and y_0 are the initial x and y positions, respectively, and t is the time since launch. If the projectile is fired from the Earth's surface, x_0 and y_0 can be assumed to be zero.

From here, a variety of complexity can be added to the modeling of the problem. For higher altitudes, the constant g assumption becomes less applicable and can be replaced by inverse square gravity:

$$\boldsymbol{g} = -\frac{\mu_e \mathbf{r}}{|\mathbf{r}|^3} \tag{1.7}$$

where μ_e is the earth's gravitational parameter and $\mathbf{r}$ is the radius vector from the center

of the earth to the ballistic object[4]. This is sufficient for applications where the atmosphere is negligible or nonexistent, which is appropriate for the exoatmospheric portion of a long-range ballistic missile's flight or for spacecraft and satellites. For applications where higher accuracy is required, (e.g., for the orbital determination of the GPS satellites), additional terms that reflect the non-uniform composition and non-spherical shape of the earth are added. For example, the National Geospatial-Intelligence Agency, which is responsible for the operation of the GPS satellite constellation, utilizes the World Geodetic System 84 [3] coordinate system and gravitational models in the determination of the satellite orbits and clocks and the precise location of the numerous ground stations required for the system.

If the problem dictates the representation of an atmosphere, the projectile can be modeled such that aerodynamic forces can be added to the acceleration terms. Atmospheric modeling varies, but usually takes the form of lookup tables for items such as air density, pressure and temperature (and often the gradients of these parameters) depending on altitude. Tables may contain actual data which is then interpolated; alternatively, atmospheric models may contain polynomial fits of conditions.

Depending on the analytical requirements, it may be sufficient to treat the projectile as a point mass and utilize axial and normal force[5] coefficients to represent the aerodynamic forces. For something as intricate as a detailed design of a vehicle, a three-dimensional representation may be necessary. Depending on the demands of the problem, we can move away from a purely ballistic model to one in which the projectile has movable control surfaces for aerodynamic control or reaction control jets for exoatmospheric control (or both for vehicles that operate both in and outside of the atmosphere). Phillips, et al [4, 5], Lin, et al [6], Ohlmeyer, et al [7-9], Pamadi, et al [10], and others have extensively researched the

[4]The use of bold text refers to vector or matrix quantities.

[5]In the study of projectiles, the convention is to use axial and normal forces for the aerodynamic forces as opposed to lift and drag, which some readers will be more familiar with. The relationship among these will be discussed more in section 2.4.

modeling of different control mechanisms for guided projectiles, missiles, and rockets. With added complexity, it becomes more difficult to integrate the equations of motion analytically and numerical integration techniques must be invoked.

For the purposes of this research, it is sufficient to represent the projectile as a point mass while using axial and normal force coefficients to account for the aerodynamic forces. The projectiles of interest have an arrangement of tail fins for pitch, roll and yaw control; they do not have lifting surfaces per se, but a normal force coefficient is typically included since the body produces lift depending on the angle of attack. Because of the very high speeds (Mach numbers of 3-7) at which these projectiles are fired, they are typically very long and tapered, this shape being conducive to high-speed flight (see Figure 1.2). The atmosphere is represented using a subroutine that uses a polynomial fit consistent with the 1962 Standard Atmosphere model. Winds, air turbulence, rotating atmosphere, and weather are not explicitly modeled in this research. These perturbations do not significantly impact the results and are typically compensated for by the closed-loop guidance scheme used by the actual system. This research would form the basis for the development of the closed-loop scheme.

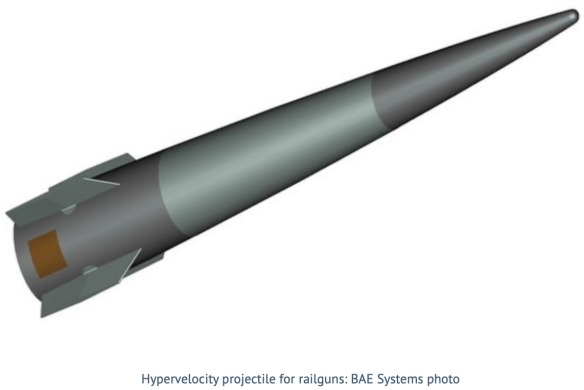

Figure 1.2: BAE Systems Hypervelocity Projectile

It is appropriate to have a brief discussion of modeling in general. It is a common misconception that the more detailed a model is the higher the fidelity of the results that will be obtained. This is not always, (in fact not usually), the case. Detail speaks to the complexity of the model while fidelity refers to the accuracy of the results when compared to the real world. For example, an analyst might be asked to develop a model of a pair of dice for use in a role-playing game. The ambitious engineer could do significant research and build a dynamic six-degree-of-freedom representation of each die and provide parameters that describe the force of the throw, the friction between the die and rolling surface, etc. Or, one could design a model which contains two random number generators that produce a set of values between 1 and 6. For almost all applications, (other than perhaps a PhD-level exam question on the dynamics of dice), the latter model is perfectly suitable and will require much less development and execution time. Not only that, but the more detailed model may contain very subtle inconsistencies or errors that may result in *less* fidelity than the simpler model.[6]

The same applies to the representation of aerospace vehicles. Models can be extremely detailed (including representations of individual components, wiring and fasteners) or may be relatively simple, using a point mass to represent the vehicle and applying the appropriate forces and dynamics. Can the vehicle's response to control surface movement be accurately approximated using coefficients of the aerodynamic forces, (or look-up tables), or will it be necessary to model the actual control surfaces as part of the representation? Excessive detail will result in unnecessary additional computer run time and may hide subtle nuances or errors as mentioned previously.

The morale of the model story is this: it is vitally important to understand the requirements for the model before building it. What level of detail is dictated by the analytical needs? Will more complexity be of any benefit? Can the same level of accuracy result from a

[6]The author has over 35 years of experience conducting model-based analyses for the U. S. Navy and nearly almost that amount of time arguing about model detail vs. fidelity.

simpler representation? Time spent answering questions such as these with an eye toward the intended use of the model will save significant time in the end.

1.3 Introduction to Control Theory

The ability to control a projectile is based on the ability to produce guidance commands, to execute those commands, and then to determine the difference or error between the desired commands and the actual execution of those commands. Although there are numerous ways of achieving each of these three functions, they are all essential for achieving a successful impact of the target by a projectile.

If the target is a fixed land site, its position can be programmed into the projectile prior to launch and the guidance commands can be computed on board. A GPS receiver can be used to determine the position of the projectile and guidance commands generated; GPS information can also be used to assess how well the guidance commands are being executed.

For projectiles intended to impact airborne or moving land targets, a sensor is needed to provide updates on the target's position. This information can be provided by off-board sensors (such as a radar or optical system located near the launcher), by an on-board sensor (such as a short-range radar, infrared or optical sensor) or by a combination of the two. Often a system will have an off-board sensor to provide updates through the majority of the projectile's flight and then an on-board sensor to provide updates during the last few seconds of flight. Off-board sensors usually track the projectile and the target and develop guidance commands accordingly; the commands are then broadcast (or *uplinked*) to the projectile (often using the same tracking sensor as the broadcast mechanism) to be executed. Alternatively, the position and velocity of both the projectile and the target can be uplinked and the guidance command generation done on-board the projectile. Projectiles with on-

board sensors may use GPS position data and the sensor's information to develop guidance commands and to determine their effectiveness. Regardless of the mechanism, some form of control system is used to provide and execute the guidance commands to achieve an intercept of the target. To develop and prove various methods of control, a model of the system is necessary.

Kirk [11] indicates the objective of control theory is "*to determine the control signals that will cause a process to satisfy the physical constraints and at the same time minimize (or maximize) some performance criterion"* (emphasis in the original). In other words, the intent is to influence the system's behavior in order to minimize some performance criterion (also known as the *cost*). Alternately, we can find a maximum for the performance measure by looking for the negative of the minimum, with no loss in generality. In the study of ballistics, there are many different potential performance criteria. Time of flight is perhaps one of the most common; it is often advantageous to impact a target as quickly as possible to allow follow-on shots. Accuracy or miss distance are also common and often depend on warhead size or blast radius. A measure of the damage caused, also known as the probability of kill, or p_k, is also often used. Many times, several of these criteria will be used in combination; it is also common to have a ranking of criteria (i.e., p_k followed by time of flight, etc.). For projectiles for which the main damage mechanism is kinetic energy, the impact velocity, which is directly tied to the p_k, is a good performance measure.

Thus, the problem becomes one of solving for a control history (i.e., the control as a function of time) that does not violate the physical constraints of the problem (the equations of motion and the initial and final conditions) but still minimizes (or maximizes) the value of the performance criteria. Such a control history is known as the optimal control.

1.4 Optimal Control Methods

In the development of optimal control solutions, there are two general approaches: the direct method and the indirect method. In the direct method, the control is directly manipulated (hence the name) in an iterative manner until an optimal result is achieved; this is accomplished by measuring the impact of the candidate control on the performance measure and then manipulating the control until a minimum (or maximum) of the performance measure is found.

In the indirect method, the iteration is applied to the costate or adjoint variables, on which the controls are dependent, (through the necessary conditions of the Maximum Principle [12]), in an effort to achieve an optimal control.

Both the direct and indirect methods have strengths and weaknesses, and a significant body of knowledge exists for both methods. The direct method is generally more straightforward and can provide quick results provided that the initial guess for the optimal solution is sufficiently close to allow convergence. However, direct-method approaches usually result in approximate optimal solutions. A significant body of work by Betts [13], and others [14, 15] focuses on variations on the direct approach involving collocation.

Indirect methods, on the other hand, provide very accurate solutions. Benson, Huntington, Thorvaldsen, and Rao [16] indicate that "The primary advantages of indirect methods are their high accuracy and the assurance that the solution satisfies first-order optimality conditions."

With regards to the indirect method, Betts [17], quoting Bryson and Ho [18], states "The main difficulty with these methods is getting started..." One of the difficulties to be addressed in using the indirect method for the solution of an optimal control problem is the necessity of obtaining estimates of the initial costates such that convergence to a solution is feasible

and efficient. This primary difficulty lies in the fact that the arrival at an optimal solution is dependent on the selection of the initial costates, which unfortunately is not usually intuitive. Saghamanesh and Baoyin [19] state that "the optimal control solution of (sic) indirect method is exclusively sensitive to the initial guess of the Lagrange multipliers." Bushong and Lawton [20] develop estimates of the costates for an optimal trajectory by using a near-optimal parameterized solution. Jiang, Tang and Li [21] use a nominal trajectory to estimate the adjoint variables for a low-thrust application. Seywald and Kumar [22] discuss a method to estimate the costates for an optimal control problem using the indirect method using results obtained from a related discretized, direct-method solution. Lee and Bang [23] propose an initial guess structure for the costates based on known costate characteristics for the design of spiral orbit transfers. Yan and Wu [24] use a first-order Taylor series expansion of the adjoint variables and investigate an adjoint variable guess technique. Miswanto, Pranoto, Muhammad, and Mahayana [25] assume initial values for the costates and then iteratively use the steepest gradient method to update the initial values. Farhoo and Ross [26] and Gong, Ross, and Farhoo [27] use pseudospectral methods to estimate the costates. Gath and Well [28] and Bulirsch, Nerz, and von Stryk [29] use a combination of the direct and indirect method to obtain costate estimates. Martell and Lawton [30] provide a methodology which allows for estimation of the costates using finite differences obtained from a known trajectory and implemented through the use of an auxiliary optimization problem.

This research revisits the approach of Martell and Lawton, replacing the auxiliary optimization problem with another auxiliary problem which has an analytic solution. Therefore, no numerical iterations of the initial costate estimates are required for this present approach. The costate solutions from this new approach, which were derived using a reference trajectory, were then used in a multiple-shooting method implementation to determine the optimal control for the same problem. The method was then applied to problems involving other

reference trajectories in an attempt to determine how "rough" the initial costate estimates could be and still be within the neighborhood to result in a converged solution. Ballistic (i.e., angle of attack set to zero) and manually manipulated trajectories were both examined as reference trajectories.

In an effort to gain insights into another potential method for estimating the initial costates, a backwards integration scheme was developed. While the original methodology used a forward integration scheme to replicate the reference trajectory, this alternate method added a backwards integration to obtain the costates. The resulting initial costates were then compared to those obtained from the original method.

1.5 Dynamic Pressure Constraints

Subsequently, a minimum dynamic pressure constraint was added to the problem. The imposition of a maximum dynamic pressure constraint is quite common in aerospace applications. Limits on maximum dynamic pressure are applied to prevent excessive aero-thermal heating or stresses on an airframe, missile body or rocket structure; this is typically done by intentionally avoiding situations where the limits could be exceeded (e.g., a missile's launch profile may be specified such that it does not violate the maximum dynamic pressure constraint). Examples for exploring or developing optimal controls in the presence of such constraints are abundant and well-documented. Seywald and Cliff [31], Graichen and Petit [32], and Graichen, Kugi, Petit, and Chaplais [33] explore different techniques for addressing the Goddard problem[7] in the presence of a maximum dynamic pressure constraint. Park [34] develops launch vehicle trajectories when a maximum dynamic pressure constraint is present.

[7]The Goddard problem seeks to optimize the peak altitude of a rocket, ascending vertically, and taking into account atmospheric drag and the gravitational field. Rocket thrust is the only control and the problem often involves the imposition of a maximum dynamic pressure constraint.

There are many examples in the literature where the application of a maximum dynamic pressure constraint is discussed and numerous developments of optimal control solutions in the presence of such a constraint.

However, the requirement to impose a *minimum* dynamic pressure constraint is less common and is generally applicable in cases in which a vehicle is controlled solely by utilizing aerodynamic surfaces; such a constraint could be imposed by the designer of such a vehicle to prevent it from flying into a region where low dynamic pressure results in a loss of control. The imposition of such a minimal constraint is not discussed in the literature nor is there any demonstrated method to incorporate one into the development of an optimal control history[8]. With the advent of guns capable of firing projectiles at hypersonic velocities, such as might be the case for an electromagnetic railgun, the situation exists where an aerodynamically-controlled projectile can be launched such that it traverses a region of the upper atmosphere (and beyond) where it has limited, or no, control capability. Imposing a minimum dynamic pressure constraint forces the projectile to remain within its region of control in the atmosphere. The implementation of a path constraint is added to the methodology discussed previously[9]; a process is developed to optimally control a projectile such that it "rides" a user-specified boundary of minimum dynamic pressure. The effect of imposing a variety of minimum dynamic pressure constraints is examined and numerical examples are shown comparing constrained cases against their unconstrained counterparts. The potential application of the methodology to other constraints, such as thermal heating, is also briefly discussed.

[8]Historically, minimum dynamic pressure considerations have been included in the flyout of missile systems; these are generally represented as regions of space that the missile must avoid and not a path constraint as will be discussed as part of this research.

[9]This represents another advantage of indirect methods: it is relatively easy (compared to direct method applications) to add a path constraint to the problem.

1.6 Motivation for Study

There is an extensive body of knowledge pertaining to both direct and indirect methods for determining a set of optimal controls for a system; both have their advantages and disadvantages. In addition, there are numerous examples in the literature for determining estimates for the costates needed to apply an indirect method. However, there is no ideal method available and the application of various techniques is often akin to an art form. There is ample room for another tool in the toolbox that can be used to quickly develop costate estimates, especially a methodology that provides this capability without an excessive amount of foreknowledge or computational overhead.

The development of optimal control solutions in the presence of a maximum dynamic pressure constraint is well documented. However, the imposition of a minimum dynamic pressure path constraint has not been explored. This is understandable given the fact that vehicles that are designed to exceed the limits of controllability provided by aerodynamic surfaces are typically also equipped with systems that do not rely on aerodynamic pressure to enable control.[10] With the advent of very high velocity guns like the Navy's electromagnetic railgun and the Army's Long-Range Cannon, and the inevitable increase in long-range capability, it is feasible that projectiles will be fired with the potential to exceed their aerodynamic control capability. Since it would be prohibitively complex and expensive to add redundant control systems to a guided projectile, the ability to impose a minimum dynamic pressure constraint, and to subsequently develop optimal control solutions in the presence of that constraint is highly desirable.

[10]NASA's X-15 rocket plane, which flew 199 missions in the late 1950s and most of the 1960s, was equipped with a complimentary set of aerodynamic control surfaces and reaction control jets

1.7 Outline of Remaining Chapters

Chapter 2 begins with an overview of the optimal control problem of interest and continues with the development of a methodology that uses a reference or "in-the-neighborhood" trajectory to estimate the initial costates for an optimal control solution. Subsequently, a gun-launched, guided projectile is used as an illustrative example and the methodology is applied to develop expressions for the control. The chapter includes a discussion of the multiple-shooting method, which is used to reset the costates during the integration of the states and costates. A solver is employed to produce a solution to a nonlinear system of equations which results in an optimized trajectory and corresponding optimal control history.

Chapter 3 discusses the application of the methodology developed in Chapter 2. Initially, a near-optimal trajectory from a direct method program is used as the reference trajectory and the resulting fully-optimal trajectory is discussed along with a comparison of the resulting optimal costate histories and the initial estimates. Then, a ballistic trajectory is used as the reference trajectory and an optimal trajectory is again obtained. Subsequently, a hand-manipulated range-extension trajectory is used as the reference and an optimal trajectory obtained. The chapter also includes an examination of range and time-of-flight extensions. An alternate method of obtaining the costate estimates is also presented.

Chapter 4 begins with a discussion of the minimum dynamic pressure path constraint and a derivation of an expression for the control when on the constraint. A brief examination of the possibility of a point constraint is discussed. The multiple-shooting method is modified to account for the constraint; an optimal unconstrained trajectory is used as the reference trajectory and an optimal trajectory is produced in the presence of a constraint. The chapter then includes results for cases with varying minimum dynamic pressure constraints. A brief

discussion of the application to other constraints is also included.

Chapter 5 contains a general discussion of the chronology of the research including the initial motivation for the work, the development of computer programs in both MATLAB and FORTRAN, a comparison with the direct method and the sensitivity of the convergence or non-convergence of a case depending on the costates. Methods to increase the likelihood of convergence are also discussed as is the inability to predict whether a case will converge ahead of time. Areas for follow-on research are also identified.

Conclusions drawn from this work are contained in Chapter 6 as is a brief recap of the research.

Chapter 7 contains a list of references cited in the dissertation.

Appendix A contains the derivation and discussion of a unique minimum induced-drag heading error guidance scheme.

Appendix B provides a listing of all of the computer code developed in this research to execute the unconstrained optimal control cases shown in Chapter 3. Appendix C contains the corresponding computer code for the minimum dynamic pressure constraint. All computer code listed is in FORTRAN.

1.8 Contributions

Following is a list of original contributions presented in this dissertation:

- A methodology was developed for estimating the initial costates for an indirect method for an optimal control problem. The approach utilizes a reference trajectory to analytically determine the initial costate estimates.

- The methodology was applied to an optimal control problem of a gun-launched, guided projectile which used angle of attack as the sole control. An expression for the angle of attack was developed which was used to fly the projectile to optimize the final velocity. A unique variation of the multiple-shooting method was employed in which only the costates (and not the states) were updated along the trajectory. The costate estimates were used to initialize the process. A near-optimal trajectory produced from a direct-method application was used as a reference trajectory and the methodology was used to produce an optimal trajectory. Ballistic and hand-modified trajectories were then used to produce optimal trajectories. For all of these trajectory examples, the initial costate estimates were compared to their final values.

- The methodology was extended to include a minimum dynamic pressure constraint by examining the application of both path and point constraints. An expression for the angle of attack while traveling on the constraint was developed. The multiple-shooting method application was updated to reflect the addition of the constraint. Multiple examples of constrained trajectories were analyzed and discussed.

- A unique scheme for minimum induced-drag heading error guidance was developed. This scheme is suitable for addressing out-of-plane errors that may arise during a planar engagement of a target.

Chapter 2

Methodology

2.1 Overview

The focus of this research is on finding estimates of costates, also known as adjoint variables, of the indirect method of solution for a certain class of optimal control problems. A general methodology will be developed and then applied to the problem of optimally controlling a gun-launched guided projectile with the objective of maximizing impact (final) velocity. The approach taken involves using a reference trajectory to calculate estimates of the initial costates for the optimization problem. An expression for the optimal control in terms of the costates is developed and used to "fly" the projectile. Iteration of the costate estimates occurs until a converged solution is achieved.

Bryson and Ho [35], Kirk [36], Leitmann [37], and Lee and Markus [38] provide good descriptions and derivations of the optimal control problem. The text by Bryson and Ho, although relatively old, still provides an excellent reference for optimal control work.

2.2 The Optimal-Control Problem Statement

We begin with the differential equations of the state:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \mathbf{u}, t) \tag{2.1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the state vector, $\mathbf{u} \in \mathbb{R}^m$ is the control vector, and t is time.

The performance measure or cost to be minimized is:

$$J(\mathbf{u}) = \phi(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} L(\boldsymbol{x}, \mathbf{u}, t)\mathrm{d}t \tag{2.2}$$

where $\phi(\boldsymbol{x}(t_f), t_f)$ is the terminal cost function, L is the intermediate cost function, and $[t_0, t_f]$ is the time interval of interest. The initial conditions on the state are given by $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$.

The objective of the optimal control problem is to find the control-vector function of time, $\mathbf{u}(t)$, that minimizes the cost function, J, while satisfying the differential equations of the state and the initial conditions. The problem, posed in this manner, is known as the Bolza problem. If the intermediate cost function is zero, the problem is referred to as the Mayer problem; if the terminal cost function is zero, then the problem is referred to as the Lagrange problem.

To find an expression for the costate dynamics, we construct an augmented performance function by adjoining the dynamics to the performance index using a vector of Lagrange multipliers[1], $\boldsymbol{\lambda} \in \mathbb{R}^n$:

[1]The $\boldsymbol{\lambda}$'s are known as the adjoint or costate variables. These terms will be used interchangeably in this dissertation.

$$\bar{J}(\mathbf{u}) = \phi(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} L(\boldsymbol{x}, \mathbf{u}, t) - \boldsymbol{\lambda}^T(t)[\boldsymbol{f}(\boldsymbol{x}, \mathbf{u}, t) - \dot{\boldsymbol{x}}]\mathrm{d}t \tag{2.3}$$

We define the Hamiltonian, $\mathcal{H}$:

$$\mathcal{H}(\mathbf{x}, \mathbf{u}, t) = \boldsymbol{\lambda}(t)^T \boldsymbol{f}((\mathbf{x}, \mathbf{u}, t) - L((\mathbf{x}, \mathbf{u}, t) \tag{2.4}$$

which we can use to rewrite the augmented performance function as:

$$\bar{J}(\mathbf{u}) = \phi(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} [-\mathcal{H}(\mathbf{x}, \mathbf{u}, t) + \boldsymbol{\lambda}^T(t)\dot{\boldsymbol{x}}]\mathrm{d}t \tag{2.5}$$

We focus our attention on the $\dot{\mathbf{x}}$ term in the integral in Equation (2.5). Using integration by parts, we obtain:

$$\int_{t_0}^{t_f} \boldsymbol{\lambda}^T(t)\dot{\boldsymbol{x}}\mathrm{dt} = \boldsymbol{\lambda}^T(t)\boldsymbol{x}\Big|_{t_0}^{t_f} - \int_{t_0}^{t_f} \dot{\boldsymbol{\lambda}}^T(t)\boldsymbol{x}\mathrm{dt} \tag{2.6}$$

Inserting this into (2.5) results in:

$$\bar{J}(\mathbf{u}) = \phi(\boldsymbol{x}(t_f), t_f) + \boldsymbol{\lambda}^T(t)\boldsymbol{x}\Big|_{t_0}^{t_f} - \int_{t_0}^{t_f} [\mathcal{H}(\mathbf{x}, \mathbf{u}, t) + \dot{\boldsymbol{\lambda}}^T(t)\boldsymbol{x}]\mathrm{d}t \tag{2.7}$$

We now consider an infinitesimal change in the control, denoted $\delta\mathbf{u}(t)$, which results in a corresponding change in the state of $\delta\mathbf{x}(t)$ and in the augmented performance index of $\delta\bar{J}$ (assuming that t_0 and t_f are fixed):

$$\delta \bar{J} = \left[\left(\frac{\partial \phi}{\partial \mathbf{x}} + \boldsymbol{\lambda}^T \right) \delta \mathbf{x} \right]_{t=t_f} - \left[\boldsymbol{\lambda}^T \delta \mathbf{x} \right]_{t=t_0} + \int_{t_0}^{t_f} \left[- \left(\frac{\partial \mathcal{H}}{\partial \mathbf{x}} + \dot{\boldsymbol{\lambda}}^T \right) \delta \mathbf{x} + \left(- \frac{\partial \mathcal{H}}{\partial \mathbf{u}} \delta \mathbf{u} \right) \right] \mathrm{d}t \quad (2.8)$$

Just as in regular calculus, where the gradient must be 0 to obtain a minimum, so must the variation, $\delta \bar{J}$, be equal to 0 per the calculus of variations [39]; this is a necessary condition for optimality. We observe that if the state is fixed at t_0, (i.e., $\mathbf{x}(t_0) = \mathbf{x}_0$) then $\delta \mathbf{x}(t_0) = \mathbf{0}$. For $\delta \bar{J}$ to be equal to 0, the coefficients of the $\delta \boldsymbol{x}(\mathrm{t})$, $\delta \boldsymbol{x}(t_f)$, and $\delta \mathbf{u}$ terms must also be zero. This leads to an expression for the costate dynamics:

$$\dot{\boldsymbol{\lambda}}(t) = - \left(\frac{\partial \mathcal{H}}{\partial \boldsymbol{x}} \right)^T = - \left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} \right)^T \boldsymbol{\lambda} + \left(\frac{\partial L}{\partial \boldsymbol{x}} \right)^T \quad (2.9)$$

The costates at the final time are:

$$\boldsymbol{\lambda}^T(t_f) = - \left(\frac{\partial \phi}{\partial \boldsymbol{x}} \right) \Bigg|_{t=t_f} \quad (2.10)$$

And the partial derivative of the Hamiltonian with respect to the control is:

$$\frac{\partial \mathcal{H}}{\partial \mathbf{u}} = 0 \quad (2.11)$$

Given these definitions, Pontryagin's Maximum Principle [12] states that the optimal $\mathbf{u}$ will satisfy the necessary condition,

$$\mathbf{u}^* = \arg \max_{\mathbf{u}} \mathcal{H}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \mathbf{u}, t) \quad (2.12)$$

where $\boldsymbol{x}^*(t)$ is the value of the state on the optimal trajectory at time t. In other words at each time t on the optimal trajectory, the optimal control is the one that maximizes the Hamiltonian. Because (2.12) is a necessary condition, the optimal control obtained results in a local extremum of the Hamiltonian; we do not have any guarantee that it is a global one.

The class of problems to be solved using the technique discussed later in this dissertation is such that $\frac{\partial \mathcal{H}}{\partial \mathbf{u}} = 0$ is necessary throughout the whole trajectory. This is a large class of problems, to which many aerospace control problems belong.

If there is a terminal constraint on the state of the form:

$$\boldsymbol{\psi}_f[\boldsymbol{x}(t_f), t_f] = \mathbf{0} \tag{2.13}$$

we adjoin this along with the dynamics to the performance index (2.3) using the Lagrange multipliers $\boldsymbol{\nu}$ and $\boldsymbol{\lambda}(t)$, respectively to get:

$$\bar{J}(\mathbf{u}) = \phi(\boldsymbol{x}(t_f), t_f) - \boldsymbol{\psi}^T(\boldsymbol{x}(t_f), t_f)\boldsymbol{\nu} + \int_{t_0}^{t_f} L(\boldsymbol{x}, \mathbf{u}, t) - \boldsymbol{\lambda}^T(t)[\boldsymbol{f}(\boldsymbol{x}, \mathbf{u}, t) - \dot{\boldsymbol{x}}]\mathrm{d}t \tag{2.14}$$

Substituting the Hamiltonian, integrating by parts, and considering an infinitesimal deviation on the final time we arrive at two additional conditions. The costates at the final time are given by:

$$\boldsymbol{\lambda}(t_f) = \left[-\frac{\partial \phi}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x}}^T \boldsymbol{\nu} \right] \Bigg|_{t=t_f} \tag{2.15}$$

and the Hamiltonian at t_f (if the final time is free):

$$\mathcal{H}(t_f) = \left[\frac{\partial \phi}{\partial t} - \frac{\partial \boldsymbol{\psi}_f}{\partial t}^T \boldsymbol{\nu}\right]\Bigg|_{t=t_f} \tag{2.16}$$

If the final time is fixed, then Equation (2.16) does not apply.

To summarize, the general solution process requires the appropriate number of costate boundary conditions (also known as tranversality conditions) and state boundary conditions given by:

$$\boldsymbol{\lambda}(t_0) = \frac{\partial \boldsymbol{\psi}_0^T(x)}{\partial \boldsymbol{x}} \boldsymbol{\nu}_0 \tag{2.17}$$

$$\boldsymbol{\lambda}(t_f) = \left[-\frac{\partial \phi}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x}}^T \boldsymbol{\nu}\right]\Bigg|_{t=t_f} \tag{2.18}$$

$$\boldsymbol{\psi}_0[\boldsymbol{x}(t_0), t_0] = \mathbf{0} \tag{2.19}$$

$$\boldsymbol{\psi}_f[\boldsymbol{x}(t_f), t_f] = \mathbf{0} \tag{2.20}$$

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0 \tag{2.21}$$

Additionally, if the final time is free, then the optimal t_f will be such that

$$\mathcal{H}(t_f) = \left[\frac{\partial \phi}{\partial t} - \frac{\partial \boldsymbol{\psi}_f}{\partial t}^T \boldsymbol{\nu}\right]\Bigg|_{t=t_f} \tag{2.22}$$

These equations represent the classical approach to optimal control theory; a good derivation of this approach may be found in [37]. The indirect method, in this case, has reduced the optimal control problem to the task of finding the parameters $\boldsymbol{\lambda}(t_0)$, $\boldsymbol{\nu}$, $\boldsymbol{\nu}_0$, and t_f such that the conditions (2.17) - (2.22) are satisfied. The multiple-shooting method [40, 41] is one of the more successful methods for solving boundary-value problems of this type.

With an initial estimate for the costates, $\boldsymbol{\lambda}_0$, and a guess for $\boldsymbol{\nu}_0$, one could numerically integrate the differential equations for the state and costates from time $t = 0$ forward to the estimate for t_f. This method, known as the simple-shooting method, suffers from the shortfall that small errors in the initial estimates will lead to uncontrollable growth of the costates as the integration progresses in time. An alternative to this approach, which specifically address this issue, is the multiple-shooting method. In this method, the trajectory is divided up into a number of nodes, and estimates for the costates are generated at each of the nodes. The integration proceeds as before, but now upon arriving at each node, the costates are "reset" to the estimates previously calculated for that node. In this manner, the number of nodes may be selected such that the costates do not diverge uncontrollably and the method converges to a solution.

2.3 Method for Estimating the Costates from In-the-Neighborhood Trajectories

We will utilize the finite difference method used in [30]. To begin, the adjoint variable differential equations are written as:

$$\dot{\boldsymbol{\lambda}} = [\boldsymbol{A}(\tilde{\boldsymbol{x}}, \tilde{\mathbf{u}}, t)]\boldsymbol{\lambda} + \frac{\partial L}{\partial \boldsymbol{x}}\Big|_{\tilde{\boldsymbol{x}}, \tilde{\mathbf{u}}, t} \tag{2.23}$$

where

$$\boldsymbol{A} = -\Big(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\Big)^T \tag{2.24}$$

and the tildas ($\sim$) denote quantities from the reference trajectory.

If we divide the the timeline of the problem into w nodes, with the first node at $t = 0$ and the final node at $t = t_f$, Equation (2.23) can then be approximated by:

$$[\boldsymbol{I}]\boldsymbol{\lambda}_{i+1} - [\boldsymbol{I} + \boldsymbol{A}_i(\tilde{\boldsymbol{x}}_i, \tilde{\mathbf{u}}_i, t)\Delta t_i]\tilde{\boldsymbol{\lambda}}_i = \Delta t_i \frac{\partial L}{\partial \boldsymbol{x}}\Big|_{\tilde{\boldsymbol{x}}_i, \tilde{\mathbf{u}}_i, t_i}; \qquad i = 1, ..., w-1 \tag{2.25}$$

where $\boldsymbol{I}$ is the identity matrix. There are no hard and fast criteria for selecting w other than the desire to balance accuracy (a large w) vs. computer run time. The selection of w is discussed more in Chapter 5.

The time between nodes, which does not need to be the same throughout the problem, is given by:

$$\Delta t_i = t_{i+1} - t_i \tag{2.26}$$

In matrix notation, Equation (2.25) becomes:

$$\begin{bmatrix} \boldsymbol{\alpha}_1 & \boldsymbol{I} & \mathbf{0} & ... & ... & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\alpha}_2 & \boldsymbol{I} & ... & ... & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & ... & \boldsymbol{\alpha}_{w-2} & \boldsymbol{I} & : \\ : & : & ... & ... & \boldsymbol{\alpha}_{w-1} & \boldsymbol{I} \\ \mathbf{0} & \mathbf{0} & ... & ... & \mathbf{0} & \boldsymbol{I} \end{bmatrix} \begin{Bmatrix} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \\ : \\ \boldsymbol{\lambda}_{w-1} \\ \boldsymbol{\lambda}_w \end{Bmatrix} = \begin{Bmatrix} \Delta t_1 \dfrac{\partial L}{\partial \boldsymbol{x}}\Big|_1 \\ \Delta t_2 \dfrac{\partial L}{\partial \boldsymbol{x}}\Big|_2 \\ : \\ \Delta t_{w-1} \dfrac{\partial L}{\partial \boldsymbol{x}}\Big|_{w-1} \\ \boldsymbol{\lambda}(t_f) \end{Bmatrix} \tag{2.27}$$

where each $n \times n$ submatrix $\boldsymbol{\alpha}_i$ is given by:

$$\boldsymbol{\alpha}_i = -(\boldsymbol{I} + \boldsymbol{A}_i \Delta t_i) \tag{2.28}$$

(The reader should note that these matrix quantities, $\boldsymbol{\alpha}_i$, are not related to the angle of attack.)

Inverting the matrix on the left side of Equation (2.27) and multiplying to both sides yields:

$$\begin{Bmatrix} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \\ : \\ : \\ \boldsymbol{\lambda}_{w-1} \\ \boldsymbol{\lambda}_w \end{Bmatrix} = \begin{bmatrix} \boldsymbol{\alpha}_1^{-1} & -\boldsymbol{\alpha}_1^{-1}\boldsymbol{\alpha}_2^{-1} & \boldsymbol{\alpha}_1^{-1}\boldsymbol{\alpha}_2^{-1}\boldsymbol{\alpha}_3^{-1} & -\boldsymbol{\alpha}_1^{-1}\boldsymbol{\alpha}_2^{-1}\boldsymbol{\alpha}_3^{-1}\boldsymbol{\alpha}_4^{-1} & ... & \pm\boldsymbol{\alpha}_1^{-1}\boldsymbol{\alpha}_2^{-1}...\boldsymbol{\alpha}_{w-1}^{-1} \\ \mathbf{0} & \boldsymbol{\alpha}_2^{-1} & -\boldsymbol{\alpha}_2^{-1}\boldsymbol{\alpha}_3^{-1} & \boldsymbol{\alpha}_2^{-1}\boldsymbol{\alpha}_3^{-1}\boldsymbol{\alpha}_4^{-1} & ... & \mp\boldsymbol{\alpha}_2^{-1}\boldsymbol{\alpha}_3^{-1}...\boldsymbol{\alpha}_{w-1}^{-1} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{\alpha}_3^{-1} & -\boldsymbol{\alpha}_3^{-1}\boldsymbol{\alpha}_4^{-1} & ... & \pm\boldsymbol{\alpha}_3^{-1}\boldsymbol{\alpha}_4^{-1}...\boldsymbol{\alpha}_{w-1}^{-1} \\ : & : & ... & ... & ... & : \\ \mathbf{0} & \mathbf{0} & ... & ... & \boldsymbol{\alpha}_{w-1}^{-1} & -\boldsymbol{\alpha}_{w-1}^{-1} \\ \mathbf{0} & \mathbf{0} & ... & ... & \mathbf{0} & \mathbf{0} \end{bmatrix} \cdot \begin{Bmatrix} \Delta t_1 \dfrac{\partial L}{\partial \boldsymbol{x}}\Big|_1 \\ \Delta t_2 \dfrac{\partial L}{\partial \boldsymbol{x}}\Big|_2 \\ : \\ \Delta t_{w-1} \dfrac{\partial L}{\partial \boldsymbol{x}}\Big|_{w-1} \\ \boldsymbol{\lambda}(t_f) \end{Bmatrix} \tag{2.29}$$

A brief note with regard to the signs of the terms in the last column of the matrix on the right-hand side of Equation (2.29): the $w-1$ term in that column always has a negative sign associated with it. From there, the terms alternate in sign upward (i.e., the $w-2$ term

is positive, the $w-3$ term is negative and so on).

Multiplying the right side of Equation (2.29) results in:

$$\begin{Bmatrix} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \\ : \\ : \\ \boldsymbol{\lambda}_{w-1} \\ \boldsymbol{\lambda}_w \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{\alpha}_1^{-1}\Delta t_1 \frac{\partial L}{\partial \boldsymbol{x}}\Big|_1 - \boldsymbol{\alpha}_1^{-1}\boldsymbol{\alpha}_2^{-1}\Delta t_2 \frac{\partial L}{\partial \boldsymbol{x}}\Big|_2 + ... \pm \boldsymbol{\alpha}_1^{-1}\boldsymbol{\alpha}_2^{-1}...\boldsymbol{\alpha}_{w-1}^{-1}\boldsymbol{\lambda}(t_f) \\ \boldsymbol{\alpha}_2^{-1}\Delta t_2 \frac{\partial L}{\partial \boldsymbol{x}}\Big|_2 - ... \mp \boldsymbol{\alpha}_2^{-1}\boldsymbol{\alpha}_3^{-1}...\boldsymbol{\alpha}_{w-1}^{-1}\boldsymbol{\lambda}(t_f) \\ : \\ : \\ \boldsymbol{\alpha}_{w-1}^{-1}\Delta t_{w-1} \frac{\partial L}{\partial \boldsymbol{x}}\Big|_{w-1} - \boldsymbol{\alpha}_{w-1}^{-1}\boldsymbol{\lambda}(t_f) \\ \boldsymbol{\lambda}(t_f) \end{Bmatrix} \tag{2.30}$$

Given that:

$$\boldsymbol{\lambda}(t_f) = \left(\frac{\partial \phi}{\partial \boldsymbol{x}}^T + \frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x}}^T \boldsymbol{\nu}\right)\Big|_{t=t_f} \tag{2.31}$$

we find that the costates are given by:

$$\begin{Bmatrix} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \\ : \\ : \\ \boldsymbol{\lambda}_{w-1} \\ \boldsymbol{\lambda}_w \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{\alpha}_1^{-1}\Delta t_1 \frac{\partial L}{\partial \boldsymbol{x}}\Big|_1 - ... \mp \boldsymbol{\alpha}_1^{-1}\boldsymbol{\alpha}_2^{-1}...\boldsymbol{\alpha}_{w-2}^{-1}\Delta t_{w-1}\frac{\partial L}{\partial \boldsymbol{x}}\Big|_{w-1} \\ -\boldsymbol{\alpha}_2^{-1}\Delta t_2 \frac{\partial L}{\partial \boldsymbol{x}}\Big|_2 + ... \pm \boldsymbol{\alpha}_2^{-1}\boldsymbol{\alpha}_3^{-1}...\boldsymbol{\alpha}_{w-2}^{-1}\Delta t_{w-1}\frac{\partial L}{\partial \boldsymbol{x}}\Big|_{w-1} \\ : \\ : \\ \boldsymbol{\alpha}_{w-1}^{-1}\Delta t_{w-1}\frac{\partial L}{\partial \boldsymbol{x}}\Big|_{w-1} \\ \mathbf{0} \end{Bmatrix} + \begin{Bmatrix} \pm\boldsymbol{\alpha}_1^{-1}\boldsymbol{\alpha}_2^{-1}...\boldsymbol{\alpha}_{w-1}^{-1} \\ \mp\boldsymbol{\alpha}_2^{-1}\boldsymbol{\alpha}_3^{-1}...\boldsymbol{\alpha}_{w-1}^{-1} \\ : \\ : \\ -\boldsymbol{\alpha}_{w-1}^{-1} \\ \boldsymbol{I} \end{Bmatrix} \left(\frac{\partial \phi}{\partial \boldsymbol{x}}^T + \frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x}}^T \boldsymbol{\nu} \right)\Bigg|_{t=t_f} \tag{2.32}$$

Which can be rewritten as:

$$\begin{Bmatrix} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \\ : \\ : \\ \boldsymbol{\lambda}_{w-1} \\ \boldsymbol{\lambda}_w \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{d}_1 \\ \boldsymbol{d}_2 \\ : \\ : \\ \boldsymbol{d}_{w-1} \\ \mathbf{0} \end{Bmatrix} + \begin{Bmatrix} \boldsymbol{S}_1 \\ \boldsymbol{S}_2 \\ : \\ : \\ \boldsymbol{S}_{w-1} \\ \boldsymbol{I} \end{Bmatrix} \left(\frac{\partial \phi}{\partial \boldsymbol{x}}^T + \frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x}}^T \boldsymbol{\nu} \right)\Bigg|_{t=t_f} \tag{2.33}$$

For simplicity's sake, we make the following notation change:

$$\left(\frac{\partial \phi}{\partial \boldsymbol{x}_f}^T + \frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x}_f}^T \boldsymbol{\nu}\right) \equiv \left(\frac{\partial \phi}{\partial \boldsymbol{x}}^T + \frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x}}^T \boldsymbol{\nu}\right)\Bigg|_{t=t_f} \tag{2.34}$$

and Equation (2.33) becomes:

$$\left\{\begin{array}{c} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \\ : \\ : \\ \boldsymbol{\lambda}_{w-1} \\ \boldsymbol{\lambda}_w \end{array}\right\} = \left\{\begin{array}{c} \boldsymbol{d}_1 + \boldsymbol{S}_1 \frac{\partial \phi}{\partial \boldsymbol{x}_f}^T + \boldsymbol{S}_1 \frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x}_f}^T \boldsymbol{\nu} \\ \boldsymbol{d}_2 + \boldsymbol{S}_2 \frac{\partial \phi}{\partial \boldsymbol{x}_f}^T + \boldsymbol{S}_2 \frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x}_f}^T \boldsymbol{\nu} \\ : \\ : \\ \boldsymbol{d}_{w-1} + \boldsymbol{S}_{w-1} \frac{\partial \phi}{\partial \boldsymbol{x}_f}^T + \boldsymbol{S}_{w-1} \frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x}_f}^T \boldsymbol{\nu} \\ \frac{\partial \phi}{\partial \boldsymbol{x_f}}^T + \frac{\partial \boldsymbol{\psi_f}}{\partial \boldsymbol{x_f}}^T \boldsymbol{\nu} \end{array}\right\} \tag{2.35}$$

Now, if we add the initial condition function, $\boldsymbol{\psi}_0$, at $t = 0$, transversality gives:

$$\boldsymbol{\lambda}_1 = \frac{\partial \boldsymbol{\psi}_0}{\partial \boldsymbol{x}}\Big|^T_{\tilde{\boldsymbol{x}}_1, \tilde{\mathbf{u}}_1, t_1} \boldsymbol{\nu}_0 \tag{2.36}$$

Therefore,

$$\frac{\partial \boldsymbol{\psi}_0}{\partial \boldsymbol{x}}^T \boldsymbol{\nu}_0 = \boldsymbol{d}_1 + \boldsymbol{S}_1 \frac{\partial \phi}{\partial \boldsymbol{x}_f}^T + \boldsymbol{S}_1 \frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x}_f}^T \boldsymbol{\nu} \tag{2.37}$$

where $\boldsymbol{S}_1 = \pm \boldsymbol{\alpha}_1^{-1} \boldsymbol{\alpha}_2^{-1} ... \boldsymbol{\alpha}_{w-1}^{-1}$.

We now need to find an expression for $\boldsymbol{\nu}$. Our motivation is the Maximum Principle, where the optimal control, (for the class of problems that are the focus of this research), will be the one that results in $\frac{\partial \mathcal{H}}{\partial \mathbf{u}} = 0$. By using $\boldsymbol{x}$ and $\mathbf{u}$ at each of the nodes, we seek to find costates that leave $\frac{\partial \mathcal{H}}{\partial \mathbf{u}}$ as close to 0 as possible consistent with these data.

We begin with an alternative to Equation (18) from [30]:

$$\frac{\partial \mathcal{H}}{\partial \mathbf{u}}\Big|_{t_i} = \boldsymbol{\lambda}^T \frac{\partial \boldsymbol{f}(\boldsymbol{x}, \mathbf{u})}{\partial \mathbf{u}}\Big|_{t_i} \tag{2.38}$$

$$= \hat{\boldsymbol{\lambda}}_i(\boldsymbol{\nu})^T \frac{\partial \boldsymbol{f}(\tilde{\boldsymbol{x}}_i, \tilde{\mathbf{u}}_i)}{\partial \mathbf{u}} \tag{2.39}$$

$$= \Big[\boldsymbol{d}_i^T + \frac{\partial \phi}{\partial \boldsymbol{x}_f}\boldsymbol{S}_i^T + \boldsymbol{z}^T \frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x_f}}\boldsymbol{S}_i^T\Big]\boldsymbol{B}_i \tag{2.40}$$

where

$$\boldsymbol{B}_i = \frac{\partial \boldsymbol{f}(\tilde{\boldsymbol{x}}_i, \tilde{\mathbf{u}}_i)}{\partial \mathbf{u}} \tag{2.41}$$

We minimize $Q(\boldsymbol{\nu})$ over $\boldsymbol{\nu}$, where

$$Q = \sum_{i=1}^{w} \frac{1}{2} \frac{\partial \mathcal{H}}{\partial \mathbf{u}} \frac{\partial \mathcal{H}}{\partial \mathbf{u}}^T \tag{2.42}$$

$$Q(\boldsymbol{\nu}) = \frac{1}{2}\sum_{i=1}^{w}\Big[\boldsymbol{d}_i^T + \Big(\frac{\partial \phi}{\partial \boldsymbol{x}_f} + \boldsymbol{\nu}^T \frac{\partial \boldsymbol{\psi}_f}{\partial x_f}\Big)\boldsymbol{S}_i^T\Big]\boldsymbol{B}_i\boldsymbol{B}_i^T\Big[\boldsymbol{d}_i + \boldsymbol{S}_i\Big(\frac{\partial \phi}{\partial \boldsymbol{x}_f}^T + \frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x}_f}^T \boldsymbol{\nu}\Big)\Big] \tag{2.43}$$

Taking the partial of Q with respect to $\boldsymbol{\nu}$ and setting equal to 0 yields:

$$\frac{\partial Q}{\partial \boldsymbol{\nu}} = \sum_{i=1}^{w}\Big[\boldsymbol{d}_i^T + \Big(\frac{\partial \phi}{\partial \boldsymbol{x}_f} + \boldsymbol{\nu}^T \frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{x}_f}\Big)\boldsymbol{S}_i^T\Big]\boldsymbol{B}_i\boldsymbol{B}_i^T\boldsymbol{S}_i\frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x}_f}^T = 0 \tag{2.44}$$

which has the solution:

$$\boldsymbol{\nu} = -\Big[\sum_{i=1}^{w} \boldsymbol{d}_i^T \boldsymbol{B}_i \boldsymbol{B}_i^T \boldsymbol{S}_i \frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x}_f}^T + \frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x}_f}\Big(\sum_{i=1}^{w} \boldsymbol{S}_i^T \boldsymbol{B}_i \boldsymbol{B}_i^T \boldsymbol{S}_i\Big)\frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x}_f}^T\Big]^{-1} \cdot \Big[\frac{\partial \boldsymbol{\psi}_f}{\partial \boldsymbol{x}_f}\Big(\sum_{i=1}^{w} \boldsymbol{S}_i^T \boldsymbol{B}_i \boldsymbol{B}_i^T \boldsymbol{S}_i\Big)\frac{\partial \phi}{\partial \boldsymbol{x}_f}^T\Big] \quad (2.45)$$

Note that the summation, $\sum_{i=1}^{w} \boldsymbol{S}_i^T \boldsymbol{B}_i \boldsymbol{B}_i^T \boldsymbol{S}_i$, is the finite-difference version of the Controllability Grammian. So, although each term of the summation is of rank $m < n$, the sum will be of rank n if the problem is controllable. This ensures that the first term in Equation (2.45) is invertible for problems that make sense (i.e., which are completely controllable). Equation (2.45) replaces the auxiliary optimization procedure of [30].

2.4 Illustrative Example - Guided Projectile

The method discussed above is now applied to the problem of a projectile with a single control, α, the angle of attack, which is defined as the angle between the projectile body x-axis, $\hat{\mathbf{x}}_\mathbf{b}$[2], and the velocity vector, $\mathbf{v}$, as shown in Figure 2.1. $\hat{\mathbf{x}}$ is in the vertical direction at the time of launch and $\hat{\mathbf{y}}$ is the local horizontal at launch in the same plane as the target. Two aerodynamic forces, the axial force, $\mathbf{A}$, and the normal force, $\mathbf{N}$, act on the projectile. $\mathbf{A}$ acts in the negative $\hat{\mathbf{x}}_\mathbf{b}$ direction and $\mathbf{N}$ acts perpendicular to $\hat{\mathbf{x}}_\mathbf{b}$. $\hat{\mathbf{v}}$ and $\hat{\mathbf{u}}$ are unit vectors parallel to and perpendicular to $\mathbf{v}$, respectively, with $\hat{\mathbf{u}}$ in the "up" direction. The pitch angle, θ is measured from the vertical ($\hat{\mathbf{x}}$) to $\hat{\mathbf{x}}_\mathbf{b}$.

The reader may be more accustomed to the use of lift and drag as opposed to axial and normal forces; the latter are the aerodynamic forces commonly used in lieu of lift and drag

[2] A ˆ symbol denotes a unit vector. $\hat{\mathbf{v}}$ is a vector of unit length in the same direction as $\mathbf{v}$.

by engineers of both missiles and guided projectiles. It is easy to convert from **N** and **A** to lift, **L**, and drag, **D**, by using $\mathbf{L} = \mathbf{N}\cos\alpha - \mathbf{A}\sin\alpha$ and $\mathbf{D} = \mathbf{A}\cos\alpha + \mathbf{N}\sin\alpha$.

The projectile is fired from the surface of the earth with an initial pitch angle, θ_0, at an initial velocity magnitude of v_0. We will consider planar motion in a non-rotating, earth-centered coordinate system. The projectile is fired at a target with a fixed location on the earth.

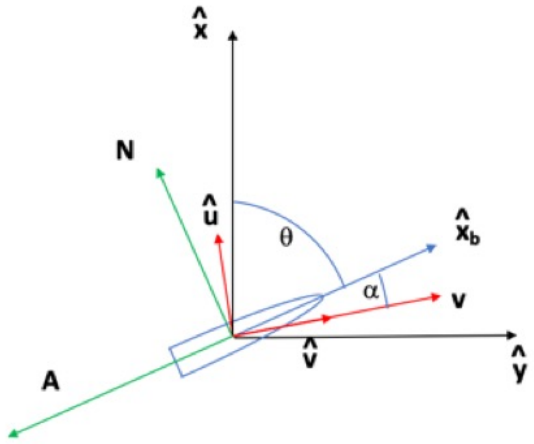

Figure 2.1: Coordinate System Showing Projectile Body Orientation

We define the state as follows:

$$\boldsymbol{x} = \left\{ \begin{array}{c} \mathbf{r} \\ \mathbf{v} \end{array} \right\} \tag{2.46}$$

where **r** is the position of the projectile measured from the center of the earth and **v** is the velocity. The dynamics are defined as:

$$\dot{\boldsymbol{x}} = \left\{ \begin{array}{c} \mathbf{v} \\ \mathbf{a} \end{array} \right\} \tag{2.47}$$

where **a**, the acceleration, is given by:

$$\mathbf{a} = (\mathbf{A} + \mathbf{N})/m + \boldsymbol{g} \tag{2.48}$$

where m is the projectile mass and **g** is the acceleration due to gravity. The magnitude of **A** is given by:

$$\|\mathbf{A}\| = \frac{1}{2}\rho v^2 \mathcal{S} C_A \tag{2.49}$$

where ρ is the density, v is the magnitude of the velocity, $\mathcal{S}$ is the reference area of the projectile and C_A is the constant coefficient of the axial force. The magnitude of **N** is given by:

$$\|\mathbf{N}\| = \frac{1}{2}\rho v^2 \mathcal{S} C_{N_\alpha} \alpha \tag{2.50}$$

where C_{N_α} is the gradient of the coefficient of the normal force with respect to α. Both C_A and C_{N_α} are aerodynamic characteristics of the projectile.

The initial conditions are given by:

$$\boldsymbol{\psi}_0 = \left\{ \begin{array}{c} \mathbf{r}_0 - \mathbf{r}_\mathrm{L} \\ \frac{1}{2}\mathbf{v}_0^T\mathbf{v}_0 - \frac{1}{2}\mathbf{v}_\mathrm{L}^T\mathbf{v}_\mathrm{L} \end{array} \right\} \tag{2.51}$$

where the 0 subscript denotes the initial position and velocity and the L subscript denotes the launch position and velocity. The final conditions are given by:

$$\boldsymbol{\psi}_f = \{\mathbf{r}(t_f) - \mathbf{r}_\mathrm{T}\} \tag{2.52}$$

where t_f denotes the final time and the subscript T denotes the target position.

We define the cost function as:

$$J = -v^2(t_f) \tag{2.53}$$

and we construct the Hamiltonian to yield:

$$\mathcal{H} = \boldsymbol{\lambda}_\mathbf{r}{}^T\mathbf{v} + \boldsymbol{\lambda}_\mathbf{v}{}^T\mathbf{a} \tag{2.54}$$

where $\boldsymbol{\lambda}_\mathbf{r}$ and $\boldsymbol{\lambda}_\mathbf{v}$ are the costates of the position and velocity, respectively. Taking the partial derivative of $\mathcal{H}$ with respect to the control, α, and setting equal to zero gives:

$$\frac{\partial \mathcal{H}}{\partial \alpha} = \boldsymbol{\lambda}_\mathbf{v}^T \frac{\partial \mathbf{a}}{\partial \alpha} = 0 \tag{2.55}$$

To find $\frac{\partial \mathbf{a}}{\partial \alpha}$, we write $\hat{\mathbf{N}}$ and $\hat{\mathbf{A}}$ as:

$$\hat{\mathbf{N}} = \hat{\mathbf{u}}\cos\alpha - \hat{\mathbf{v}}\sin\alpha \tag{2.56}$$

and

$$\hat{\mathbf{A}} = -\hat{\mathbf{u}}\sin\alpha - \hat{\mathbf{v}}\cos\alpha \tag{2.57}$$

where $\hat{\mathbf{v}}$ is the unit vector along the velocity vector and $\hat{\mathbf{u}}$ is perpendicular to $\hat{\mathbf{v}}$ in the "up" direction.

Defining a_N and a_A to be $\frac{\|\mathbf{N}\|}{m}$ and $\frac{\|\mathbf{A}\|}{m}$, respectively, we obtain:

$$\mathbf{a}_{aero} = -a_N \sin\alpha \hat{\mathbf{v}} + a_N \cos\alpha \hat{\mathbf{u}} - a_A \cos\alpha \hat{\mathbf{v}} + a_A \sin\alpha \hat{\mathbf{u}} \tag{2.58}$$

Recognizing that $\frac{\partial \mathbf{g}}{\partial \alpha} = \mathbf{0}$ and therefore $\frac{\partial \mathbf{a}}{\partial \alpha} = \frac{\partial \mathbf{a}_{aero}}{\partial \alpha}$,

$$\begin{aligned}\boldsymbol{\lambda_v}^T \frac{\partial \mathbf{a}}{\partial \alpha} = \Big[\Big(-\frac{\partial a_N}{\partial \alpha} + a_A\Big)\sin\alpha - \Big(a_N + \frac{\partial a_A}{\partial \alpha}\Big)\cos\alpha\Big]\boldsymbol{\lambda_v}^T\hat{\mathbf{v}} + \\ \Big[\Big(\frac{\partial a_N}{\partial \alpha} - a_A\Big)\cos\alpha - \Big(a_N + \frac{\partial a_A}{\partial \alpha}\Big)\sin\alpha\Big]\boldsymbol{\lambda_v}^T\hat{\mathbf{u}} = 0\end{aligned} \tag{2.59}$$

Rearranging yields:

$$\begin{aligned}\Big[\Big(-\frac{\partial a_N}{\partial \alpha} + a_A\Big)\boldsymbol{\lambda_v}^T\hat{\mathbf{v}} - \Big(a_N + \frac{\partial a_A}{\partial \alpha}\Big)\boldsymbol{\lambda_v}^T\hat{\mathbf{u}}\Big]\sin\alpha \\ = \Big[\Big(a_N + \frac{\partial a_A}{\partial \alpha}\Big)\boldsymbol{\lambda_v}^T\hat{\mathbf{v}} + \Big(-\frac{\partial a_N}{\partial \alpha} + a_A\Big)\boldsymbol{\lambda_v}^T\hat{\mathbf{u}}\Big]\cos\alpha\end{aligned} \tag{2.60}$$

which can be solved for $\tan\alpha$:

$$\tan\alpha = \frac{\Big[\Big(a_N + \frac{\partial a_A}{\partial \alpha}\Big)\boldsymbol{\lambda_v}^T\hat{\mathbf{v}} + \Big(-\frac{\partial a_N}{\partial \alpha} + a_A\Big)\boldsymbol{\lambda_v}^T\hat{\mathbf{u}}\Big]}{\Big[\Big(-\frac{\partial a_N}{\partial \alpha} + a_A\Big)\boldsymbol{\lambda_v}^T\hat{\mathbf{v}} - \Big(a_N + \frac{\partial a_A}{\partial \alpha}\Big)\boldsymbol{\lambda_v}^T\hat{\mathbf{u}}\Big]} \tag{2.61}$$

We thus have an expression for the optimal control, α, which is determined from the costates,

which we can obtain using the method outlined in Section 2.2.

To solve for α from the adjoint variables, we express a_N and a_A as:

$$a_N = \frac{q_\infty S}{m} C_{N_\alpha} \cdot \alpha, \quad a_A = \frac{q_\infty S}{m} C_A \tag{2.62}$$

Taking partials with respect to the control, α, gives:

$$\frac{\partial a_N}{\partial \alpha} = \frac{q_\infty S}{m} C_{N_\alpha}, \quad \frac{\partial a_A}{\partial \alpha} = 0 \tag{2.63}$$

Combining these with Equation (2.61) yields:

$$\tan \alpha = \frac{C_{N_\alpha} \alpha \boldsymbol{\lambda}_\mathbf{v} \cdot \hat{\mathbf{v}} + (C_A - C_{N_\alpha}) \boldsymbol{\lambda}_\mathbf{v} \cdot \hat{\mathbf{u}}}{(C_A - C_{N_\alpha}) \boldsymbol{\lambda}_\mathbf{v} \cdot \hat{\mathbf{v}} - C_{N_\alpha} \alpha \boldsymbol{\lambda}_\mathbf{v} \cdot \hat{\mathbf{u}}} \tag{2.64}$$

We will now use Newton's method to solve for α. Newton's method is an iterative approach to finding the values of a function which make that function equal to zero (i.e., finding the zeroes or roots of the function). To implement this method, we need a function of α, $F(\alpha) = 0$ and also the derivative of that function with respect to α.

Rearranging Equation (2.64) and setting it equal to zero provides us with $F(\alpha)$:

$$\begin{aligned} F(\alpha) = \boldsymbol{\lambda}_\mathbf{v} \cdot \hat{\mathbf{v}} (C_A - C_{N_\alpha}) \sin \alpha - \boldsymbol{\lambda}_\mathbf{v} \cdot \hat{\mathbf{u}} C_{N_\alpha} \cdot \alpha \sin \alpha \\ - \boldsymbol{\lambda}_\mathbf{v} \cdot \hat{\mathbf{v}} C_{N_\alpha} \cdot \alpha \cos \alpha + \boldsymbol{\lambda}_\mathbf{v} \cdot \hat{\mathbf{u}} (C_{N_\alpha} - C_A) \cos \alpha = 0 \end{aligned} \tag{2.65}$$

and we take the derivative of Equation (2.65) with respect to α to obtain $F'(\alpha)$:

$$F'(\alpha) = \boldsymbol{\lambda}_{\mathbf{v}} \cdot \hat{\mathbf{v}}(C_A - C_{N_\alpha}) \cos\alpha - \boldsymbol{\lambda}_{\mathbf{v}} \cdot \hat{\mathbf{u}} C_{N_\alpha} \cdot \sin\alpha - \boldsymbol{\lambda}_{\mathbf{v}} \cdot \hat{\mathbf{u}} C_{N_\alpha} \cdot \alpha \cos\alpha$$
$$- \boldsymbol{\lambda}_{\mathbf{v}} \cdot \hat{\mathbf{v}} C_{N_\alpha} \cdot \cos\alpha + \boldsymbol{\lambda}_{\mathbf{v}} \cdot \hat{\mathbf{v}} C_{N_\alpha} \cdot \alpha \sin\alpha - \boldsymbol{\lambda}_{\mathbf{v}} \cdot \hat{\mathbf{u}}(C_{N_\alpha} - C_A) \sin\alpha \quad (2.66)$$

With $F(\alpha)$ and $F'(\alpha)$ in hand, we can obtain successively closer estimates for α from:

$$\alpha_{k+1} = \alpha_k - \frac{F(\alpha_k)}{F'(\alpha_k)} \quad (2.67)$$

where α_{k+1} refers to the new estimate and α_k is the previous one. We iterate on α until the difference between the new and previous estimates is below some predetermined threshold. At this point, α_{k+1} is a solution or root of Equation (2.65).

To begin the iteration process, we also need an initial estimate for $\alpha_{k=0}$. To determine this initial guess, we apply the small angle approximation to Equation (2.65), which reduces to:

$$- \boldsymbol{\lambda}_{\mathbf{v}} \cdot \hat{\mathbf{u}} C_{N_\alpha} \cdot \alpha_{k=0}^2 + \boldsymbol{\lambda}_{\mathbf{v}} \cdot \hat{\mathbf{v}}(C_A - 2C_{N_\alpha})\alpha_{k=0} + \boldsymbol{\lambda}_{\mathbf{v}} \cdot \hat{\mathbf{u}}(C_{N_\alpha} - C_A) = 0 \quad (2.68)$$

which results in an initial guess of:

$$\alpha_{k=0} = \frac{-\boldsymbol{\lambda}_{\mathbf{v}} \cdot \hat{\mathbf{v}}(C_A - 2C_{N_\alpha}) \pm \sqrt{(\boldsymbol{\lambda}_{\mathbf{v}} \cdot \hat{\mathbf{v}})^2 (C_A - 2C_{N_\alpha})^2 - 4(\boldsymbol{\lambda}_{\mathbf{v}} \cdot \hat{\mathbf{u}})^2 C_{N_\alpha}(C_{N_\alpha} - C_A)}}{2\boldsymbol{\lambda}_{\mathbf{v}} \cdot \hat{\mathbf{u}} C_{N_\alpha}} \quad (2.69)$$

2.5 The Multiple Shooting Method

Initially, the state and costates estimates were integrated forward with the intent of comparing the final computed results with the final state and costate conditions. This is known as the simple shooting method where differences between the integrated and estimated costates are only examined at the final time and those differences are used to adjust the initial costate estimates ($\boldsymbol{\lambda}_0$); the process is then repeated until the final differences are driven to 0. Due to the highly unstable nature of the costates, where minor changes in their initial estimates can result in quickly diverging values, the problem was modified to apply a unique variation of the multiple shooting method. In this method, the trajectory is divided into w nodes, and the value of the costates are calculated at each of the nodes using Equation (2.35); these values are stored and will be used later to "reset" the costates to prevent unstable behavior. Figure 2.2 shows a notional depiction of the application of the multiple shooting method for the initial integration of λ_x. At t_1, λ_x is integrated forward until t_2 using Equation (2.9). At this point, the value of λ_x is reset to the value of λ_x at t_2 computed and stored earlier. The integration continues to t_3 where the process is repeated until reaching the final node. Using a nonlinear systems-of-equation solver[3], the costate estimates at each node are adjusted and stored and the integration is executed again. The process continues until the differences, $\Delta\lambda_x$, also known as defects, between the integrated and computed (stored) values, are driven to zero. Figure 2.3 shows the final result of this process for λ_x.

[3]The NSWC Math Library [42] routine, HBRD, was used in this application.

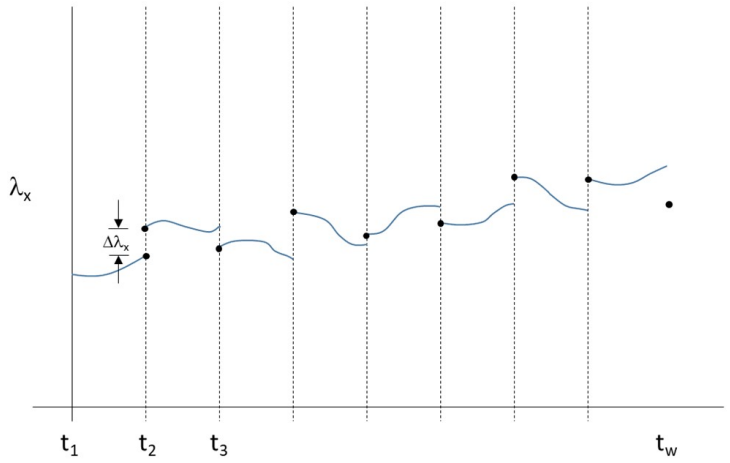

Figure 2.2: λ_x for a Non-converged Solution

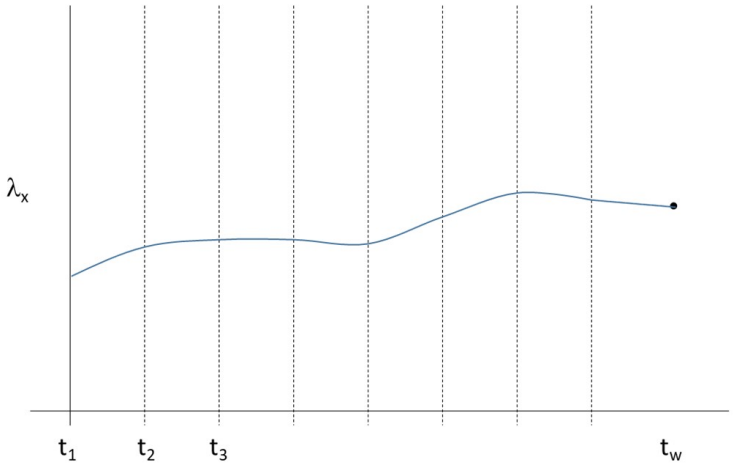

Figure 2.3: λ_x for a Converged Solution

To employ the solver, we established a vector of independent variables and an associated vector of dependent variables. The solver functions by manipulating the vector of independent variables, $\boldsymbol{p}$, to drive each element of the dependent vector, $\boldsymbol{F}(\boldsymbol{p})$, to be zero (within a user-defined tolerance). Upon convergence of the solver, the final elements of $\boldsymbol{p}$ will be the solution to the system of nonlinear equations.

The independent vector, $\boldsymbol{p} \in \mathbb{R}^{n_{\boldsymbol{p}}}$, where $n_{\boldsymbol{p}} = 4w$, is defined by:

$$\boldsymbol{p} = \left\{ \begin{array}{c} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \\ : \\ \boldsymbol{\lambda}_{w-1} \\ \boldsymbol{\nu} \\ \nu_0 \\ \theta_0 \end{array} \right\} \tag{2.70}$$

where each $\boldsymbol{\lambda}_i$ is the four-dimensional costate vector (comprised of λ_x, λ_y, λ_{v_x}, and λ_{v_y}) at the i^{th} node. Only the first $w-1$ values of $\boldsymbol{\lambda}$ are included in $\boldsymbol{p}$; $\boldsymbol{\lambda}_f$ is computed from $\boldsymbol{\nu}$.

The unique aspect of this application results from the fact that only the costates, along with some transversality conditions, are included in $\boldsymbol{p}$; the state variables are merely integrated forward in time as usual. This has the advantage of keeping $\boldsymbol{p}$ smaller, and was possible due to the fact that the state variables were not nearly as divergent or non-linear as the costates.

The corresponding dependent vector $\boldsymbol{F}(\boldsymbol{p}) : \mathbb{R}^{n_{\boldsymbol{p}}} \rightarrow \mathbb{R}^{n_{\boldsymbol{p}}}$,

$$\boldsymbol{F}(\boldsymbol{p}) = \left\{ \begin{array}{c} \Delta\boldsymbol{\lambda}_2 \\ \Delta\boldsymbol{\lambda}_3 \\ \vdots \\ \Delta\boldsymbol{\lambda}_{w-1} \\ \Delta\boldsymbol{\lambda}_w \\ \Delta\mathbf{r} \\ \lambda_{v_x,0} - \nu_0 * \cos\theta_0 \\ \lambda_{v_y,0} - \nu_0 * \sin\theta_0 \end{array} \right\} \tag{2.71}$$

where the $\Delta\boldsymbol{\lambda}_i \in \mathbb{R}^4$ are the differences between the costate vector integrated forward from the previous node, $i-1$, to the current node, i, and the previously stored $\boldsymbol{\lambda}_i$. For example, $\Delta\boldsymbol{\lambda}_2 = \boldsymbol{\lambda}_{2_{integrated}} - \begin{pmatrix} \boldsymbol{p}_5 \\ \boldsymbol{p}_6 \\ \boldsymbol{p}_7 \\ \boldsymbol{p}_8 \end{pmatrix}$, where $\boldsymbol{\lambda}_{2_{integrated}}$ comes from integrating $\boldsymbol{\lambda}_1 = \begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_3 \\ \boldsymbol{p}_4 \end{pmatrix}$ forward from t_1 ($t = 0$) to time t_2, using Equation (2.9).

As to the last four terms of Equation (2.71), $\Delta\boldsymbol{\lambda}_w$ (the wth node corresponds with t_f) is the difference between the integrated value of $\boldsymbol{\lambda}_w$ and $\boldsymbol{\lambda}_f$ from Equation (2.10), $\Delta\mathbf{r}$ is the difference between the projectile position and the position of the target at time t_f, and the last two terms are from the initial transversality conditions. The numerical nonlinear-system-of-equations solver iterates on $\boldsymbol{p}$ until $\boldsymbol{F}(\boldsymbol{p}) \approx \mathbf{0}$. Figure 2.3 shows how the solver drives the $\Delta\lambda_x$ values to zero, as representative of what happens to all four values of the costate vector.

Until now, the final time, t_f, has been fixed. If we allow t_f to be free as the optimization code is executing, an optimal time of flight, which maximizes the projectile final velocity, will be determined. Based on the Maximum Principle, this optimal time of flight will be achieved when the Hamiltonian, $\mathcal{H}$, is zero. If we add t_f as the new last term in the independent vector in Equation (2.70) and add the Hamiltonian as the new last variable in the dependent

vector in Equation (2.71), this will be achieved.

At this point, we have a method to obtain the costates for a reference trajectory, which can be used as the initial estimates for the costates for a general optimal control problem. We will now apply this method to a specific optimal control problem involving a gun-launched guided projectile.

Chapter 3

Application of the Methodology

To demonstrate the utility of this methodology, several cases are presented where the reference trajectories were generated using two different methods. In the first case, an approximate optimal trajectory was obtained from previously-developed software employing a direct method. In the second case, two purely ballistic (i.e., $\alpha = 0$ throughout the flight) trajectories (both lofted and depressed) flown to the same range and final time were used. The point of using the ballistic trajectories is that the reference trajectories do not need to be nearly optimal. They just need to be of the right shape to produce estimates for the costates that are close enough to permit convergence. In each case, the reference trajectory was then divided into $w - 1$ panels (defined by w nodes). Using Equation (2.45), a value for $\boldsymbol{\nu}$ was calculated and subsequently Equation (2.35) was used to determine the $\boldsymbol{\lambda}$'s at each of the nodes. With the $\boldsymbol{\lambda}$'s in hand, Equations (2.65), (2.66), (2.67), and (2.69) were then used to calculate the control, α, at each time.

Table 3.1 lists the values for the physical characteristics of the projectile that were used in the subsequent analysis cases.

Table 3.1: Values for physical parameters used in example problem.

Radius of the Earth, R_e	6371 km
Earth Gravitational Constant, μ_e	398600.5 km^3/s^2
Reference Area of Projectile, S	0.005352 m^2
Mass of Projectile, m	12 kg
Initial Velocity, v_0	1.0 km/s
Axial Aerodynamic Coefficient, C_A	0.15
Normal Aerodynamic Coefficient Slope, C_{N_α}	5 per radian
Number of Nodes, w	20

This approach successfully converged to an optimal solution in both cases discussed above with the results shown in the following figures. In each case shown below, "optimal" refers to the solution resulting from the Multiple-Shooting Method approach. Figure 3.1 shows a comparison of the approximate optimal and optimal trajectories, indicating the degree to which the optimal trajectory matches the original approximate optimal one (since the velocity profiles of the two trajectories are similarly close, this plot is not included).

Figure 3.2 shows a comparison of the angle-of-attack histories for the approximate optimal and optimal trajectories. Notice that the approximate optimal angle-of-attack history is not as "smooth" as is the case for the optimal trajectory. This comparison exemplifies one of the differences between the direct and indirect methods: whereas the direct method provides a good approximation of the optimal control, the indirect method results in an exactly optimal solution.

A comparison of the lofted ballistic trajectory and the resulting optimally-controlled trajectory is shown in Figure (3.3). This comparison indicates that the lofted ballistic trajectory is very nearly optimal already. For the optimal trajectory, the projectile is fired at a slightly lower pitch angle ($\theta_0 = 9.4^o$) than the ballistic one ($\theta_0 = 14.1^o$). There is a slight increase in the final velocity for the optimal trajectory than for the ballistic case (575 m/s vs. 573 m/s, respectively). Since the velocity profiles are very similar, they are not shown.

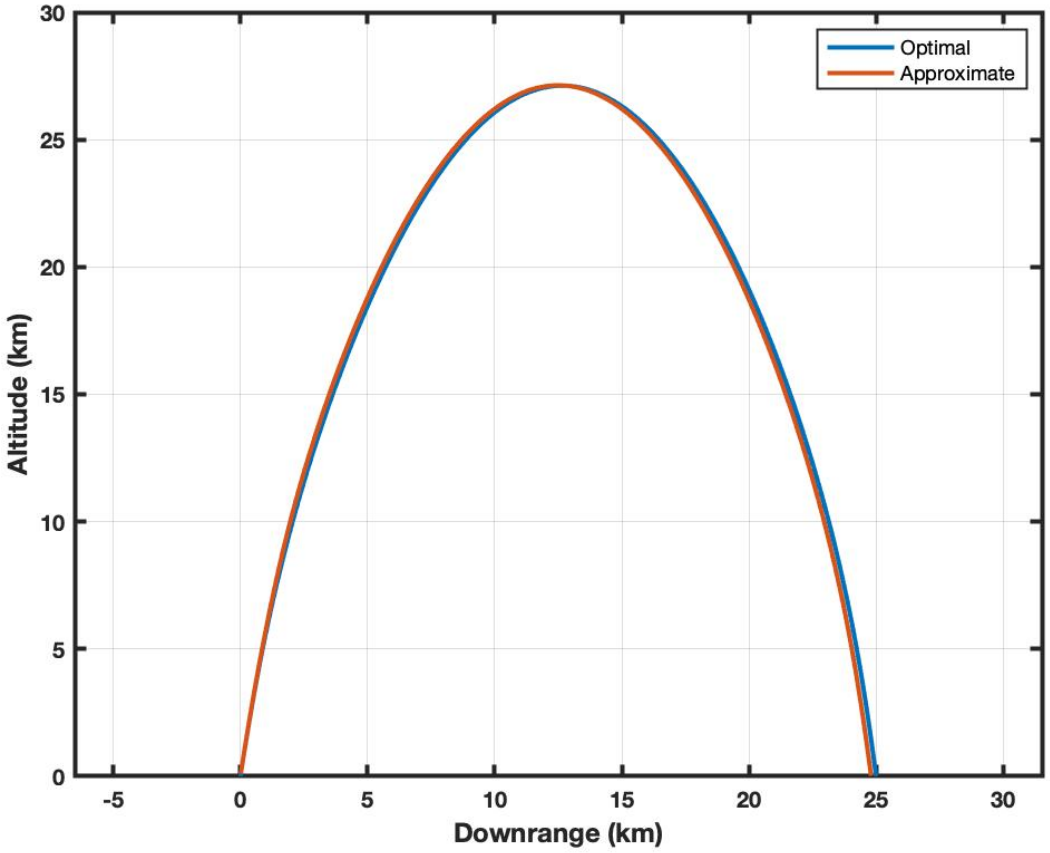

Figure 3.1: Comparison of Approximate and Optimal Trajectories

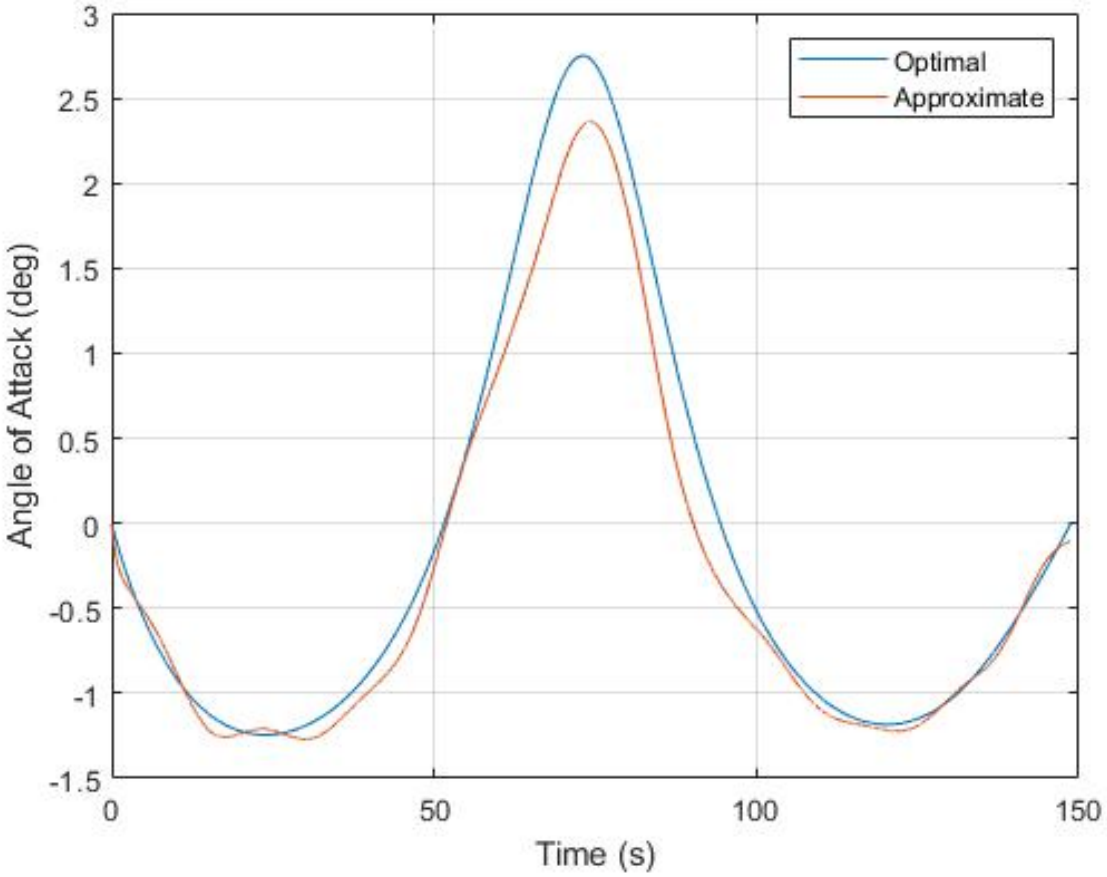

Figure 3.2: Comparison of Approximate and Optimal Angle of Attack Histories

A comparison of the depressed ballistic trajectory and the resulting optimally-controlled trajectory is shown in Figure (3.4). The optimal trajectory achieves the same range and time of flight as the ballistic trajectory, despite the fact that it flies significantly higher. On the optimal trajectory, the projectile is fired at a lower pitch angle ($\theta_0 = 62^o$) compared to the 74^o pitch angle for the ballistic trajectory. This flight through thinner air allows the projectile to arrive with a higher final velocity than the purely ballistic case (436 m/s vs. 413 m/s) as specified by the cost function.

This comparison showcases one of the main trends evident in the optimal trajectories. In general, when compared to the reference trajectory, the optimal one will be launched at a

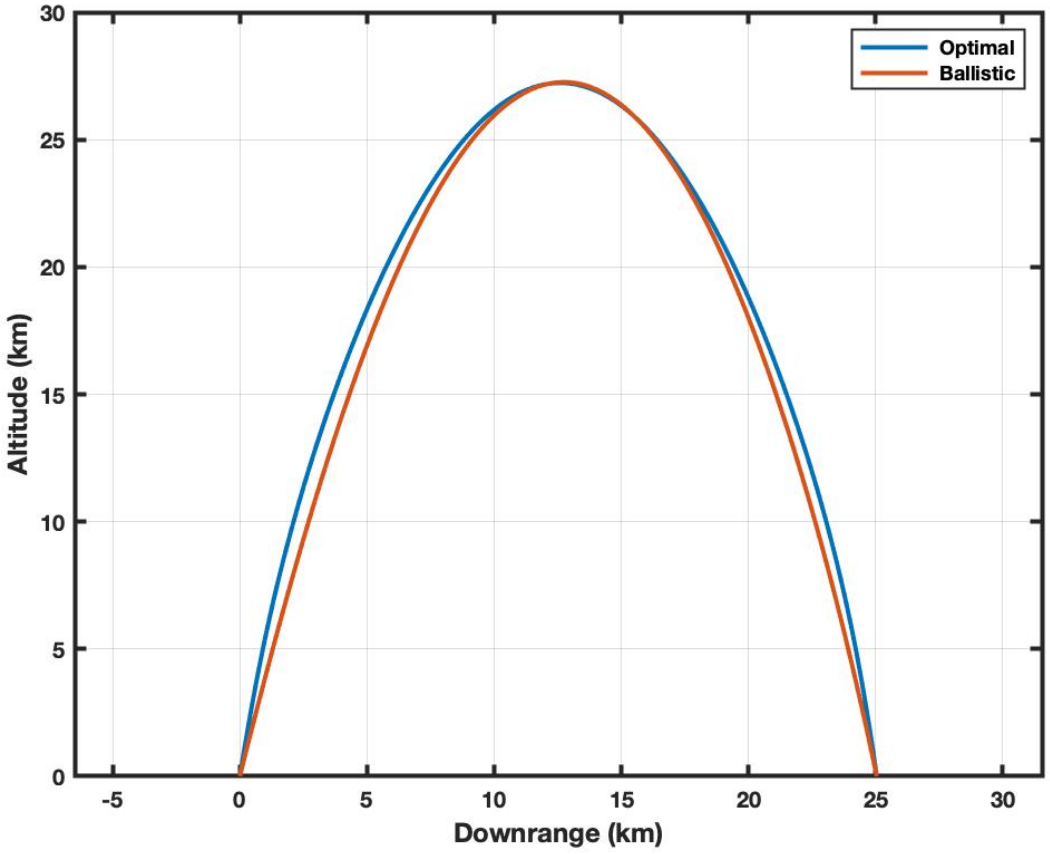

Figure 3.3: Comparison of Lofted Ballistic and Optimal Trajectories

steeper launch angle and will fly higher through thinner air to achieve a higher final velocity. Another interesting item is that the optimal trajectory results in a higher final velocity than the ballistic case. The general belief in the gun community is that ballistic cases (because they minimize the control input, and therefore minimize drag) will result in a higher final velocity. This is not the case; optimization permits a control history that achieves a higher final velocity.

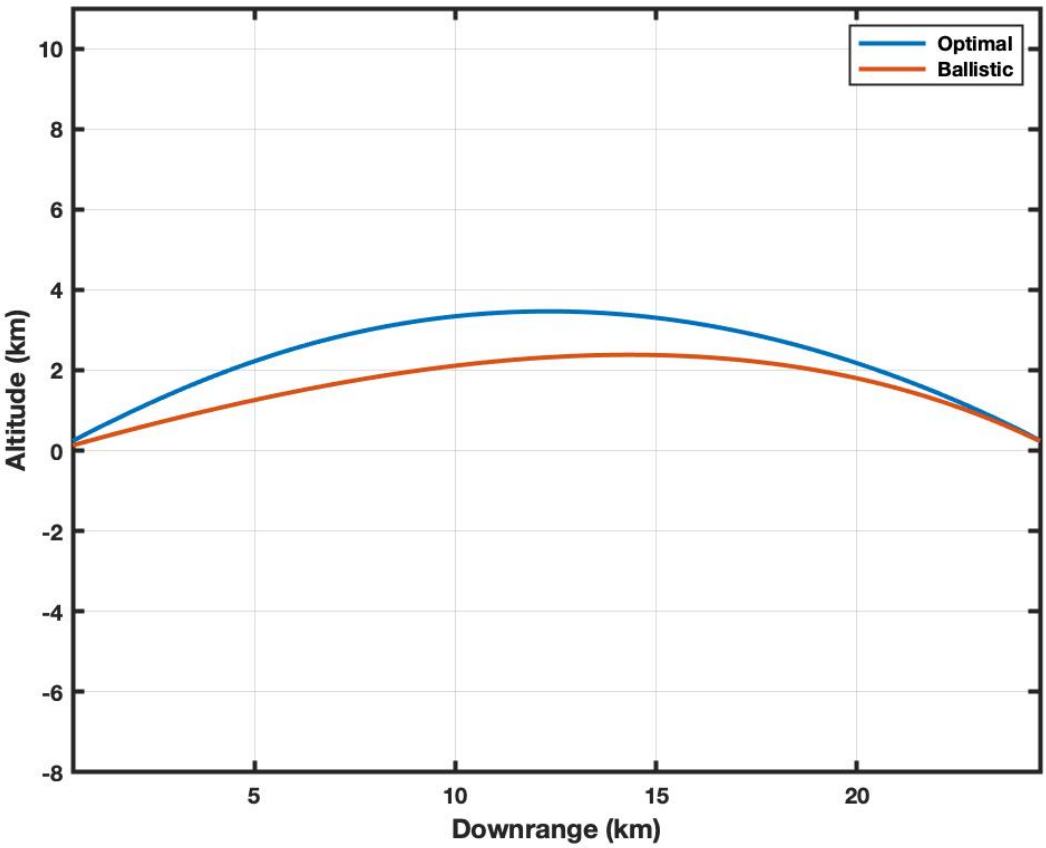

Figure 3.4: Comparison of Depressed Ballistic and Optimal Trajectories

Figure (3.5) shows a comparison of the velocity for the depressed ballistic and optimal trajectories. As a result of travelling through less dense air, the optimally-controlled projectile has a higher velocity for the entire flight, and finishes with a slightly higher velocity as previously mentioned.

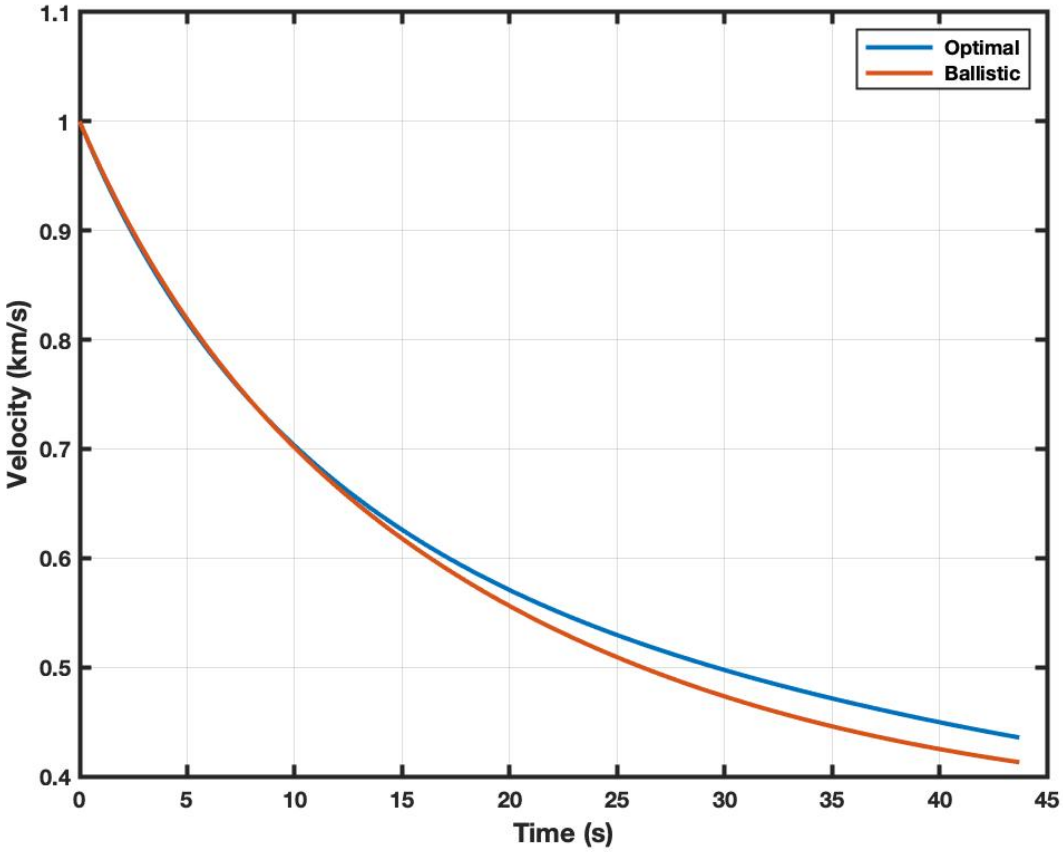

Figure 3.5: Comparison of Depressed Ballistic and Optimal Speed Profiles

Figure (3.6) shows a history of the angle of attack for the optimal trajectory (the angle of attack for the ballistic trajectory is uniformly 0). The optimally-controlled projectile gradually pitches over (negative angle of attack) from the initial velocity vector, then pulls slightly positive angles of attack on the descent portion of the flight.

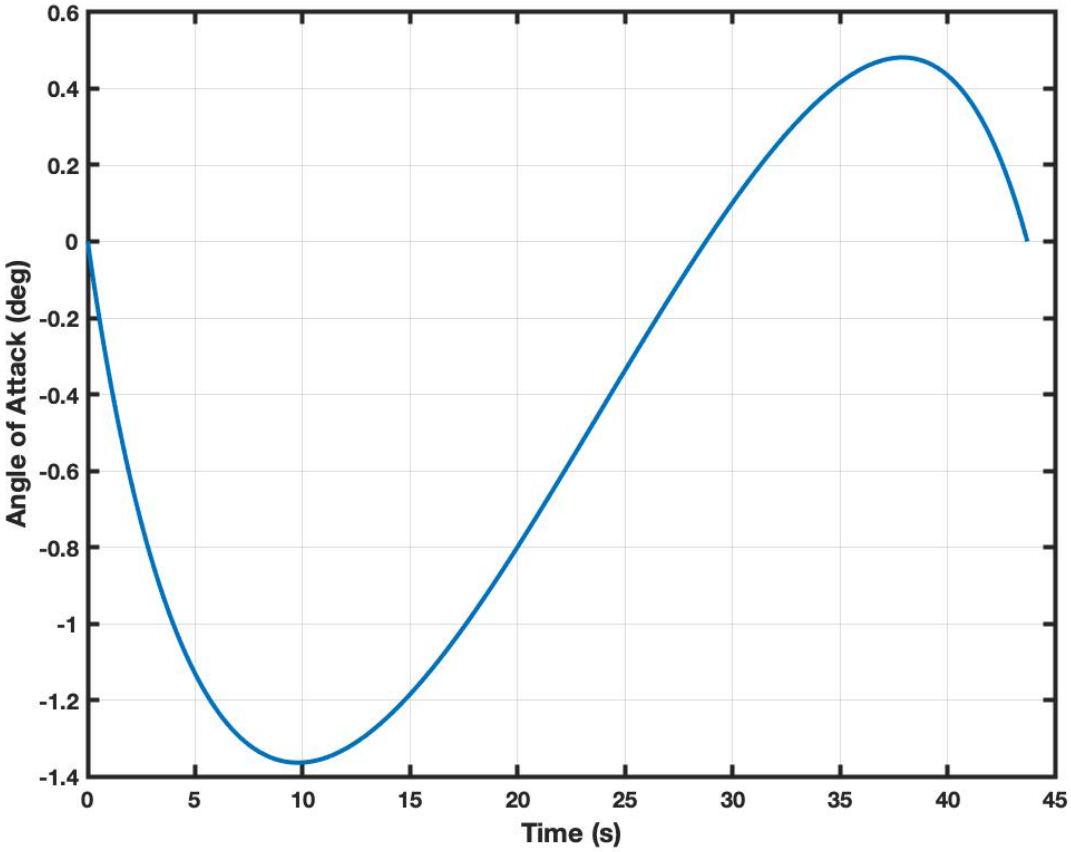

Figure 3.6: Optimal Angle of Attack History

Figures 3.7 - 3.10 show comparisons of the initial estimates for the costates for the approximate optimal and optimal from depressed ballistic cases discussed thus far. Figures 3.7 and 3.8 show the position costate estimates while Figures 3.9 and 3.10 show the estimates of the velocity costates. As can be seen from the plots, the estimates derived from the approximate optimal trajectory are closer to the histories produced during the generation of the optimal trajectory than those produced from the ballistic trajectory. However, even the coarser initial estimates from the ballistic trajectory are sufficient to allow the method to converge to a solution. It should be noted that the times of flights of these two trajectories (approximate optimal and depressed ballistic) are significantly different ($\sim$150 s vs. $\sim$50 s, respectively)

which explains the difference in the length of the plotted data.

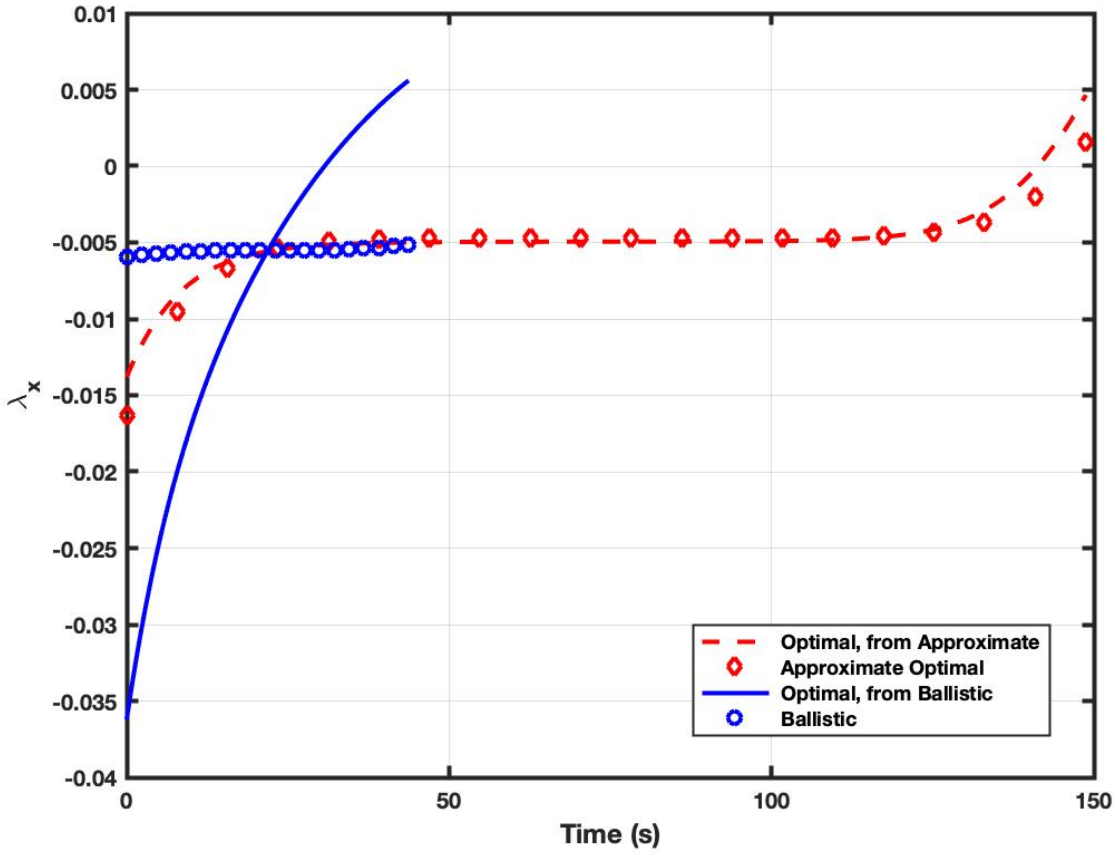

Figure 3.7: Comparison of λ_x for Approximate, Ballistic and Optimal Trajectories

It is worth mentioning that although some of the costate estimates appear to be relatively far off from the histories generated during the optimal trajectories, they are sufficiently close so as be in the correct "valley" to allow the methodology to converge to an optimal solution.

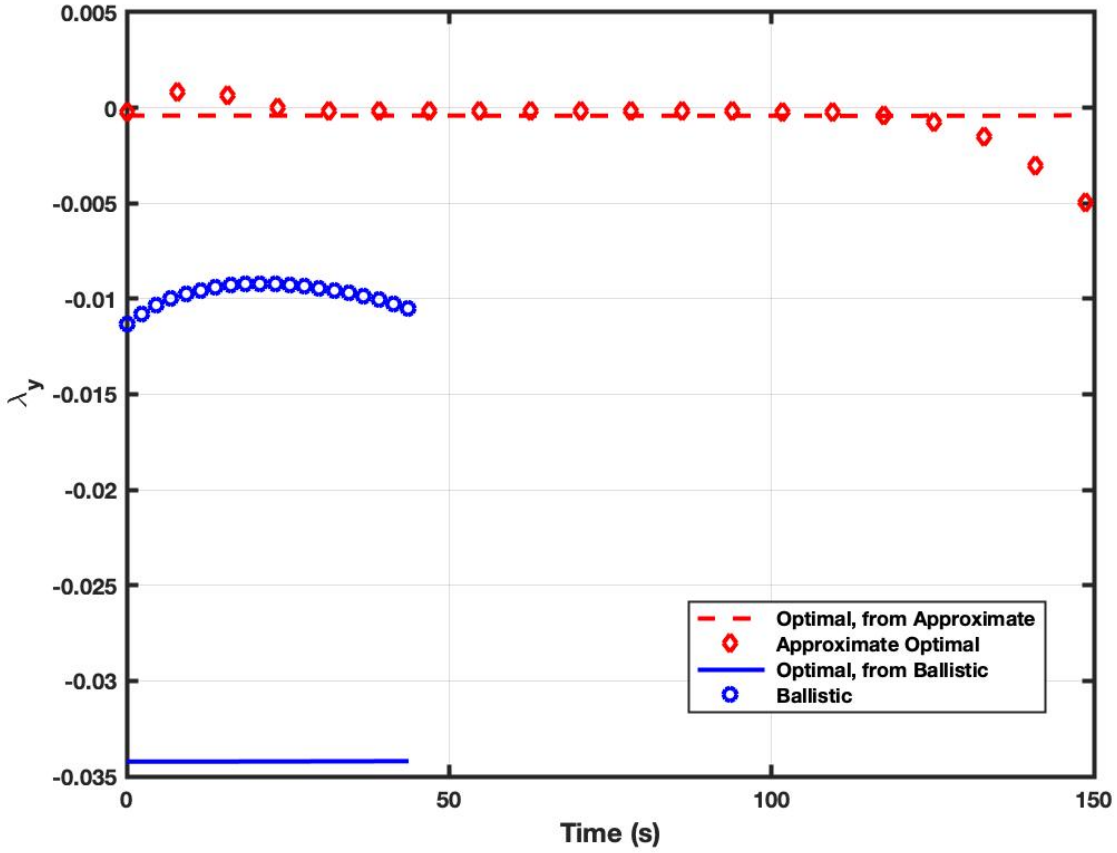

Figure 3.8: Comparison of λ_y for Approximate, Ballistic and Optimal Trajectories

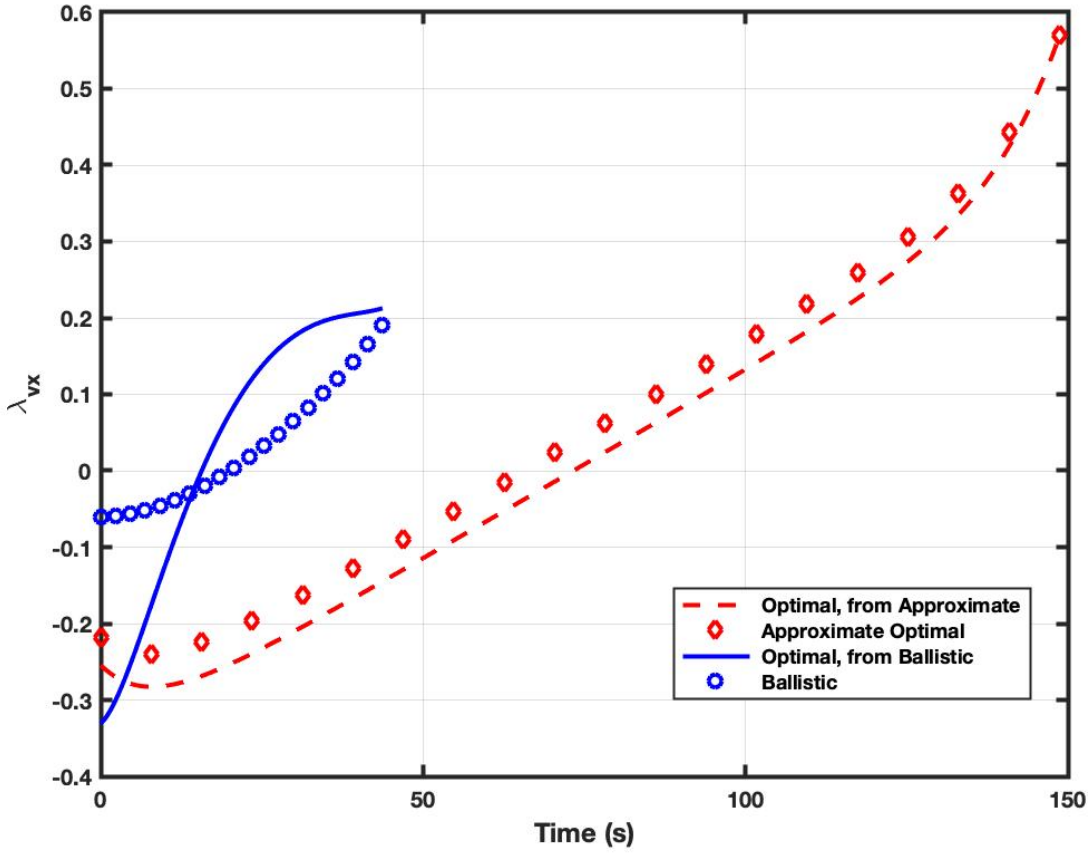

Figure 3.9: Comparison of λ_{v_x} for Approximate, Ballistic and Optimal Trajectories

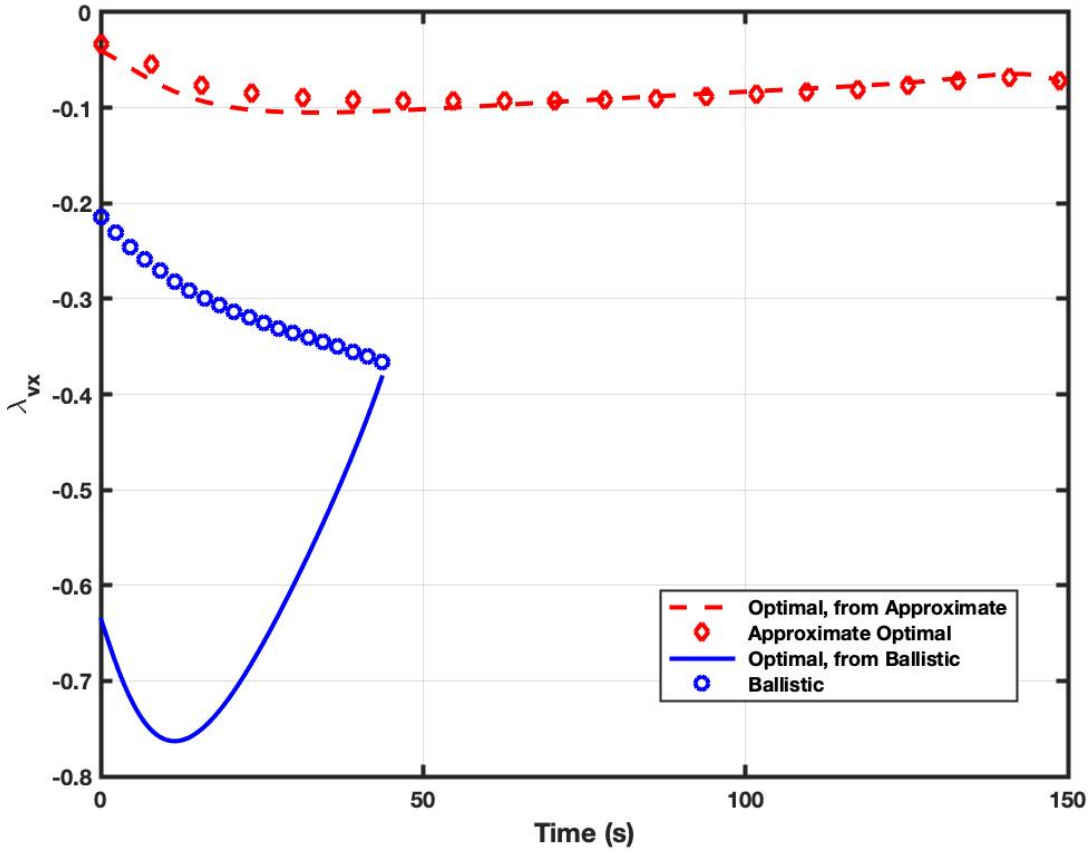

Figure 3.10: Comparison of λ_{v_y} for Approximate and Optimal Trajectories

Subsequently, another case was chosen to explore the robustness of the methodology. By manipulating a longer-range (approximately 40 km) ballistic trajectory by manually altering the angle of attack history, a trajectory was developed that extended the range to approximately 56 km. This was done by specifying an angle of attack at each of the nodes and manually iterating until a longer range was achieved. The methodology was applied using this manually-derived control history and an optimal trajectory was achieved. The two trajectories are shown in Figure 3.11. As can be seen, both trajectories are essentially ballistic during the first half of their flight; the optimal trajectory has a higher launch angle and pulls up slightly relative to the manual one. Also the optimal trajectory is higher in the range extension portion of flight, allowing flight through less dense air and resulting in a higher final velocity, as shown in Figure 3.12.

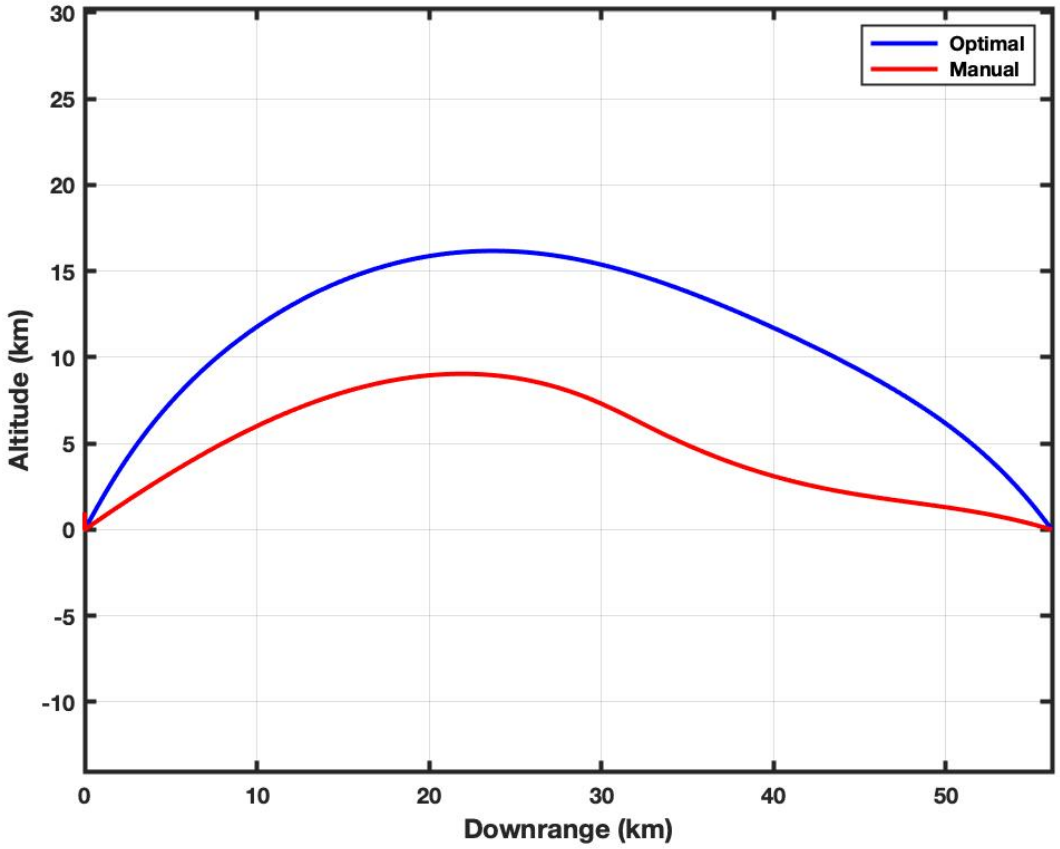

Figure 3.11: Comparison of 56 km Manual and Optimal Trajectories

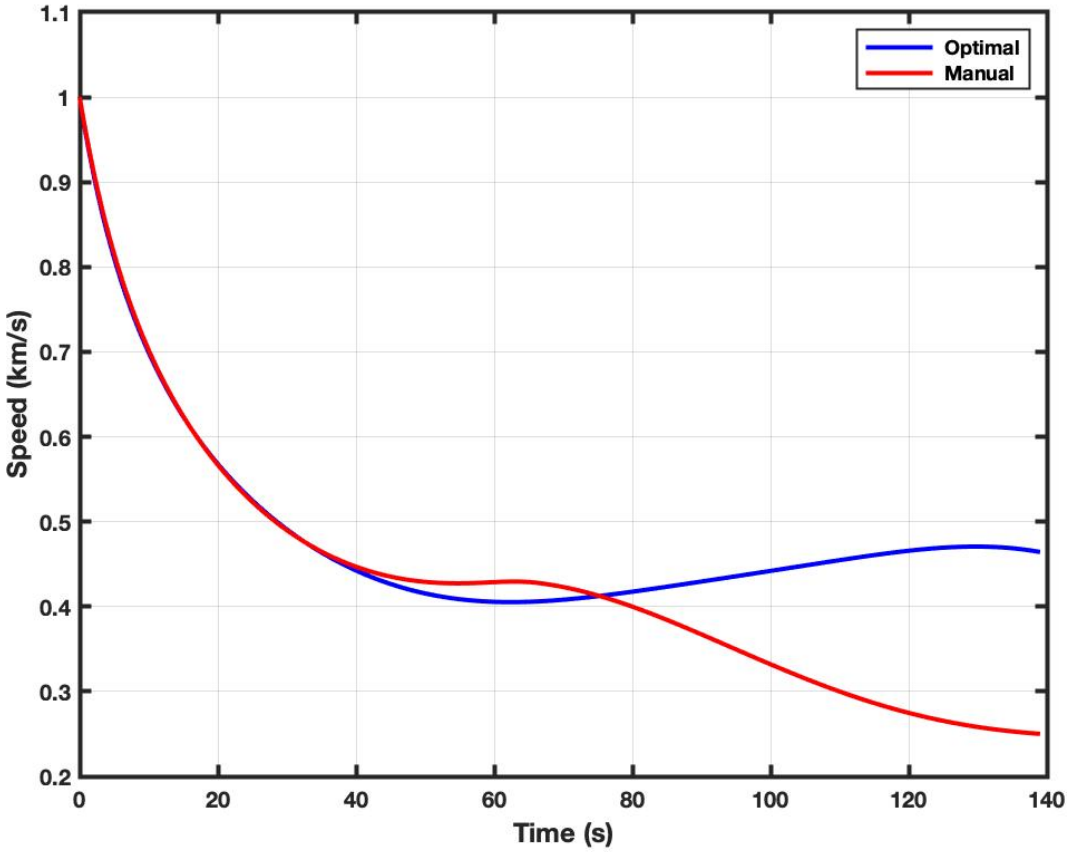

Figure 3.12: Comparison of Velocity Profiles for 56 km Manual and Optimal Trajectories

Figure 3.13 shows a comparison of the position costate estimates and optimal history for λ_x and λ_y, while Figure 3.14 shows a comparison of the velocity costate estimates and optimal history for λ_{v_x} and λ_{v_y}. As shown earlier, although the estimates appear to be relatively distant from the integrated traces, they are sufficiently close to allow convergence to the optimal solution.

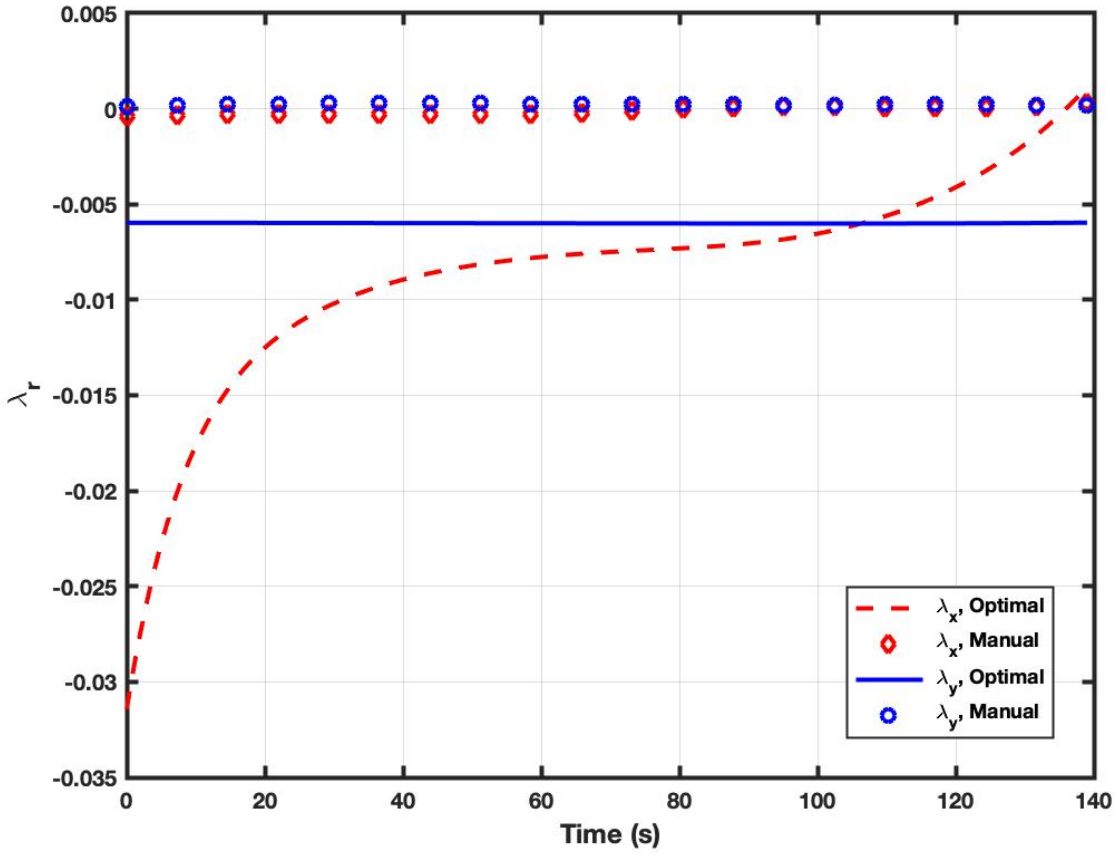

Figure 3.13: Comparison of Estimates and Optimal History for λ_x and λ_y for 56 km Trajectory

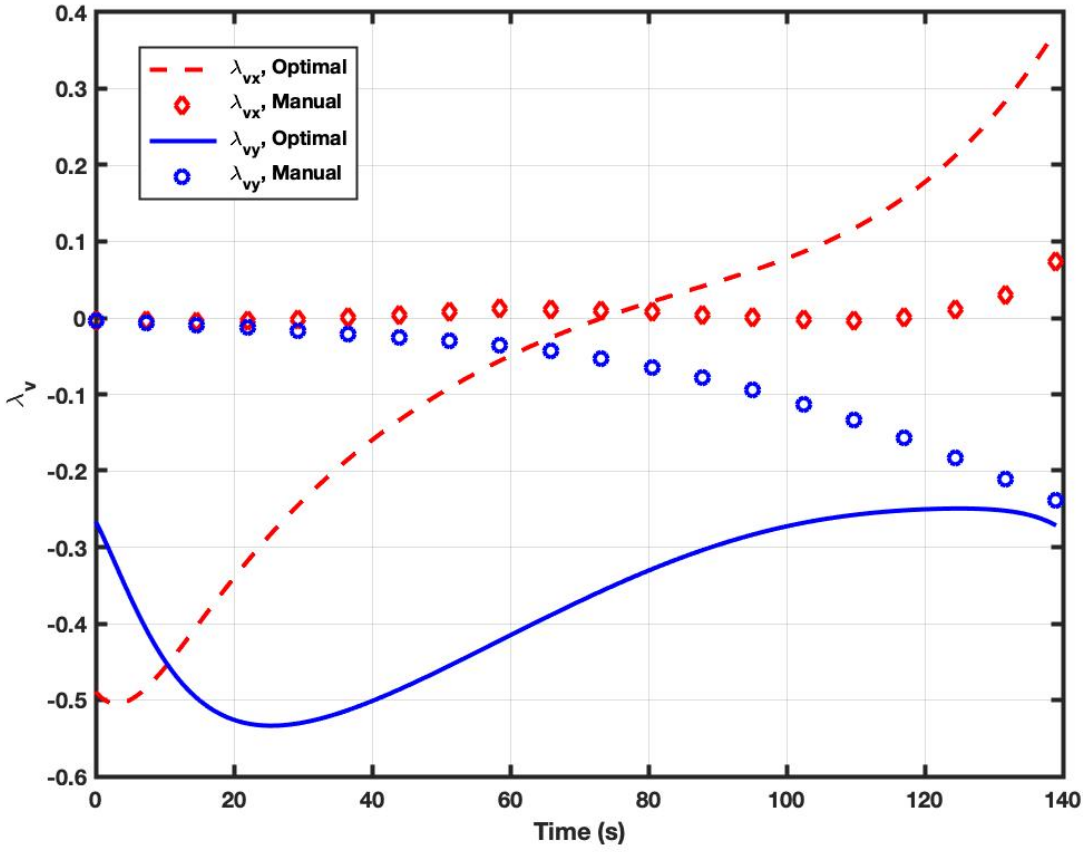

Figure 3.14: Comparison of Estimates and Optimal History for λ_{v_x} and λ_{v_y} for 56 km Trajectory

3.1 Range Extensions

As a follow-on to the previous range extension exercise, another series of runs was completed to test the ability of the methodology to extend the range of the trajectory. Initially an 80 km case was run and a converged solution achieved. The final values of the $\boldsymbol{\lambda}$'s determined upon convergence of the solution were then used as the initial estimates for the subsequent run. The desired range was incremented by 5 km (i.e., to 85 km) and the final time, t_f, was increased by approximately 10 s. The code was executed again and a new converged solution obtained. This process was repeated in distance increments of 5 km from 80 km up

to 150 km. There was no converged solution at 150 km indicating that the maximum range would be obtained at a value between 140 and 150 km. After using smaller and smaller range increments, a maximum range of 141.25 km was determined.

Figure 3.15 shows the combined altitude vs. downrange plots for all of these cases (no key is displayed on this plot for clarity). As can be seen, higher and higher pitch-up maneuvers result as the desired range increases. This is borne out in Figure 3.16 which shows the angle-of-attack history for each of these trajectories (the longer times of flight are associated with the longer range trajectories). As the desired range increases, higher and higher α's are commanded (especially towards the end of the trajectory) to achieve the range specified. The longest range trajectory (141.25 km) actually has a slightly shorter time of flight than the 140 km trajectory. This is a result of the cost function which prioritizes impact velocity vs. one which minimizes time of flight. This can be seen in Figure 3.17, where the 140 km trajectory (longest time of flight) has a higher final velocity. To achieve the max range of 141.25 km, impact velocity is sacrificed. The 140 km trajectory has sufficient extra energy to fly higher (and thus take longer) such that some impact velocity is preserved. The 141.25 km trajectory does not have this reserve energy.

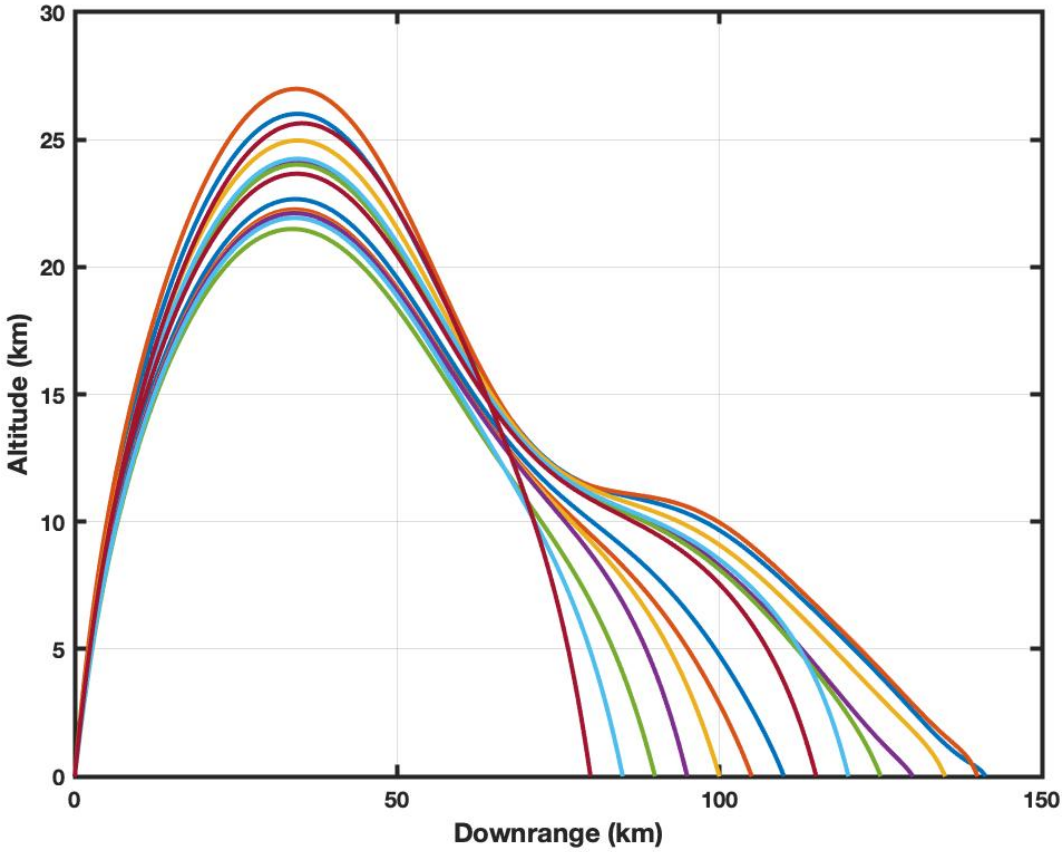

Figure 3.15: Comparison of Range-Extension Trajectories from 80 km to 141.25 km

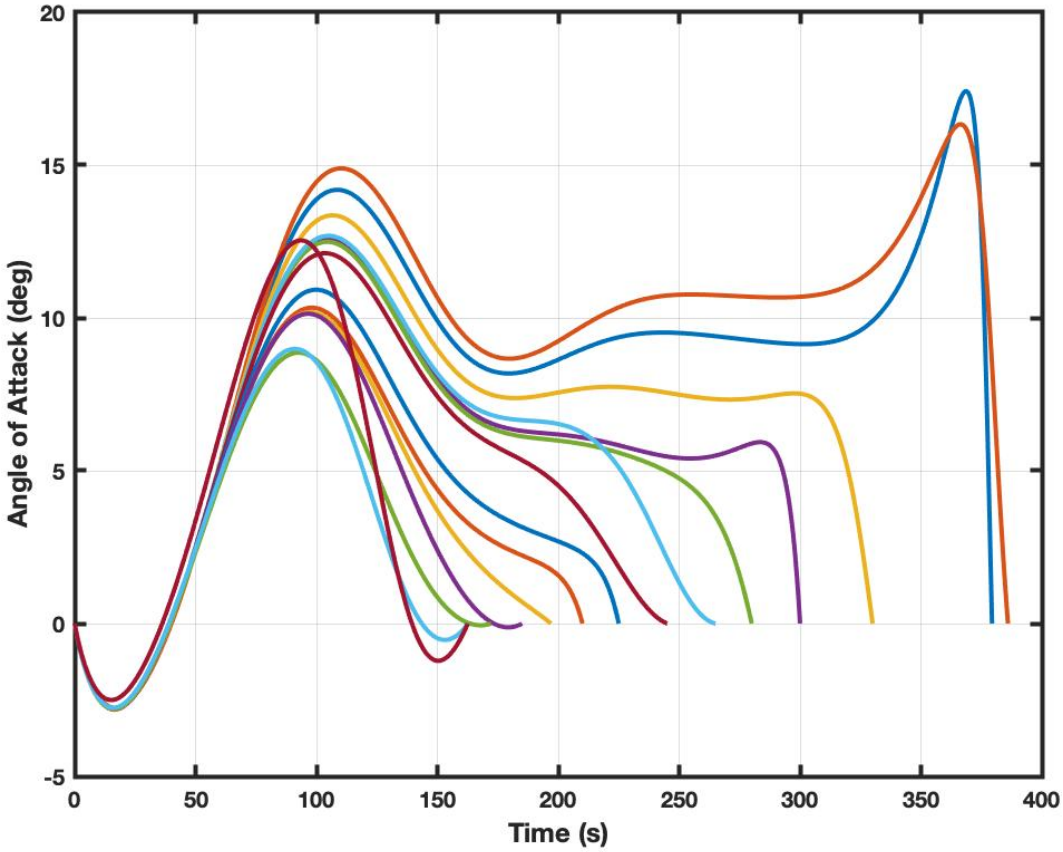

Figure 3.16: Comparison of Angle-of-Attack Profiles for Range-Extension Trajectories

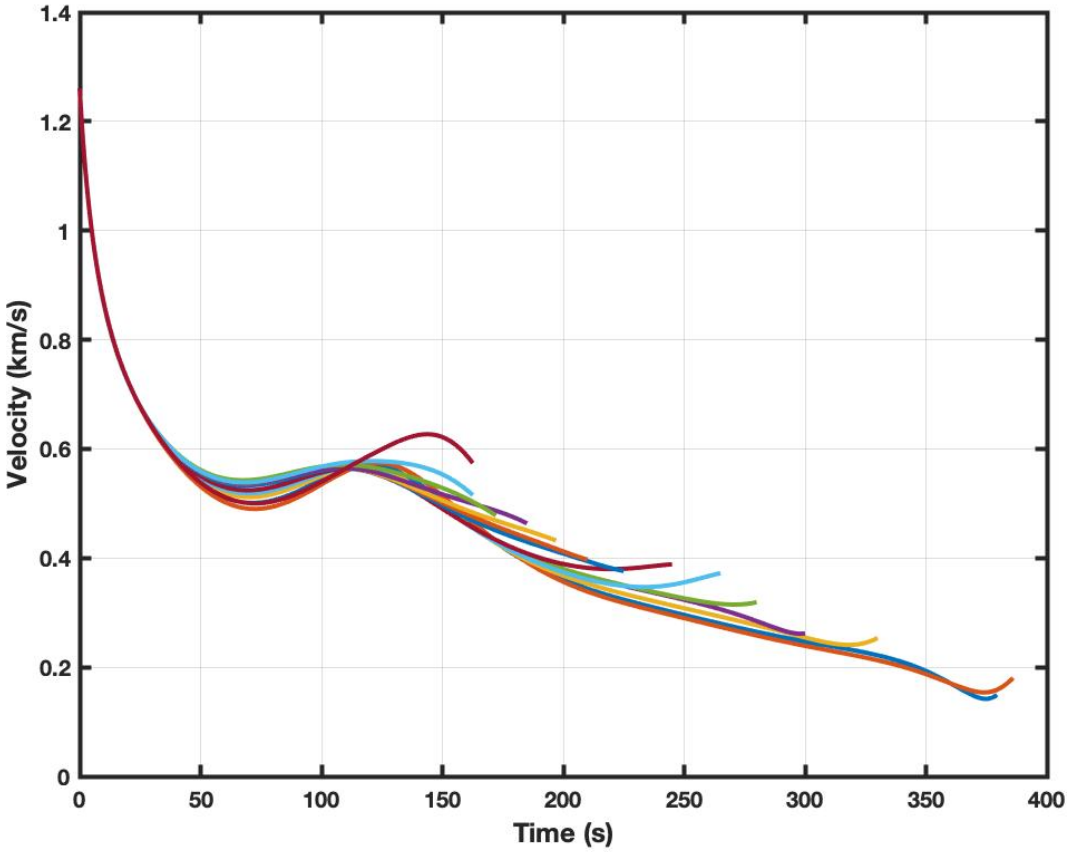

Figure 3.17: Comparison of Velocity Profiles for Range-Extension Trajectories

3.2 Time Extensions

In another exercise, a set of runs were conducted that fixed the range at 60 km and then varied the time of flight for the trajectory (beginning at 124 s and ranging up to 170 s). As before, the final $\boldsymbol{\lambda}$'s from the initial run were used as the initial $\boldsymbol{\lambda}$'s for the subsequent runs. The results of these runs (except for 170 s, which did not converge to a solution) are shown in Figures 3.18 - 3.20.

Figure 3.18 shows a time history for each trajectory at the various times of flight. As the time

is increased, the projectile is fired at higher initial pitch angles and flies a higher trajectory. This allows the projectile to fly through thinner air, experience less drag, and then finish with a higher final velocity (as seen in Figure 3.19). The character of the angle-of-attack history, Figure 3.20, also changes as the time increases. At the lower time of flight, the projectile gently pitches over as the trajectory progresses. As the time increases, the angle of attack increases (becomes less negative) and even becomes positive at the final time of flight plotted (165 s).

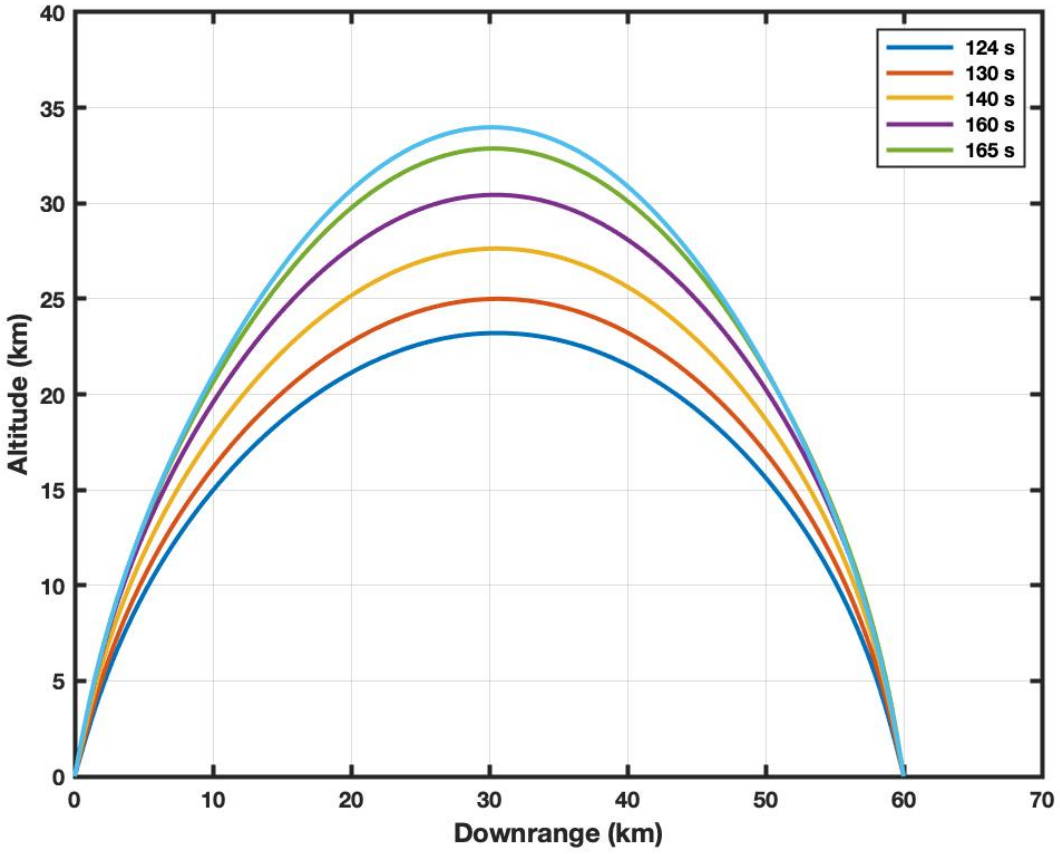

Figure 3.18: Comparison of 60 km Trajectories with Varying Times of Flight

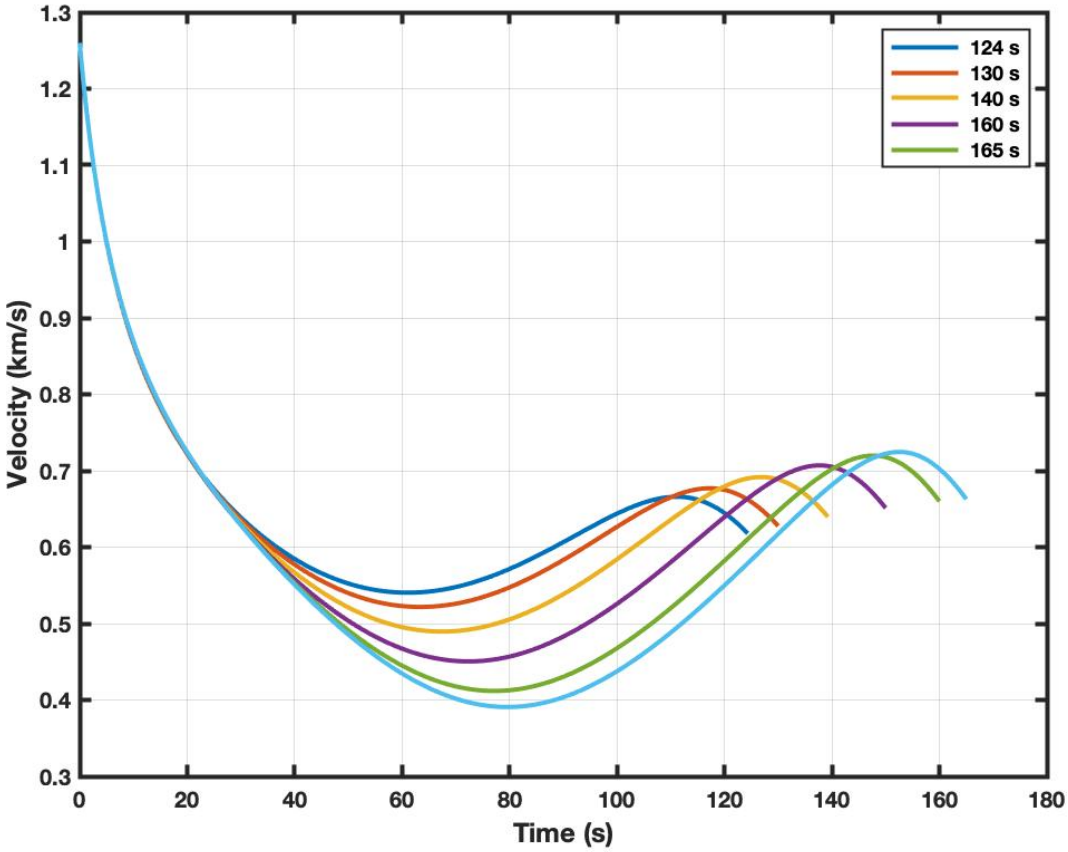

Figure 3.19: Comparison of Velocity Histories with Varying Times of Flight

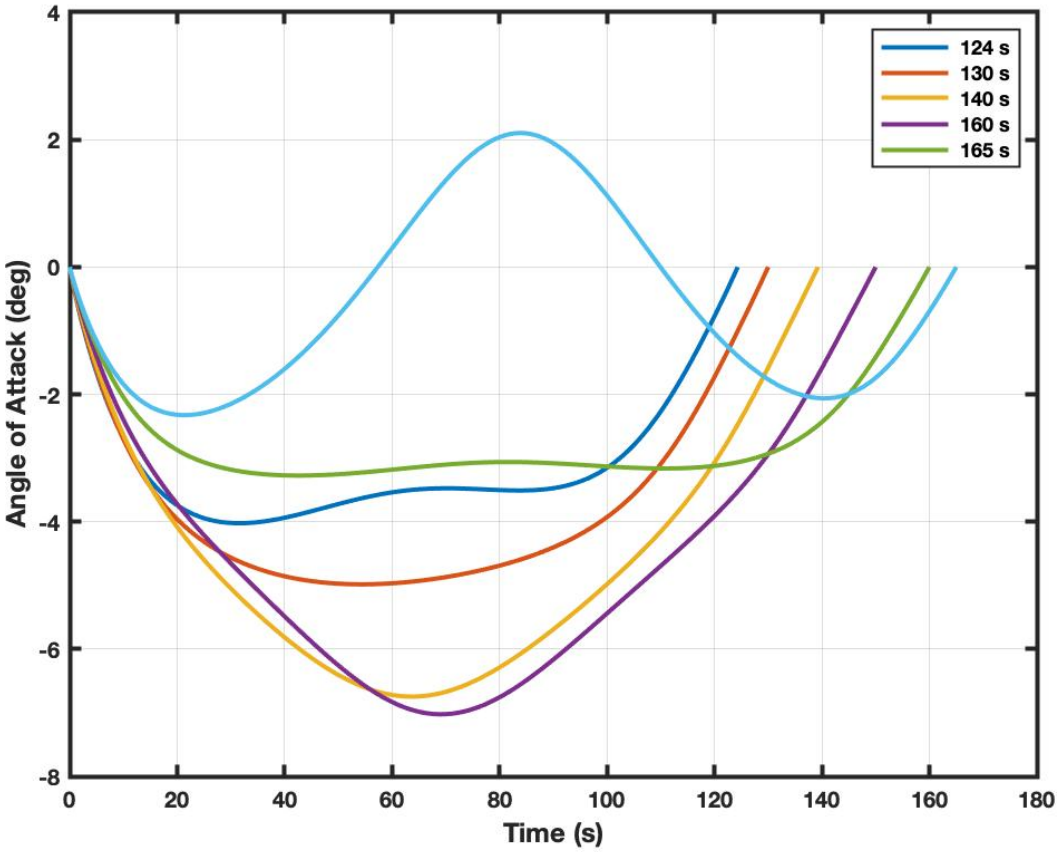

Figure 3.20: Comparison of Angle-of-Attack Histories with Varying Times of Flight

3.3 Free Time of Flight

Until this point, the cases shown have involved fixed times of flight (i.e., the optimizing routine was not permitted to adjust the time of flight as iterations were executed). Modifications were made to the optimization routine to include t_f at the end of the independent vector, $\boldsymbol{p}$, and the Hamiltonian, $\mathcal{H}$, was added to the end of the dependent vector, $\boldsymbol{F}(\boldsymbol{p})$ (the Hamiltonian should be 0 throughout the trajectory if time is free). A ballistic trajectory with a launch velocity of 2 km/s was generated and both the time-fixed and time-free routines

applied to it. The results for these runs are shown in Figures 3.21 - 3.24.

As shown in Figure 3.21, the time-free trajectory has a higher apogee than the time-fixed trajectory; this is a direct result of the optimization routine adjusting the launch angle and final time to slightly increase the final velocity (812 m/s vs. 809 m/s, as seen in Figure 3.22) which is achieved by flying higher with a correspondingly longer time of flight. As was observed previously, both of the optimal trajectories have a much higher final velocity than the ballistic case, which shows again the value of optimization. The angle-of-attack history, Figure 3.23, shows that the free-time case pitches over slightly more than the time-fixed case; this is a result of the higher launch angle obtained for the time-free case. By definition, the angle of attack for the ballistic case is identically 0.

Also of note is a comparison of the Hamiltonian for both the time-free and time-fixed cases (Figure 3.24). As mentioned earlier, the Hamiltonian will be zero for the time-free case; while it is small for time-fixed case, its non-zero value is indicative of the additional optimization to be realized if time were free). The fact that the Hamiltonian for the time-fixed case was already small, and that the final velocity did not change much are indications that time-fixed case was already very nearly an optimal solution.

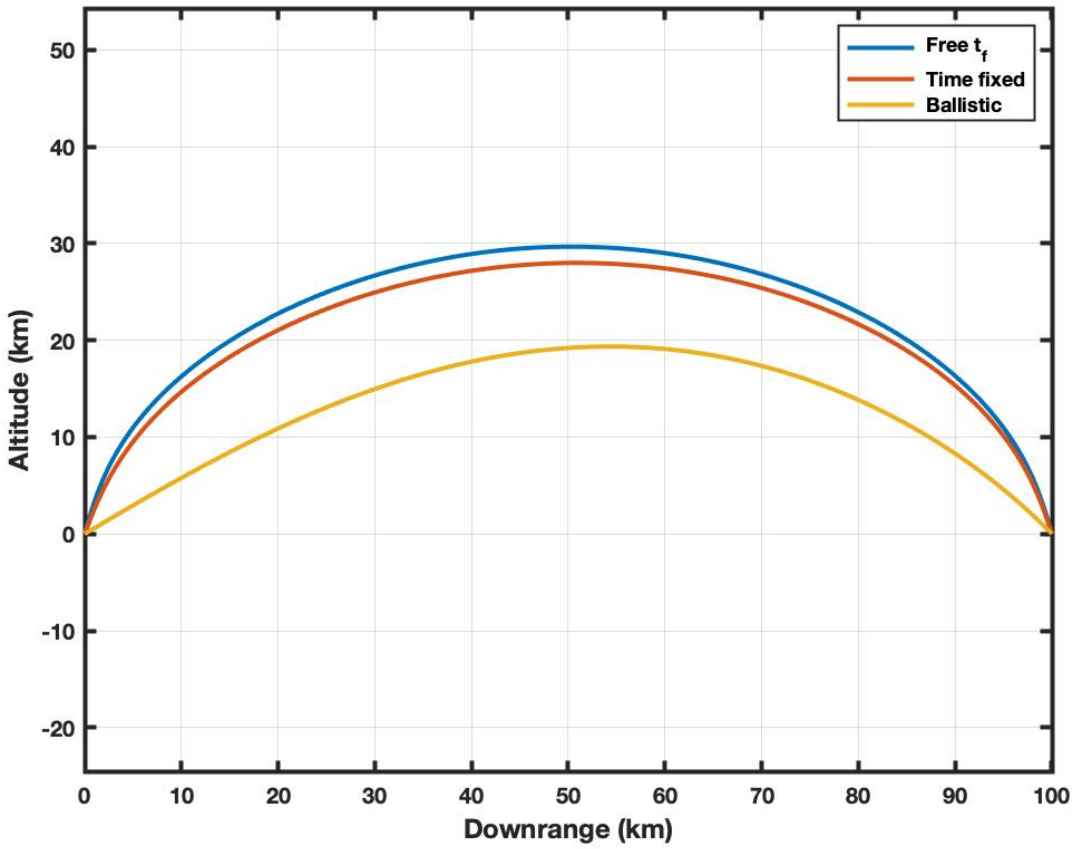

Figure 3.21: Comparison of Ballistic, Time-Fixed and Time-Free Trajectories

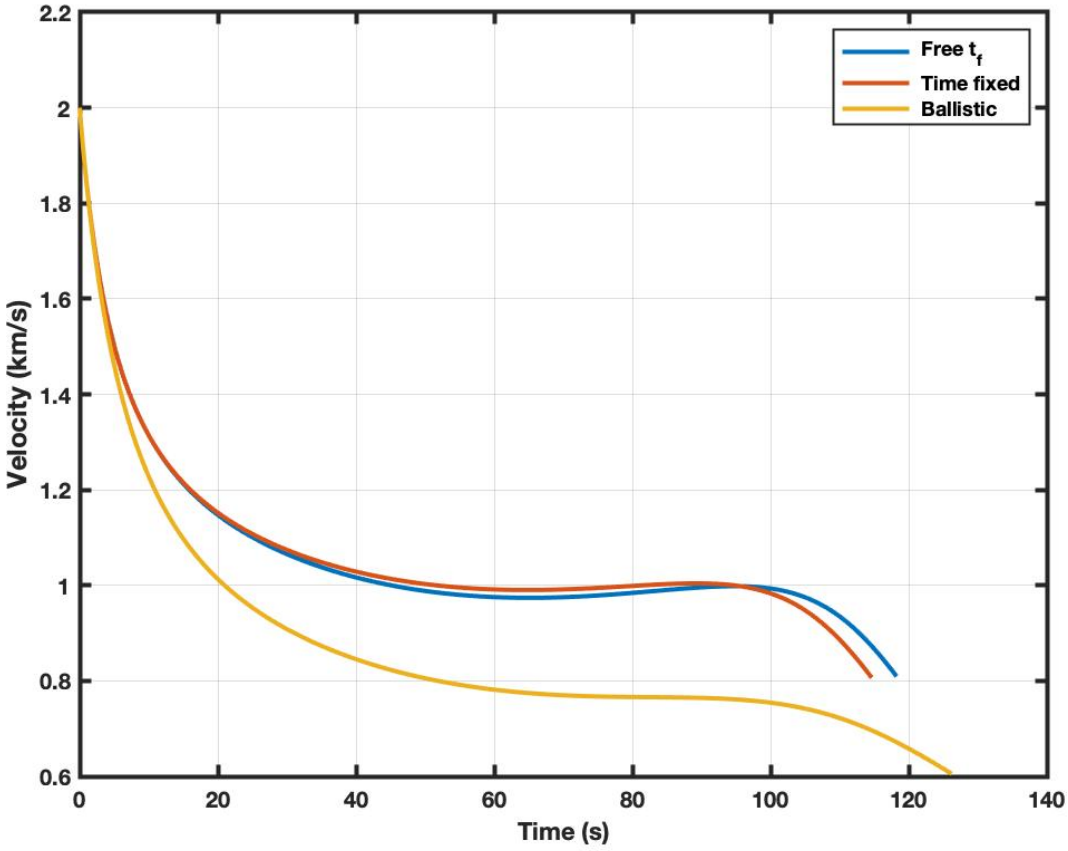

Figure 3.22: Comparison of Ballistic, Time-Fixed and Time-Free Velocity Histories

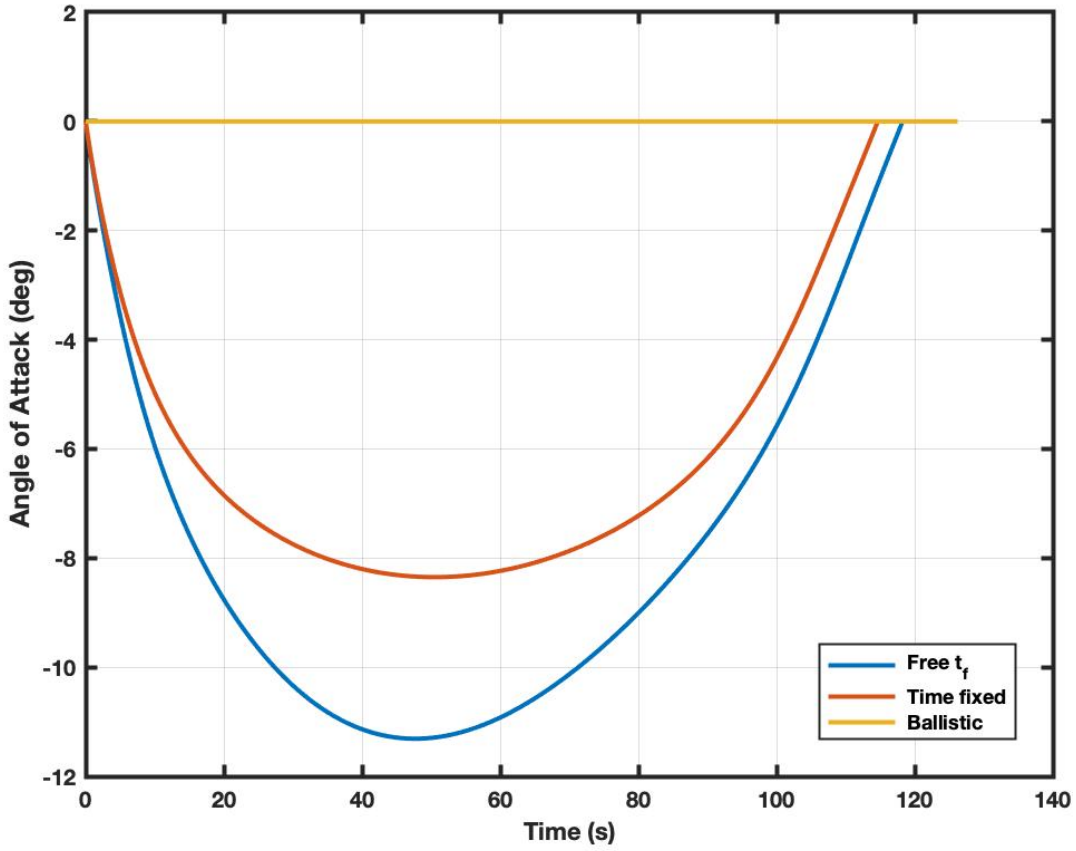

Figure 3.23: Comparison of Ballistic, Time-Fixed and Time-Free Angle-of-Attack Histories

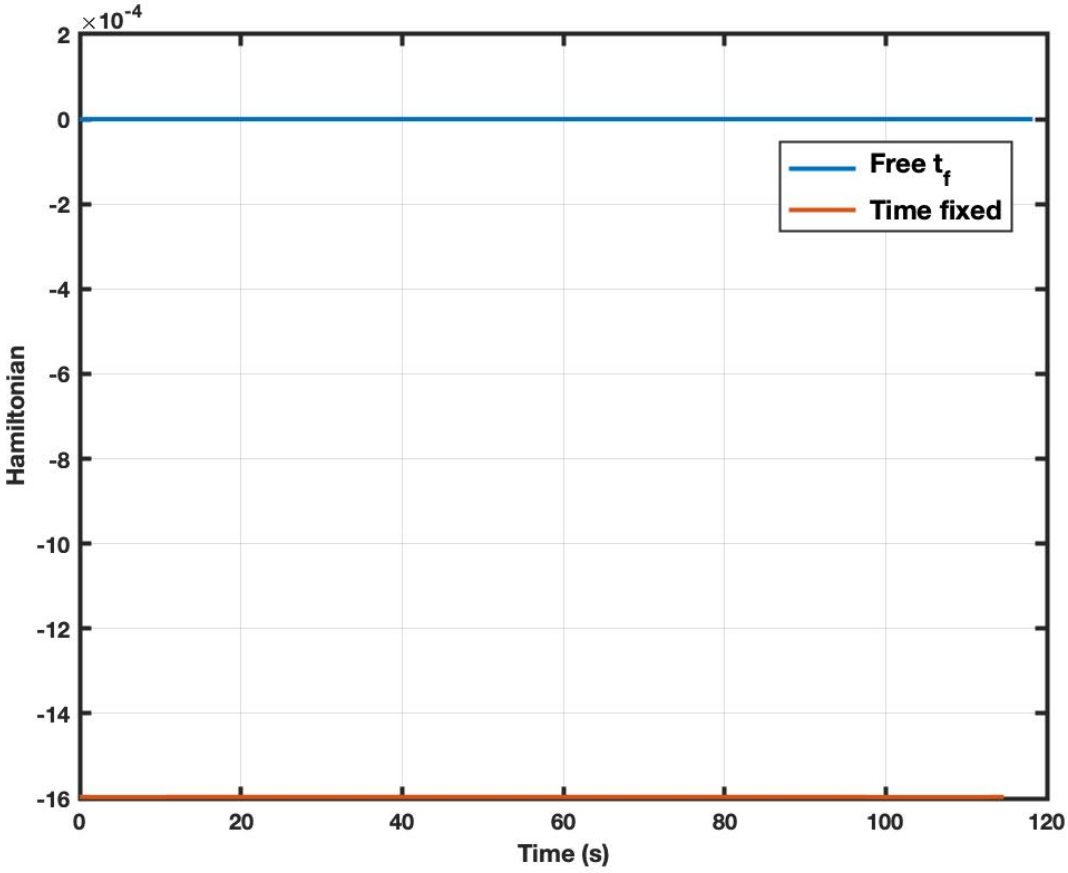

Figure 3.24: Comparison of the Hamiltonian for the Time-Fixed and Time-Free Trajectories

3.4 Costate Estimates from a Higher-Accuracy Costate Transition Matrix Computation

The method previously described used the finite-difference method from [30] to develop initial estimates for the costates. While in many cases this provides initial costate estimates sufficiently accurate to result in progression to an optimal solution, the estimates are still based on an approximation for $\boldsymbol{\nu}$. Another method was developed to compute a more accurate value of $\boldsymbol{\nu}$ for the given reference trajectory with the thought that this value of $\boldsymbol{\nu}$ would result in costate estimates that would be closer to the final converged values thus enabling a more likely path to an optimal solution.

This alternate method also relies on the reference trajectory, and integrating the states (position and velocity) forward in time as before, but now taking care to record their values at each of the w nodes. Upon completion, the costate trajectory was then integrated *backwards*. This was accomplished by using $\tau = t_f - t$ and making the state and costate dynamic equations equal to the negative of the original dynamics. That is, define:

$$(\ \ )' = \frac{d(\ \ )}{dt} \tag{3.1}$$

so that:

$$(\ \ )' = \frac{d(\ \ )}{dt}\frac{dt}{d\tau} = -\frac{d(\ \ )}{d\tau} \tag{3.2}$$

Therefore,

$$\boldsymbol{x}' = -\boldsymbol{f}(\boldsymbol{x}, \mathbf{u}, \tau) \tag{3.3}$$

and

$$\boldsymbol{\lambda}' = \left(\frac{\partial \mathcal{H}}{\partial \boldsymbol{x}}\right)^T = \left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right)^T \boldsymbol{\lambda} - \left(\frac{\partial L}{\partial \boldsymbol{x}}\right)^T \tag{3.4}$$

One of the difficulties of this method is that the states can be highly unstable[1] as τ increases (it is typical that as the costates are unstable integrating forward, the states are unstable integrating backwards). To mitigate this issue, the previously recorded state history was used to interpolate the values of the state at each τ; these values were then used in the dynamics equations for the costates. The costate transition matrix, Ψ, was set at $\tau = 0$ (i.e., $t = t_f$) to the identity matrix and was updated throughout the integration through the use of the costate dynamic equations (each column of Ψ consists of a unique value of $\boldsymbol{\lambda}$ based on different initial conditions). At each node, the value of $\mathbf{S}_i$ from Equation (2.38) is replaced by Ψ at that time. At the successful completion of the integration, the w values of $\mathbf{S}_i$ can be used in Equation (2.43) to compute a more accurate value of $\boldsymbol{\nu}$ *for that trajectory.* The values of the costates at each of the nodes would be calculated as discussed previously and the optimization process initiated.

To test this alternative method, a range extension case was chosen which did not converge using the original process. The alternate method was executed and an optimal solution was achieved. A comparison of the resulting initial costate estimates and those from the original methodology, along with the final converged values are shown in Figures 3.25 - 3.28. As can be seen, the costate estimates from this approach are closer to the final costate histories than the original approach; this is due to the fact that the backwards integration results in a more accurate value of $\boldsymbol{\nu}$ for the reference trajectory as opposed to the approximation obtained from the finite difference method. The main drawback of this methodology is that

[1] In the context of this discussion, "unstable" refers to the tendency to grow exponentially, resulting in the inability to achieve a converged solution.

it adds computational time because of the additional integration.

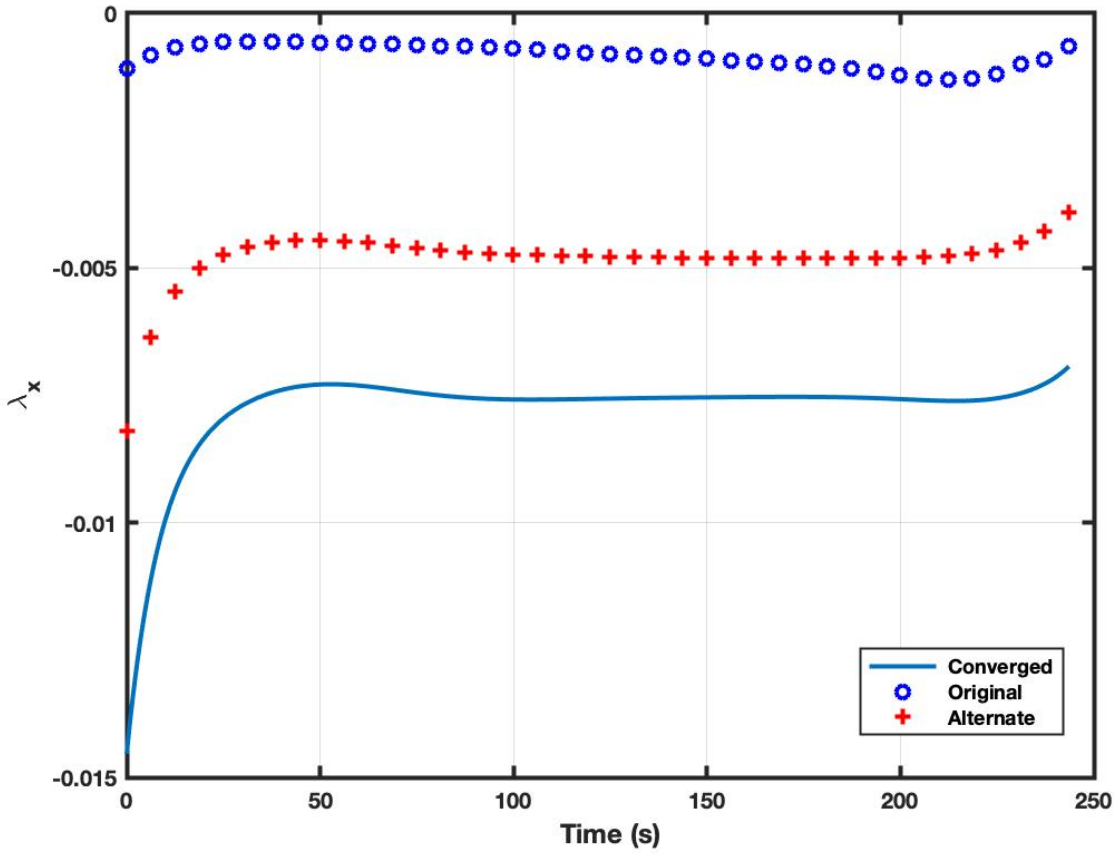

Figure 3.25: Comparison of λ_x values for Original and Alternate Methodologies

An indirect method to find the optimal angle of attack history that maximizes the impact velocity of a gun-launched guided projectile has been developed. We will now turn our attention to adding a minimum dynamic pressure constraint to this guided projectile application.

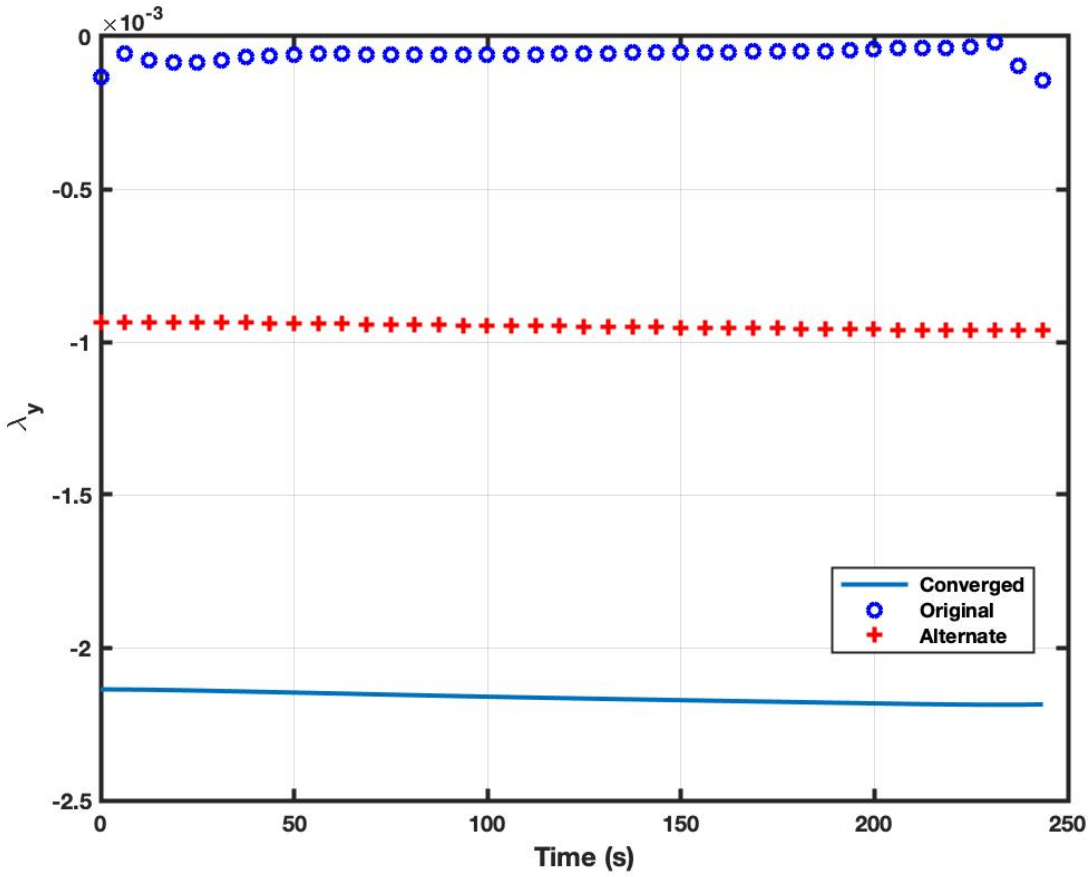

Figure 3.26: Comparison of λ_y values for Original and Alternate Methodologies

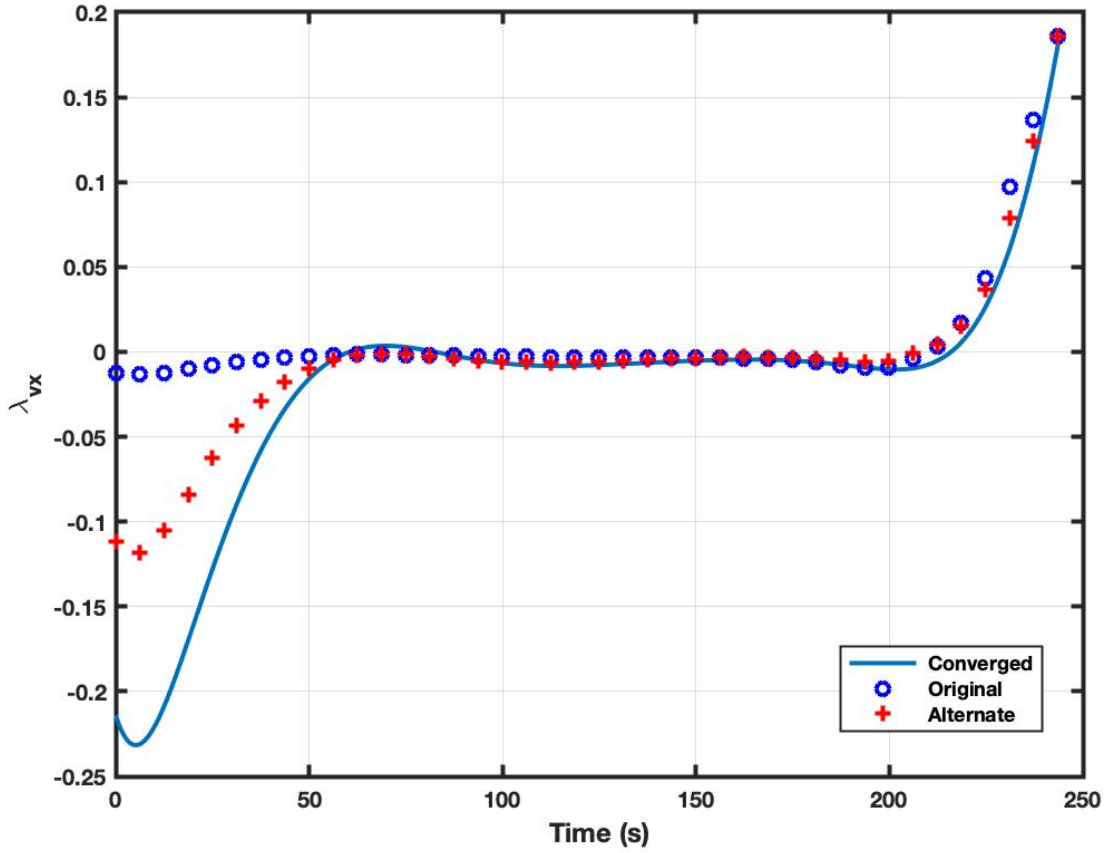

Figure 3.27: Comparison of λ_{v_x} values for Original and Alternate Methodologies

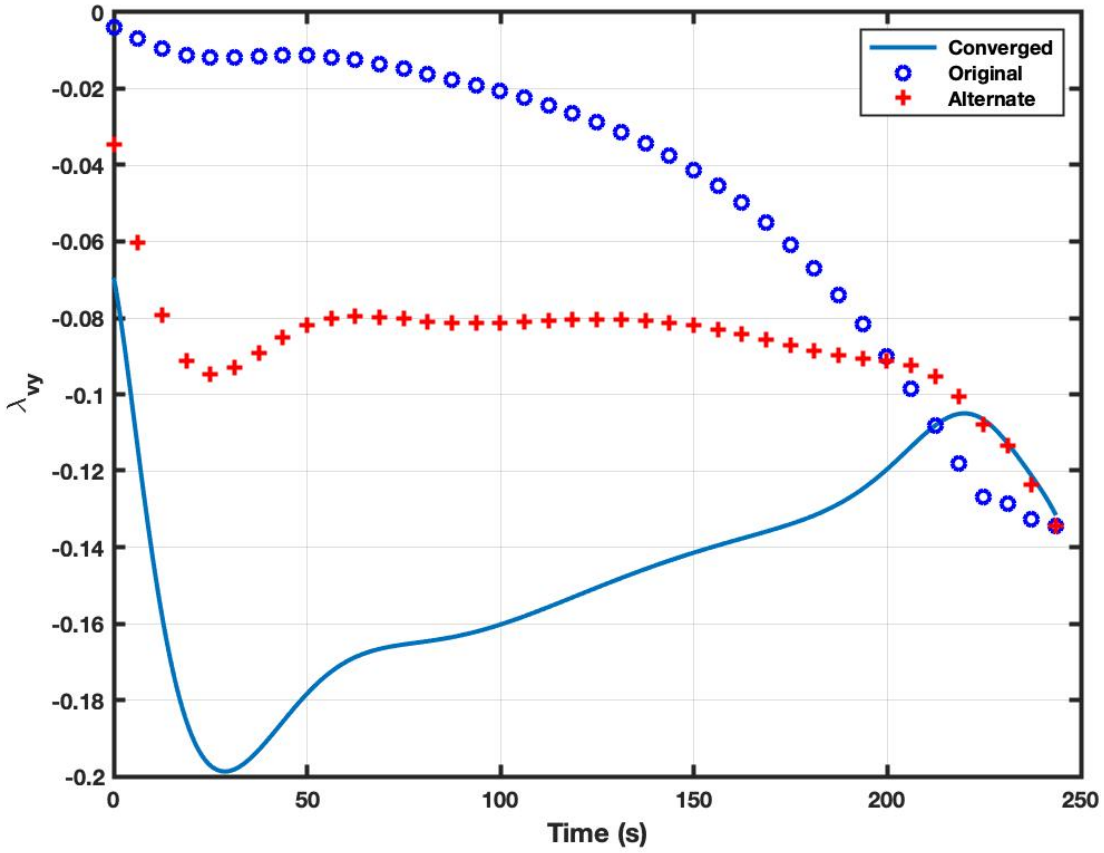

Figure 3.28: Comparison of λ_{v_y} values for Original and Alternate Methodologies

Chapter 4

Optimal Control with Minimum-Dynamic-Pressure Constraint

4.1 Minimum-Dynamic-Pressure Path Constraint

We now focus on extending the methodology previously discussed to include a minimum-dynamic-pressure inequality constraint. Clearly, an earth-surface-launched projectile has its highest dynamic pressure at launch. The vehicle trajectory will, therefore, have three segments – flying up to the constraint (while dynamic pressure is decreasing), a period of riding the constraint (maintaining the constraint as an equality constraint during this time period), then a final segment in which the dynamic pressure is again above the minimum value for the rest of the flight.

To determine the value of the control while on the constraint, we begin with the expression

for the Hamiltonian from Bryson and Ho [43][1] for inequality constraints on functions of the control variables:

$$\mathcal{H} = \boldsymbol{\lambda}^T \boldsymbol{f}(\boldsymbol{x}, \mathbf{u}) + \mu S^{(q)}(\boldsymbol{x}, \mathbf{u}) \tag{4.1}$$

where $\boldsymbol{\lambda}$ is the vector of costates, and $\boldsymbol{f}$ expresses the dynamics of the problem as a function of the state, $\boldsymbol{x}$, and the control, $\mathbf{u}$. $S^{(q)}(\boldsymbol{x}, \mathbf{u})$ is the q^{th} time derivative of $S(\boldsymbol{x})$ in which the control, α, first explicitly appears and μ is the influence function. It should be noted that $S^{(q)} = 0$ on the constraint ($S = 0$) and $\mu = 0$ off of the constraint ($S < 0$).

We define $S(\boldsymbol{x})$ to be the difference between the dynamic pressure (at any point on the trajectory) and the specified minimum dynamic pressure, q_{min}:

$$S(\boldsymbol{x}) = q_{min} - \frac{1}{2}\rho(h(\mathbf{r}))\mathbf{v}^T\mathbf{v}; \qquad S : \mathbb{R}^6 \to \mathbb{R} \tag{4.2}$$

where ρ is the density, h is the altitude, $\mathbf{r}$ is the position vector, and $\mathbf{v}$ is the velocity vector. We require that $S(x) \leq 0$ as our inequality constraint.

By definition, the time derivative of S on the constraint is zero; we differentiate Equation (4.2) with respect to time, using the straightforward application of the Product Rule and the Chain Rule, to get:

$$\dot{S}(\mathbf{x}) = -\frac{v^2}{2}\frac{\partial \rho}{\partial h}\frac{\partial h}{\partial \mathbf{r}}\frac{d\mathbf{r}}{dt} - \rho\mathbf{v}^T\dot{\mathbf{v}} \tag{4.3}$$

$$= -\frac{v^2}{2}\frac{\partial \rho}{\partial h}\hat{\mathbf{r}}\mathbf{v} - \rho\mathbf{v}^T\mathbf{a}(\mathbf{r}, \mathbf{v}, \alpha) \tag{4.4}$$

[1] This reference contains an excellent chapter on path constraints (Chapter 4).

which satisfies the requirement for $S^{(q)}$ since the acceleration, $\mathbf{a}$, is an explicit function of the control. Since the control explicitly shows up in the first time-derivative of $S(\boldsymbol{x})$, that makes this problem a first-order constraint. Both $S(\boldsymbol{x})$ and $\dot{S}(\boldsymbol{x})$ are equal to zero on the constraint.

In order to derive an expression for the dynamics of the costates, $\dot{\boldsymbol{\lambda}}$, we take the partial derivative of Equation (4.1) with respect to $\boldsymbol{x}$ to get:

$$\dot{\boldsymbol{\lambda}} = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{x}} = -\boldsymbol{\lambda}^T \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} - \mu \frac{\partial \dot{S}}{\partial \boldsymbol{x}}, \qquad \dot{S} = 0 \tag{4.5}$$

When off of the constraint (i.e., $S < 0$), the costate dynamics are determined as before using Equation (2.9).

To find an expression for μ in Equation (4.1), we take the partial derivative of the Hamiltonian with respect to the control to get:

$$\frac{\partial \mathcal{H}}{\partial \mathbf{u}} = \boldsymbol{\lambda}^T \frac{\partial \boldsymbol{f}}{\partial \mathbf{u}} + \mu \frac{\partial \dot{S}}{\partial \mathbf{u}} = 0 \tag{4.6}$$

We obtain the expression for $\frac{\partial \dot{S}}{\partial \mathbf{u}}$ by taking the partial derivative of Equation (4.4) with respect to $\mathbf{u}$:

$$\frac{\partial \dot{S}}{\partial \mathbf{u}} = -\rho \mathbf{v}^T \frac{\partial \mathbf{a}}{\partial \alpha} = -\rho \mathbf{v} \cdot \frac{\partial \mathbf{a}}{\partial \alpha} \tag{4.7}$$

Inserting Equation (4.7) into (4.6) and rearranging yields:

$$\mu = -\boldsymbol{\lambda}^T \frac{\partial f}{\partial \mathbf{u}} \left(\frac{\partial \dot{S}}{\partial \mathbf{u}} \right)^{-1} = \boldsymbol{\lambda}^T \frac{\partial f}{\partial \mathbf{u}} \left(\rho \mathbf{v} \cdot \frac{\partial \mathbf{a}}{\partial \alpha} \right)^{-1} \tag{4.8}$$

which can be used to integrate the costate dynamics in Equation (4.5) while on the constraint.

By definition, the $\boldsymbol{\lambda}'s$ will be discontinuous at the entry point to the constraint. If we assume that the constraint is entered at an arbitrary time, t_1, then the "jump" conditions for the costates are provided by:

$$\boldsymbol{\lambda}(t_1^-) = \boldsymbol{\lambda}(t_1^+) + \nu_s \frac{\partial S}{\partial \mathbf{x}}\Big|_{t_1}^{T}, \quad \nu_s \in \mathbb{R}^1 \tag{4.9}$$

The Hamiltonian, on the other hand, is continuous across the entry point since $\frac{\partial S}{\partial \mathbf{x}} = 0$:

$$\mathcal{H}(t_1^-) = \mathcal{H}(t_1^+) \tag{4.10}$$

To determine the angle of attack while on the minimum dynamic pressure constraint, we express the projectile acceleration, $\mathbf{a}$, as:

$$\mathbf{a} = \mathbf{a}_{aero} + \mathbf{g} \tag{4.11}$$

where $\mathbf{a}_{aero}$ is the acceleration due to aerodynamic forces and $\mathbf{g}$ is inverse-square gravity:

$$\mathbf{g} = -\frac{\mu_e \mathbf{r}}{|\mathbf{r}|^3} \tag{4.12}$$

Using Equation (2.58) for $\mathbf{a}_{aero}$, we use Equations (4.11) and (4.12) in Equation (4.4) to obtain:

$$\dot{S} = 0 = -\frac{v^2}{2}\frac{\partial \rho}{\partial h}\hat{\mathbf{r}}\mathbf{v} - \rho \mathbf{v} \cdot \Big[(-a_N \sin\alpha - a_A \cos\alpha)\hat{\mathbf{v}} + (a_N \cos\alpha - a_A \sin\alpha)\hat{\mathbf{u}}\Big] + \rho \mathbf{v}^T \frac{\mu_e \mathbf{r}}{|\mathbf{r}|^3} \tag{4.13}$$

$$= -\frac{v^2}{2}\frac{\partial \rho}{\partial h}\dot{h} - \rho \mathbf{v} \cdot \Big[(-a_N \sin\alpha - a_A \cos\alpha)\hat{\mathbf{v}} + (a_N \cos\alpha - a_A \sin\alpha)\hat{\mathbf{u}}\Big] + \rho\mu_e \frac{\dot{h}}{|\mathbf{r}|^2} \tag{4.14}$$

$$= \Big(\frac{\rho\mu_e}{|\mathbf{r}|^2} - \frac{v^2}{2}\frac{\partial \rho}{\partial h}\Big)\dot{h} + \rho v(a_N \sin\alpha + a_A \cos\alpha) \tag{4.15}$$

to which Newton's method can be applied to solve for α (as we did in Section 2.4). To do so, we begin by recognizing that $a_N = \frac{N_\alpha}{m}\alpha$ and $a_A = \frac{A}{m}$ and substitute in Equation (4.16) to obtain:

$$F(\alpha) = \dot{S} = \Big(\frac{\rho\mu_e}{|\mathbf{r}|^2} - \frac{v^2}{2}\frac{\partial \rho}{\partial h}\Big)\dot{h} + \rho v\Big(\frac{N_\alpha}{m}\alpha \sin\alpha + \frac{A}{m}\cos\alpha\Big) \tag{4.16}$$

We then take the partial derivative with respect to α:

$$F'(\alpha) \triangleq \frac{\partial F(\alpha)}{\partial \alpha} = \rho v\Big(\frac{N_\alpha}{m}\sin\alpha + \frac{N_\alpha}{m}\alpha\cos\alpha - \frac{A}{m}\sin\alpha\Big) \tag{4.17}$$

As before, each new estimate for the α is given by:

$$\alpha_{k+1} = \alpha_k - \frac{F(\alpha_k)}{F'(\alpha_k)} \tag{4.18}$$

where α_{k+1} is the new estimate and α_k is the previous estimate. We iterated until the difference between the new and previous estimates is below some predetermined threshold. Upon completion, the value of α_{k+1} is the solution to Equation (4.16).

To obtain an initial first guess to initiate the Newton's method iteration, we linearize (small

angle approximation) Equation (4.16) with respect to α to obtain:

$$\tilde{F}(\alpha_0) = \Big(\frac{\rho\mu_e}{|\mathbf{r}|^2} - \frac{v^2}{2}\frac{\partial\rho}{\partial h}\Big)\dot{h} + \rho v\Big(\frac{N_\alpha}{m}{\alpha_0}^2 + \frac{A}{m}\Big) = 0 \tag{4.19}$$

By rearranging

$$\rho v\frac{N_\alpha}{m}\alpha_0^2 = -\Big(\frac{\rho\mu_e}{|\mathbf{r}|^2} - \frac{v^2}{2}\frac{\partial\rho}{\partial h}\Big)\dot{h} - \rho v\frac{A}{m} \tag{4.20}$$

and solving for α we obtain:

$$\alpha_0 = \pm\sqrt{\frac{1}{N_\alpha}\left[\frac{-m}{\rho v}\Big(\frac{\rho\mu_e}{|\mathbf{r}|^2} - \frac{v^2}{2}\frac{\partial\rho}{\partial h}\Big)\dot{h} - A\right]} \tag{4.21}$$

which provides the initial guess we require. As to which sign to use, it can be observed that the same level of induced drag, (which is essentially the mechanism by which the constraint is enforced), is generated with either α or $-\alpha$. Therefore, to reduce the chance of an extreme jump in the control at t_1, the sign on α_0 was chosen to be the same as the sign on the angle of attack calculated at the same point in time on the unconstrained arcs (using Equations (2.65) to (2.69)). For all the numerical solutions obtained so far, the sign on α has always been positive.

We gain insight into when control on the constraint can be achieved by examining Equation (4.16). Since $\dot{S} = 0$, we can write:

$$\Big(\frac{\rho\mu_e}{|\mathbf{r}|^2} - \frac{v^2}{2}\frac{\partial\rho}{\partial h}\Big)\dot{h} = -\rho v\Big(\frac{N_\alpha}{m}\alpha\sin\alpha + \frac{A}{m}\cos\alpha\Big) \tag{4.22}$$

or rearranging:

$$\dot{h} = -\frac{\rho v\left(\frac{N_\alpha}{m}\alpha\sin\alpha + \frac{A}{m}\cos\alpha\right)}{\left(\frac{\rho\mu_e}{|\mathbf{r}|^2} - \frac{v^2}{2}\frac{\partial\rho}{\partial h}\right)} \tag{4.23}$$

Since $\frac{\partial\rho}{\partial h}$ is always negative, and all of the other terms in the denominator are positive, the denominator is positive. Also, for $|\alpha| < 90°$, $\alpha\sin\alpha$ and $\cos\alpha$ are also positive. Therefore, $\dot{h}$ is strictly negative on the constraint, which indicates that the entry onto the constraint can only occur *after* apogee.

4.2 Minimum-Dynamic-Pressure Point Constraint

It seems natural to consider whether the optimal solution of a minimum-dynamic-pressure inequality constraint might be a touch-point constraint (that is, the constraint being "touched" at one moment in time only, as opposed to entering the constraint, riding it for finite time, then exiting it). That is, among all candidate solutions, might a touch point solution be optimal? This possibility seems plausible, since dynamic pressure decreases due to decreasing velocity as altitude increases, as well as with decreasing atmospheric density as altitude increases. Intuitively, the minimum dynamic pressure should occur around apogee of the flight path.

Consider the interior point solution described, for example, by Bryson and Ho [44]. The jump conditions for the adjoint variables are given by:

$$\boldsymbol{\lambda}(t_1)^+ = \boldsymbol{\lambda}(t_1)^- + \frac{\partial \boldsymbol{N}}{\partial \boldsymbol{x}}^T \boldsymbol{\nu}_s, \quad \boldsymbol{\nu}_s \in \mathbb{R}^2 \tag{4.24}$$

where $\boldsymbol{N}(\boldsymbol{x})$ is defined as:

$$\boldsymbol{N}(\boldsymbol{x}) = \begin{bmatrix} S(\boldsymbol{x}) \\ \dot{S}(\boldsymbol{x}) \end{bmatrix} \tag{4.25}$$

and $\frac{\partial \boldsymbol{N}}{\partial \boldsymbol{x}}^T$ is given by:

$$\frac{\partial \boldsymbol{N}}{\partial \boldsymbol{x}}^T = \begin{bmatrix} \frac{\partial S}{\partial \boldsymbol{r}}^T & \frac{\partial \dot{S}}{\partial \boldsymbol{r}}^T \\ \frac{\partial S}{\partial \boldsymbol{v}}^T & \frac{\partial \dot{S}}{\partial \boldsymbol{v}}^T \end{bmatrix} \tag{4.26}$$

The conditions at the touch point are such that:

$$S(t_1) = 0, \quad \dot{S}(t_1) = 0, \quad \text{and } \Delta\mathcal{H}(t_1) = 0 \tag{4.27}$$

Given that:

$$S = q_{min} - \frac{1}{2}\rho \boldsymbol{v} \cdot \boldsymbol{v} \tag{4.28}$$

we can derive the four components of Equation (4.26) as:

$$\frac{\partial S}{\partial \boldsymbol{r}}^T = -\frac{1}{2}\frac{\partial \rho}{\partial h} v^2 \hat{\boldsymbol{r}} \tag{4.29}$$

$$\frac{\partial S}{\partial \boldsymbol{v}}^T = -\rho \boldsymbol{v} \tag{4.30}$$

Using Equation (4.4):

$$\frac{\partial \dot{S}}{\partial \boldsymbol{r}} = -\frac{1}{2}\left[\left[\frac{\partial^2 \rho}{\partial h^2}\hat{\boldsymbol{r}}^T(\hat{\boldsymbol{r}}^T\boldsymbol{v}) + \frac{\partial \rho}{\partial h}\frac{1}{\boldsymbol{r}^3}\left[\boldsymbol{r}^2 I - \boldsymbol{r}\boldsymbol{r}^T\right]\cdot \boldsymbol{v}\right]\boldsymbol{v}^T\boldsymbol{v}\right] - \frac{\partial \rho}{\partial h}\boldsymbol{r}^T\boldsymbol{v}^T\boldsymbol{a} \tag{4.31}$$

and

$$\frac{\partial \dot{S}}{\partial \boldsymbol{v}} = -\frac{1}{2}\frac{\partial \rho}{\partial h}\Big[\boldsymbol{v}^2 I + 2\boldsymbol{v}\boldsymbol{v}^T\Big] - \rho \boldsymbol{a} \tag{4.32}$$

However, during the course of implementing this approach, it became apparent that the minimum dynamic pressure constraint could not work as an interior point constraint. Solutions were obtained that satisfied the requirements of the Multiple Shooting problem, but, immediately after the constraint was touched, $\dot{S}$ was always strictly negative, causing the constraint to be immediately violated.

The mathematical unlikelihood of touchpoints (other than a *natural* touchpoint) was discussed by Seywald and Cliff [45] confirming the results we were observing. Since every trajectory already has a minimum dynamic pressure, one could select this value of q_{min} as the minimum value; the single condition where the projectile encountered that exact value would be a natural touchpoint. This is unlikely to occur under realistic conditions; therefore, the projectile must ride the constraint for a finite amount of time, indicating that this constraint must be a path constraint.

4.3 Application of the Methodology with Constraints

We implement the constraint by dividing the trajectory into three segments (see Figure 4.1). In the first segment, the projectile flies from launch at t_0, using the unconstrained control and costate dynamics as defined in Chapter 2 until it hits the user-defined value of q_{min} at t_1. At this point, the control is switched to the value derived in Section 4.1 and the projectile rides the constraint (i.e., α is manipulated such that the projectile maintains a constant value of $q = q_{min}$) until the minimum dynamic pressure is exceeded again at t_2. At this

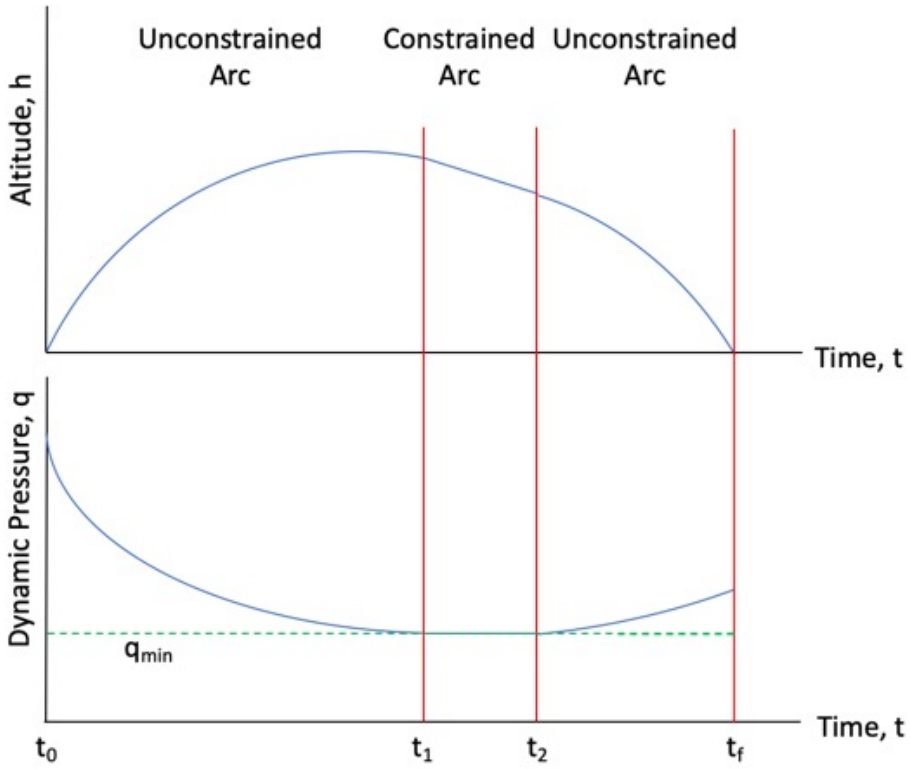

Figure 4.1: Schematic of Trajectory and Dynamic Pressure

point, the projectile comes off of the constraint and resumes control as on the first segment, impacting the ground at t_f.

4.4 Multiple Shooting Method with the Path Constraint

To incorporate the constraint in the methodology developed in Section 2, the independent vector, $\boldsymbol{p}$, and the dependent vector, $\boldsymbol{F}(\boldsymbol{p})$, were modified to include the additional parameters used to characterize the constraint. As before, a nonlinear system-of-equations solver was employed to obtain the costate estimates; the solver manipulated $\boldsymbol{p}$ to drive each element

of $\boldsymbol{F}(\boldsymbol{p})$ to zero (within a user-defined tolerance). Upon convergence of the solver, the final elements of $\boldsymbol{p}$ are the solution to the system of linear equations.

The independent vector, $\boldsymbol{p} \in \mathbb{R}^{n_{\boldsymbol{p}}}$, where $n_{\boldsymbol{p}} = 4(w+1)$, is defined by:

$$\boldsymbol{p} = \left\{ \begin{array}{c} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \\ : \\ \boldsymbol{\lambda}_{w-1} \\ \boldsymbol{\nu} \\ \nu_0 \\ \theta_0 \\ \nu_s \\ t_1 \\ t_2 \\ t_f \end{array} \right\} \tag{4.33}$$

where the first $4w$ terms are defined as in Equation (2.70) with additional terms included to accommodate the constraint. The terms t_1 and t_2 refer to the times that the projectile enters and leaves the constraint, respectively, t_f is the final time, while ν_s is used to satisfy the jump conditions in the costates at the entrance of the constraint.

The corresponding dependent vector $\boldsymbol{F}(\boldsymbol{p}) : \mathbb{R}^{n_{\boldsymbol{p}}} \to \mathbb{R}^{n_{\boldsymbol{p}}}$, is given by:

$$\boldsymbol{F}(\boldsymbol{p}) = \left\{ \begin{array}{c} \Delta\boldsymbol{\lambda}_2 \\ \Delta\boldsymbol{\lambda}_3 \\ : \\ \Delta\boldsymbol{\lambda}_{w-1} \\ \Delta\boldsymbol{\lambda}_w \\ \Delta\mathbf{r} \\ \lambda_{v_x,0} - \nu_0 * \cos\theta_0 \\ \lambda_{v_y,0} - \nu_0 * \sin\theta_0 \\ \frac{S(t_1)}{q_{min}} \\ \mathcal{H}(t_1)^+ - \mathcal{H}(t_1)^- \\ \dot{S}(t_1) \\ \mathcal{H} \end{array} \right\} \tag{4.34}$$

where again the first $4w$ terms are identical to those used in Equation (2.71). The remaining terms are added to accommodate the fact that $S(t_1)$ and $\dot{S}(t_1)$ along with the Hamiltonian, must all be zero. The $\mathcal{H}(t_1)^+ - \mathcal{H}(t_1)^-$ term ensures that there is no jump in the Hamiltonian while crossing the entry onto the constraint. Ross, et al., discuss scaling and balancing in [46]; in this case, $\Delta\mathbf{r}$ was scaled by dividing by 100, $\dot{S}$ was scaled by dividing by 50000, and $\Delta\mathcal{H}$ and $\mathcal{H}$ were scaled by multiplying by 100.

4.5 Optimal Control Results with a Path Constraint

To demonstrate the methodology, we begin with a trajectory (range = 30 km, v_0 = 1 km/s) which has already been optimized to maximize final velocity. An arbitrary q_{min} of 79000 Pa

was selected (this was close to the minimum q on the trajectory with no constraint) and the new constrained optimization code was executed. A comparison of the two cases is shown in Figures 4.2 - 4.8. Figures 4.2 and 4.3 indicate slight variations between the constrained and unconstrained position and velocity histories, with the constrained trajectory flying higher earlier in the flight and in a more "shallow" manner in the second half of the flight and with both trajectories achieving roughly the same final velocity.

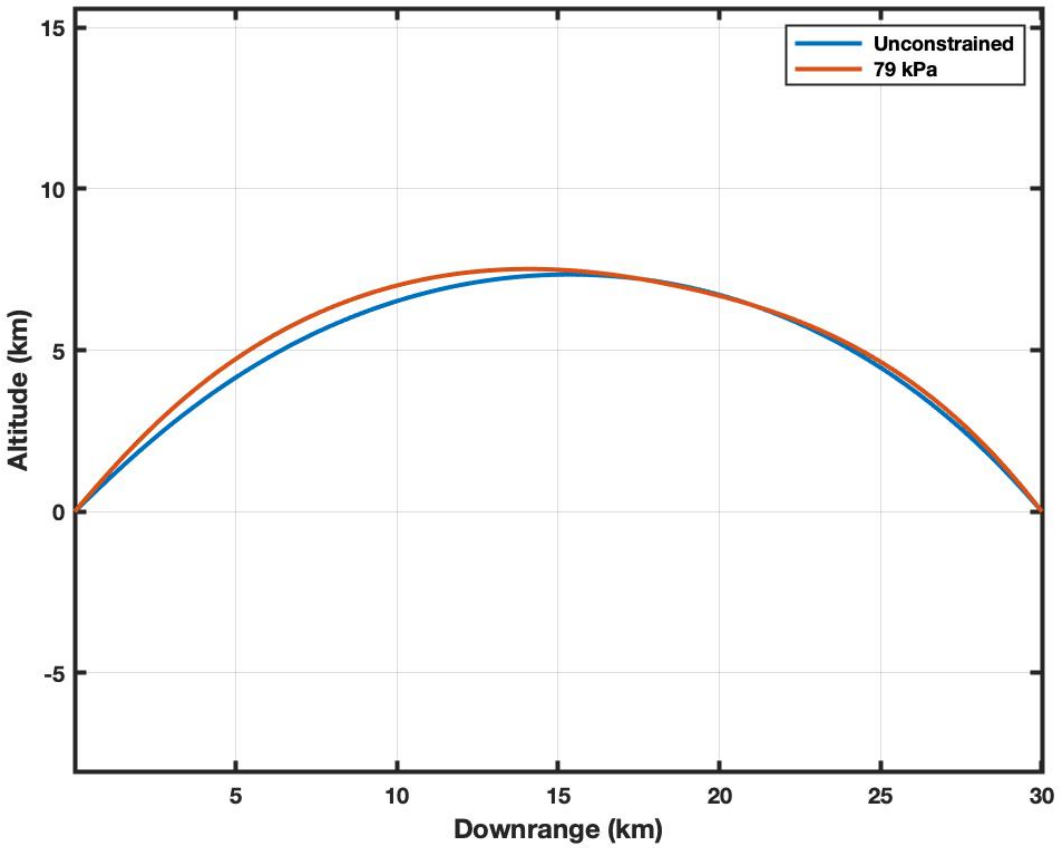

Figure 4.2: Unconstrained and Constrained Altitude vs. Downrange

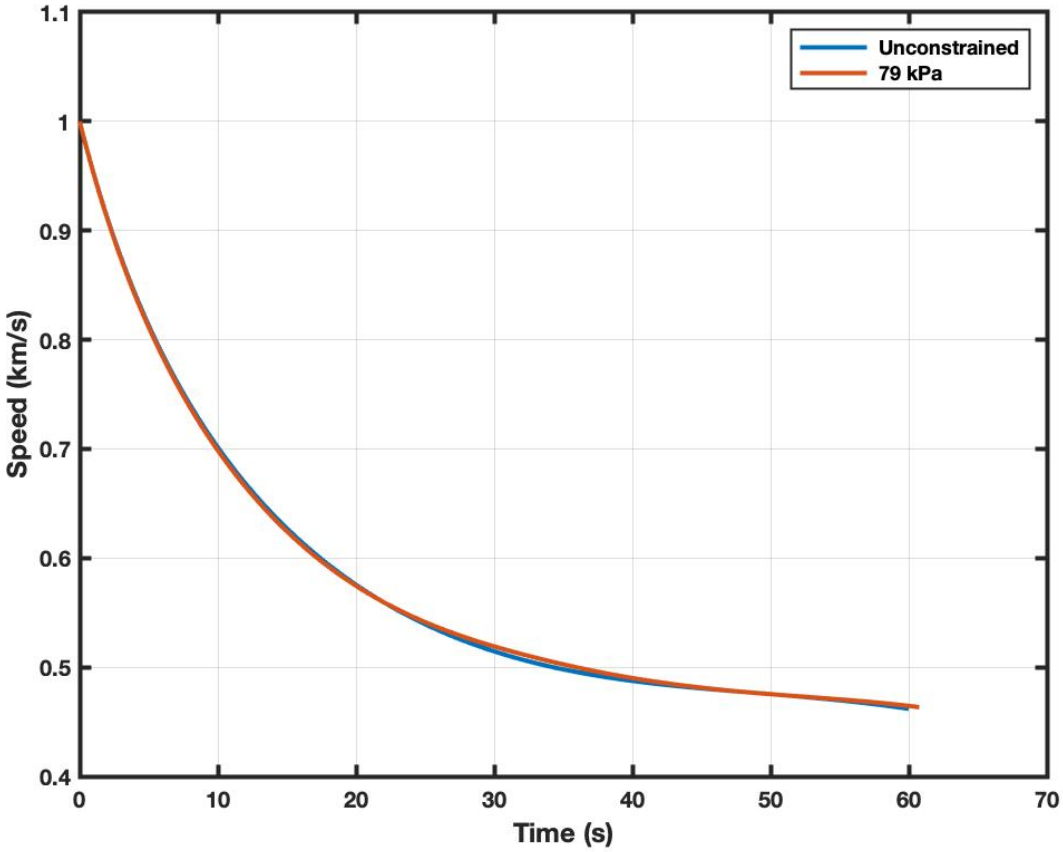

Figure 4.3: Unconstrained and Constrained Velocity Histories

Figure 4.4 shows a comparison of the unconstrained and constrained angles of attack; Figure 4.5 is an enlarged view of the constrained angle of attack prior to, during and subsequent to the constrained arc. The angle of attack for the constrained case is determined in such a way as to enter the constraint at approximately 32.2 seconds into the flight and return off of the constraint approximately 0.1 seconds later. The dynamic pressure histories of the cases are shown in Figure 4.6; two expanded views of the dynamic pressure plots are shown in Figures 4.7 - 4.8. Figure 4.7 shows the low points of the dynamic pressure traces for the two trajectories, while Figure 4.8 shows the constrained trajectory entering and leaving the constraint at the times corresponding to the angle of attack changes. It should be noted

that, as intended, the dynamic pressure stays constant while on the constraint.

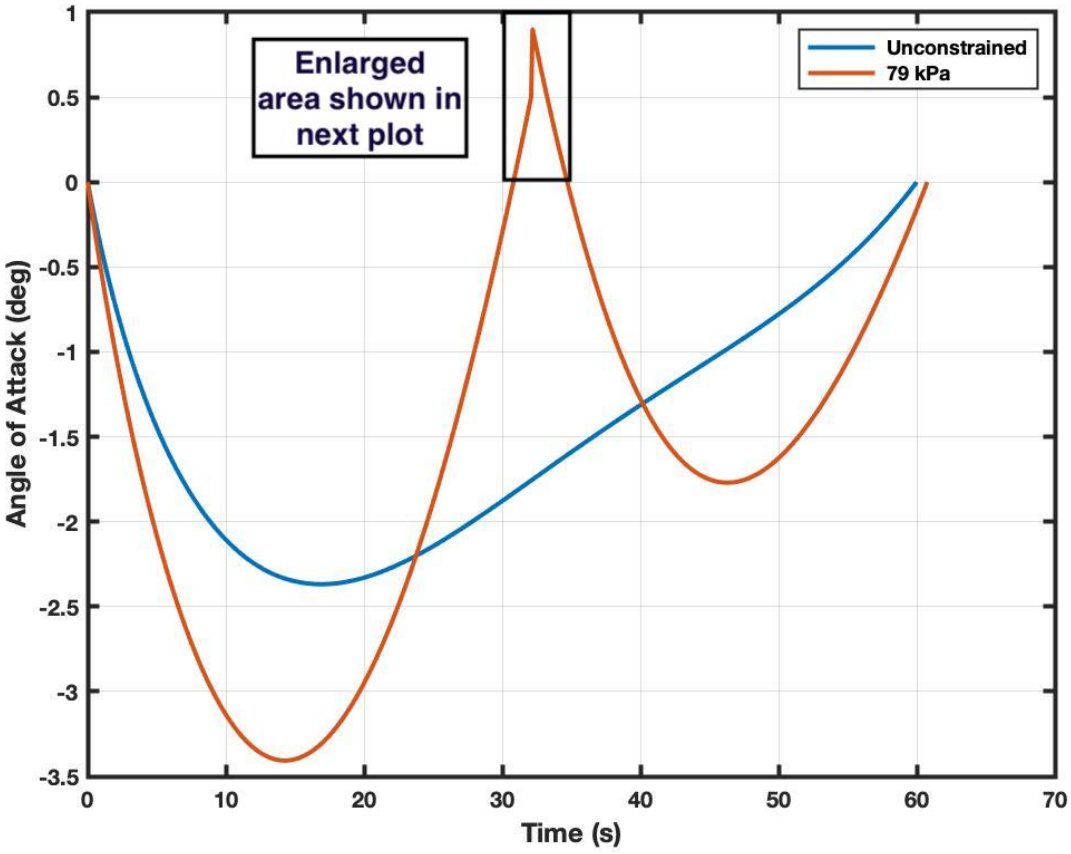

Figure 4.4: Unconstrained and Constrained Angle of Attack Histories

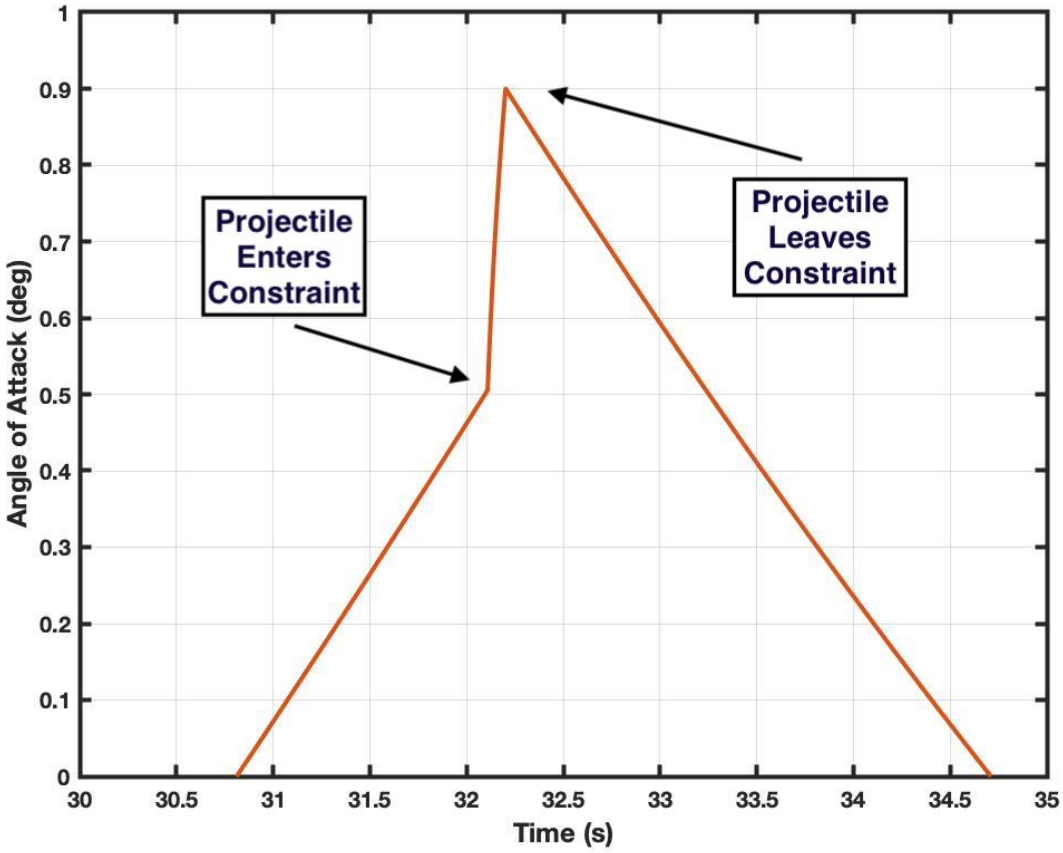

Figure 4.5: Zoomed-in View of Angle of Attack History on Constrained Trajectory

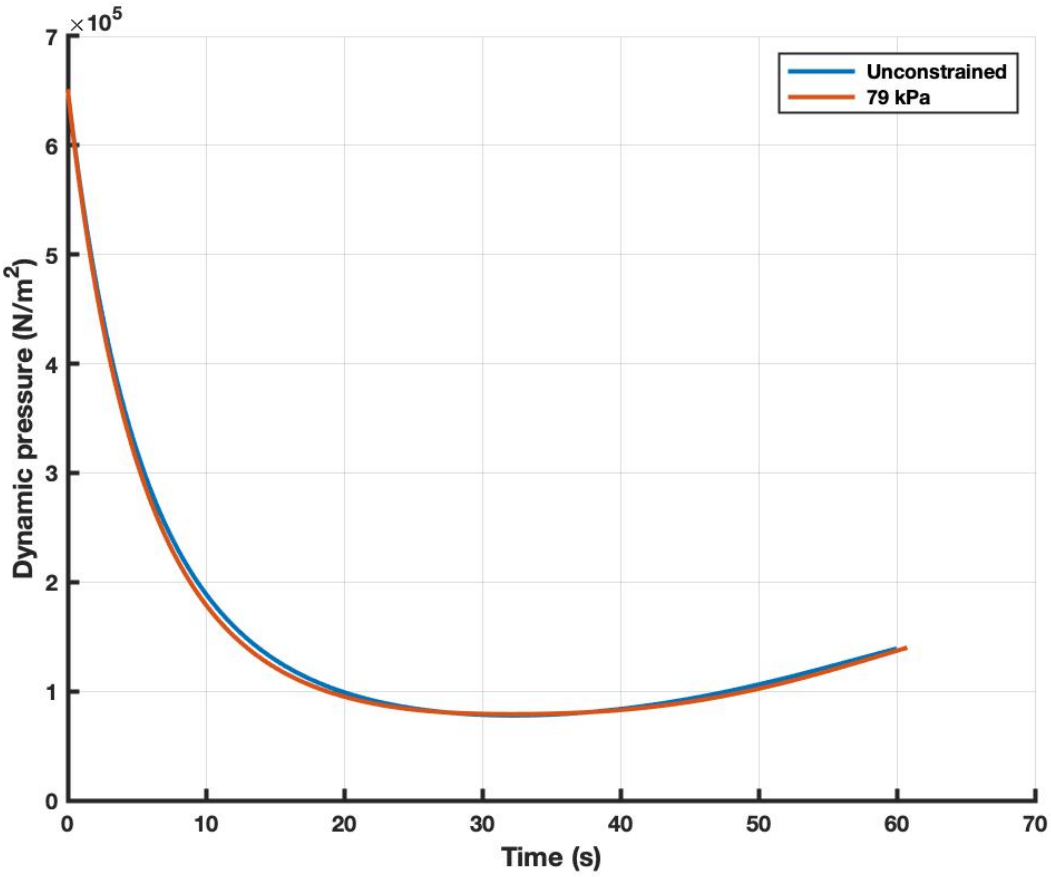

Figure 4.6: Unconstrained and Constrained Dynamic Pressure Histories

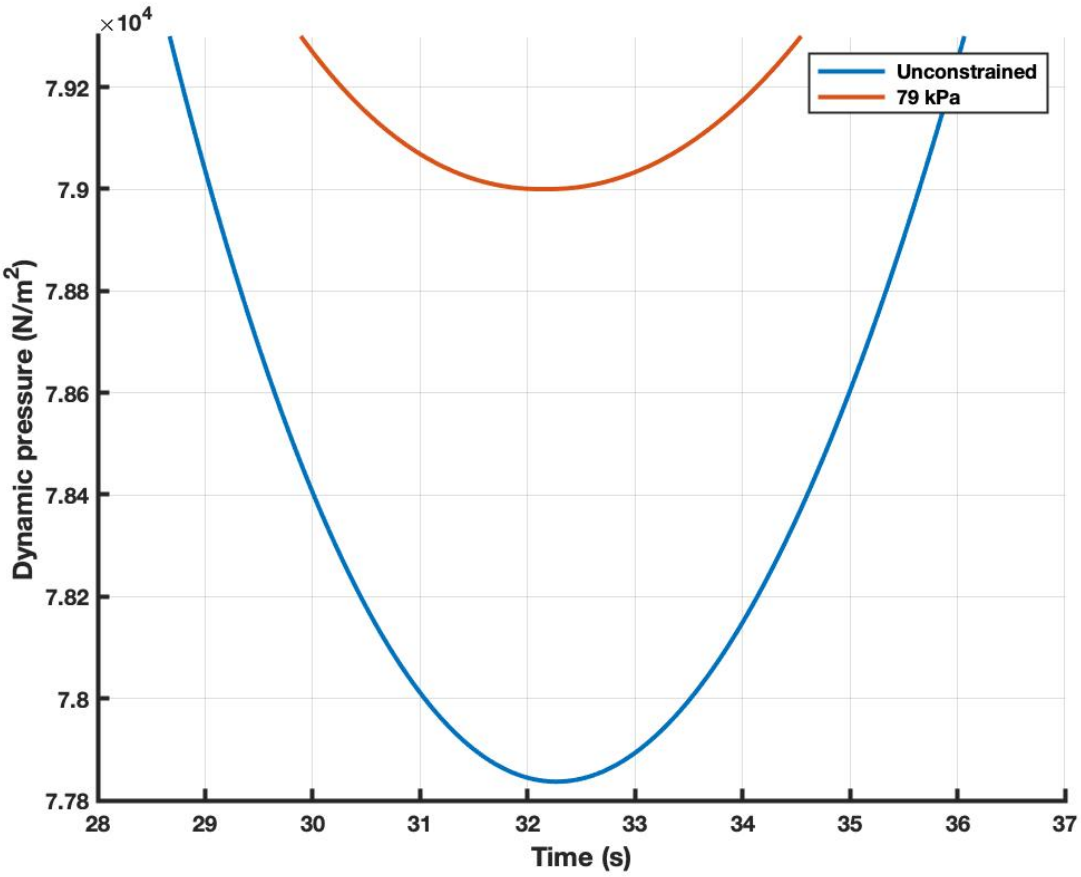

Figure 4.7: Enlarged View of Dynamic Pressure Histories

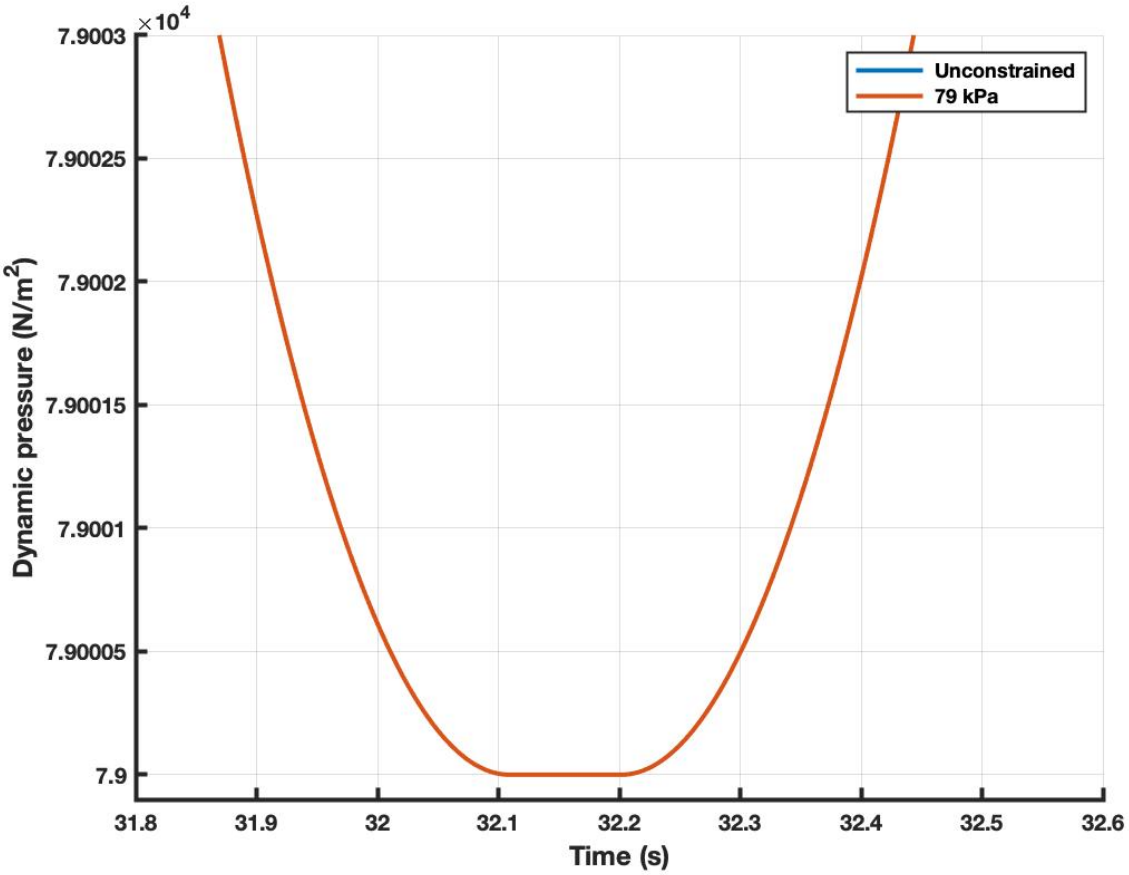

Figure 4.8: Zoomed-in View of Constrained Trajectory Showing the Constraint

At this point, successively higher values for the minimum dynamic pressure (up to 110 kPa) were chosen and executed. The trajectory and velocity histories of these cases are shown in Figures 4.9 and 4.10. An interesting feature of note in Figure 4.9 is the tendency for the optimization process to move the apogee of the trajectory further towards launch (i.e., earlier in time) as q_{min} is increased. Figure 4.10 shows the significant advantage of optimization; even though the constraint is becoming more difficult to achieve as q_{min} is increased, the final velocity is still being preserved as per the cost function. Figure 4.11 shows the corresponding angle-of-attack histories. As q_{min} is increased, positive angles of attack are held almost constant for successively longer times, causing the projectile to "ride"

the constraint. This results in the long, almost linear shape on the tail end of the trajectories shown in Figure 4.9. Figure 4.12 is a comparison of the dynamic pressure histories; Figure 4.13 is an expanded view of Figure 4.12 highlighting the behavior on the constrained portion of the trajectories. As can be seen, the higher the value for the constraint, the longer the projectile rides it. At 110 kPa, the projectile enters the constraint at just before 30 seconds and remains on it for almost the remainder of the flight, coming off of the constraint just before impact.

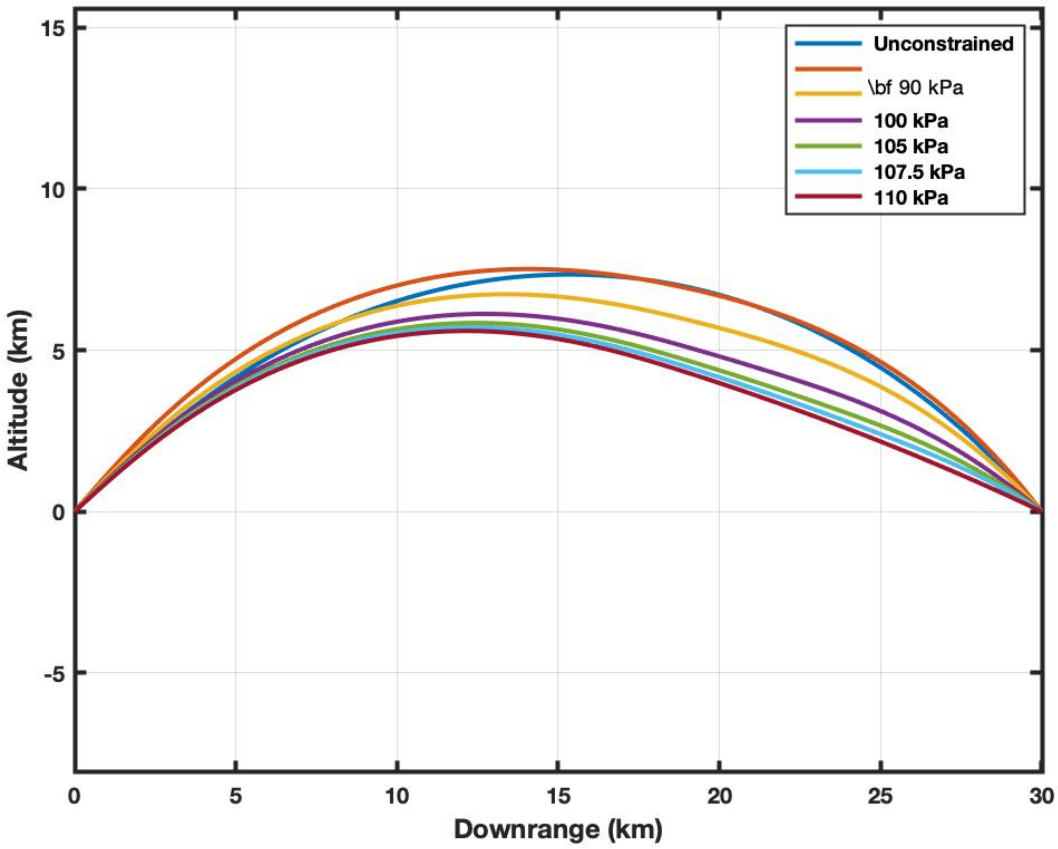

Figure 4.9: Position Histories of Constrained Trajectories

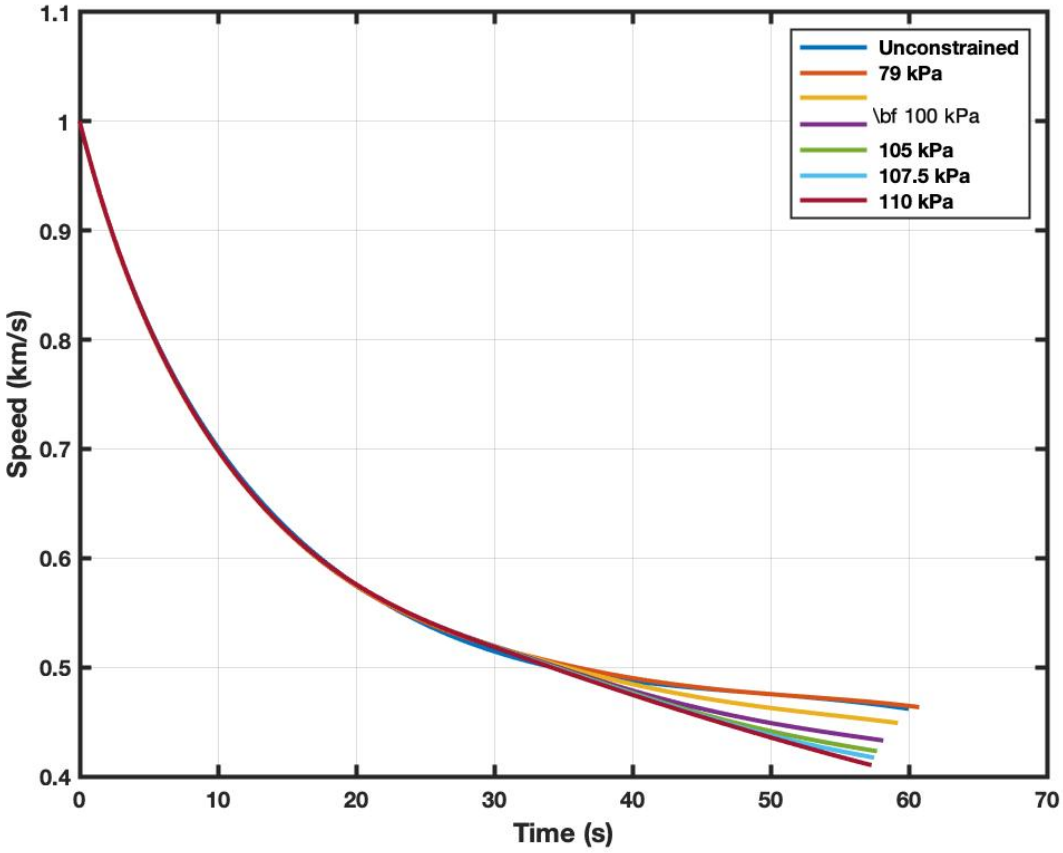

Figure 4.10: Velocity Histories of Constrained Trajectories

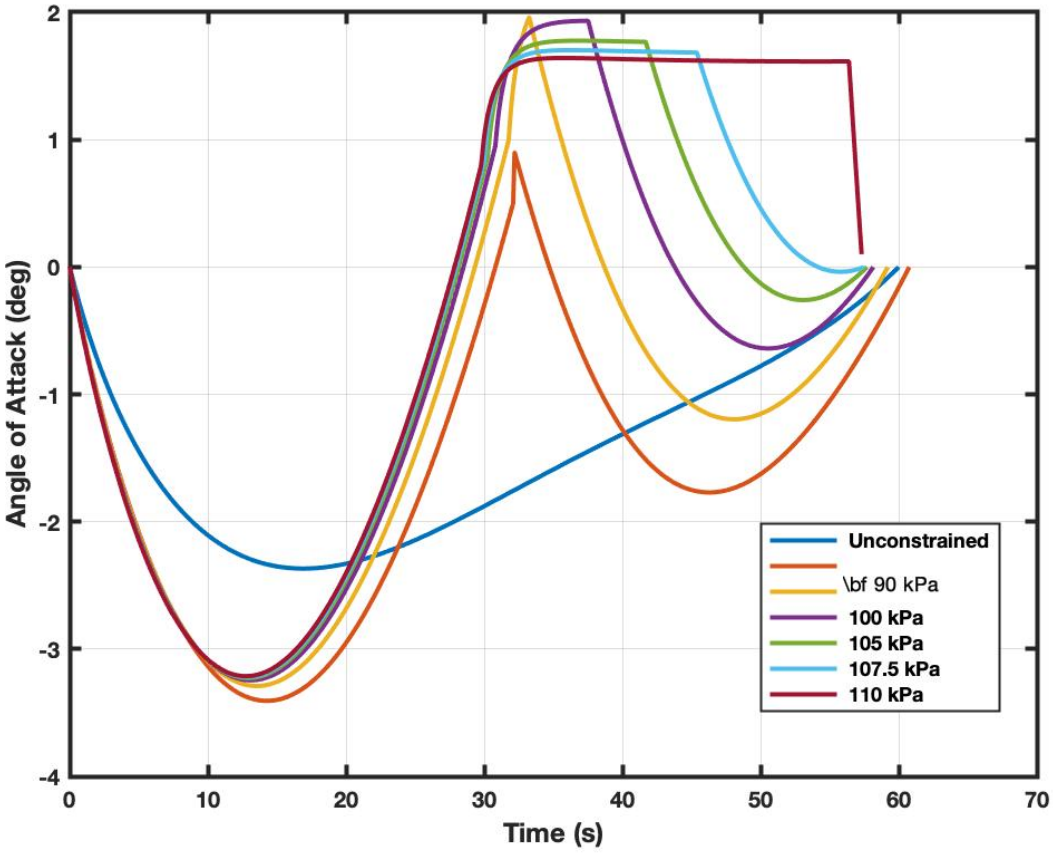

Figure 4.11: Angle-of-Attack Histories of Constrained Trajectories

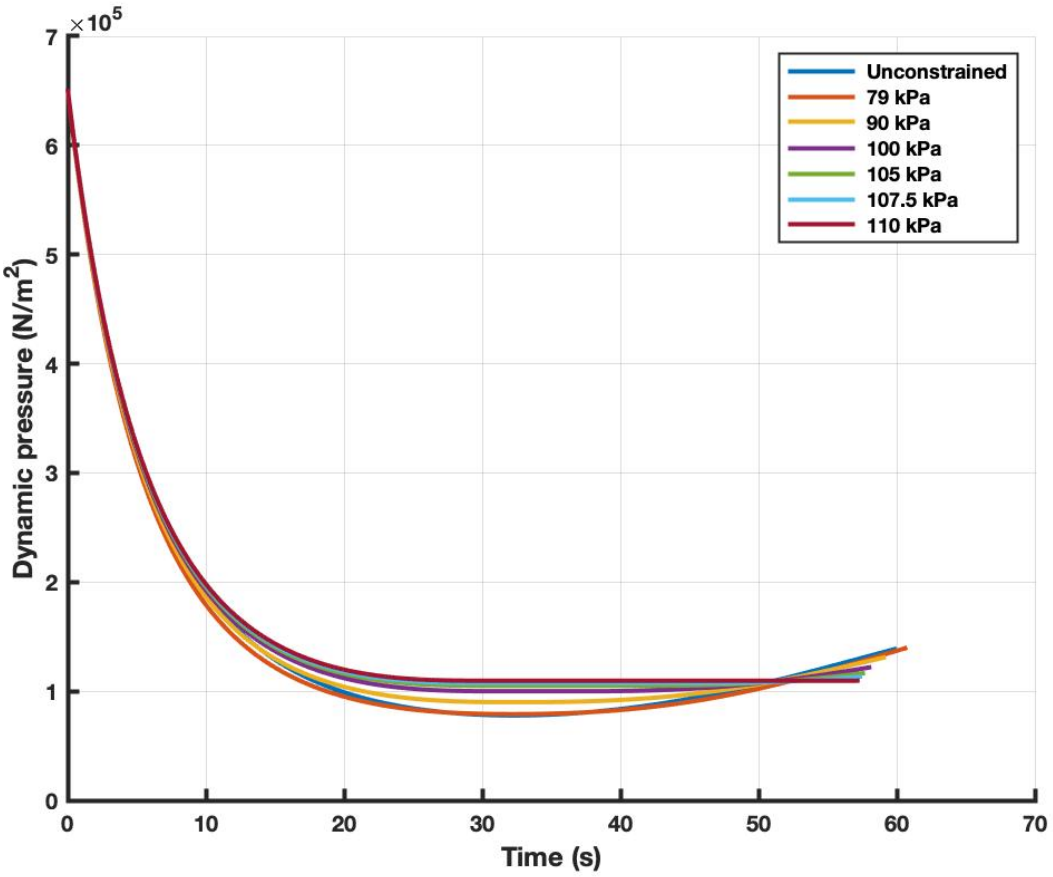

Figure 4.12: Dynamic Pressure Histories of Constrained Trajectories

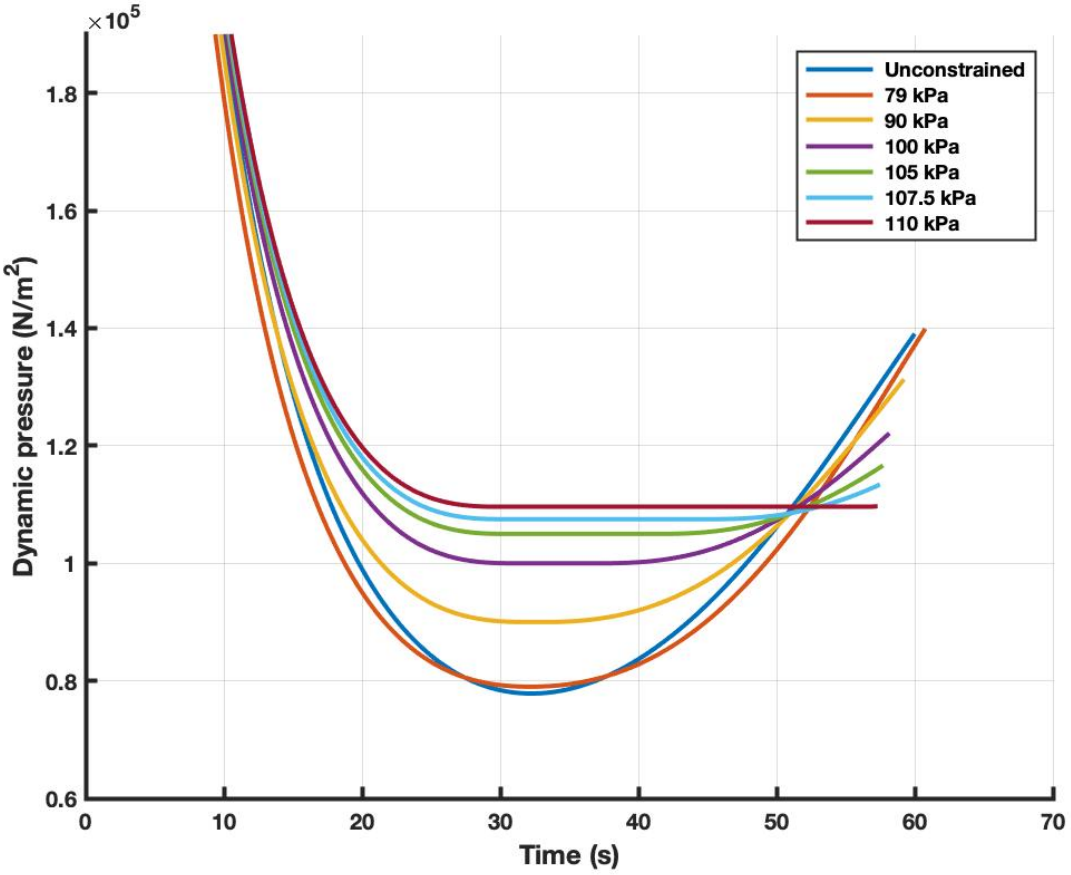

Figure 4.13: Zoomed-in View of Dynamic Pressure Histories of Constrained Trajectories

4.6 Applications to Other Constraints

It would be relatively straightforward to apply this methodology to implement other constraints, (e.g., aerodynamic, thermal, etc.) as long as they can be expressed as a function, C, which satisfies the following:

1. The value of the function on the constraint is equal to zero, and

2. The function is written such that the control, $\mathbf{u}$, shows up explicitly in one of the successive derivatives of C as per Equation (4.1). This also implies that the function C must be differentiable.

Once the derivative containing the control is found, and since this derivative is equal to zero, it can be used to derive an expression for the control on the constraint. This derivation can then be incorporated into the methodology previously discussed in this section.

Chapter 5

Discussion

5.1 Initial Work and Motivation

There is a considerable amount of research and development work underway on long-range guns (e.g., Navy electromagnetic railgun, Army Long-Range Cannon, etc.) and their corresponding ammunition. Given the extreme ranges that these guns can achieve, the projectiles they fire are almost always equipped with control surfaces to allow highly accurate results at impact. Consequently, significant investigative work has been conducted on the optimal control of these projectiles. However, there is no preferred approach for the development of an optimal control history and as such, this problem was selected to examine a novel scheme for developing costates estimates for an indirect-method approach. After examining several candidate methods of control, the angle of attack was selected as the single control for this problem. The formulations for the costate estimation machinery, the modeling of the projectile flight, and the multiple-shooting method application incorporating the nonlinear system equation solver were derived as outlined in Chapter 2.

5.2 Coding Development and Execution

The development of a computer program to execute this method occurred in phases. A subroutine was written to take the control history from the reference trajectory and integrate it to obtain the costate estimates for use in the multiple shooting method. An option was provided such that the estimates could be read as input from a file containing the final costate values from a converged trajectory. Another routine was written implementing the multiple-shooting method and the solver. All code, including a main program and all other supporting subroutines for the state and costate dynamics, calculation of α, Runga-Kutta, etc., were written in MATLAB[1] and executed on an Apple MacBook. After a significant amount of debugging, a converged solution was obtained; unfortunately, it took more than two days for the code to execute. It became apparent that although MATLAB excels at the manipulation of vectors and matrices, a different programming language would be necessary if any appreciable amount of work was to be accomplished. For a variety of reasons, FORTRAN was chosen and all of the routines[2] were translated. After a thorough checkout of the code, the same case was executed on the same machine in approximately 20 minutes. All subsequent runs were conducted using the FORTRAN code, although MATLAB was used for plotting purposes.

5.3 Comparison with the Direct Method

As discussed previously, an approximate-optimal trajectory from a direct-method application was used for a reference trajectory. Since direct-method applications are also candidates to obtain optimal control histories, it is logical to make a quick comparison of the two

[1] A previously existing subroutine containing an atmospheric model was used.

[2] Pre-existing FORTRAN subroutines for atmospheric conditions, and vector and matrix operations were incorporated.

methods for use in this application. While there are significant differences between the two implementations, which may appear to yield an "apples-to-oranges" comparison, there is still some useful information to be gleaned.

One of the main advantages of the indirect method over the direct method is that the former results in a truly optimal solution instead of an approximate one. The application which produced the reference trajectory discussed previously ran much more quickly than the indirect method application; this was primarily due to the fact that it was designed such that the control was obtained only at predefined nodes and not continuously throughout the projectile's flight as was the case with the indirect method application (this explains the the relative roughness of the control history in Figure 3.5). In addition, the direct control history is noticeably different than the optimal one. Depending on the particular problem being examined, this level of accuracy may be acceptable, especially if reduced run time is a priority.

5.4 Sensitivity of Costate Estimates

An unresolved aspect of this research is the quantification of the "roughness" of the costate estimates that will result in a converged optimal solution. One of the universally recognized drawbacks of the indirect method is the potential difficulty in obtaining the initial costate estimates. The costates can be highly unstable when integrating forward in time; this is the reason that the multiple-shooting method was employed in this research.

Several different approaches were observed to increase the probability of convergence. These include:

Better costate estimates. As discussed in Section 3.4, a set of costate estimates closer to the

final values would increase the likelihood of convergence to a solution. That method had the drawback of requiring additional run time to execute the backwards integration, but that time is negligible compared to the time required for the solver to execute.

Closer reference trajectory. One approach that proved to be successful, especially in the time and range extension research, was decreasing the difference between the reference trajectory and the desired final trajectory. For example, trying to extend the range from 40 km to 80 km in one step might prove to be too much; increasing from 40 km to 80 km in 5 km increments (or *walking* the trajectory up) would be much more likely to result in convergence.

Increased number of nodes. Increasing the number of nodes in the multiple-shooting method reduces the time between points where the $\boldsymbol{\lambda}$'s are reset, lowering the likelihood that they diverge unstably. The two drawbacks with this approach are the increased processing of the reference trajectory and, more significantly, the additional run time required to execute the solver.

The different approaches could also be in combination with each other if a problem proves to be more difficult to obtain a converged solution.

The last difficulty to be mentioned with this method is that there is no way to predict if the approach will work without actually executing the code. There is no way to determine ahead of time if the costate estimates are close enough such that a converged solution will be obtained. An investigation of the sensitivity of convergence to the uncertainty in the initial costate estimates would be a good area for follow-on research.

5.5 Real-time Guidance Solution

Another avenue for further research would be to investigate the combination of the in-plane optimal control solution methodology presented in this dissertation along with the heading error guidance scheme discussed in Appendix A. This guidance scheme, which is designed to minimize induced drag (and thereby preserve velocity) would be useful in handling out-of-plane errors that arise due to perturbations. These two applications, in combination with a closed-loop guidance scheme such as the ones described by Lawton, Pamadi, and Parks [47] or Lawton and Martell [48] would provide a real-time capability to implement the optimal control of a projectile.

Chapter 6

Conclusions

The serious scientific study of ballistics has been underway for approximately 200 years; the investigation of controlling projectiles for some intended purpose has been going on for almost as long. Very sophisticated gun-launched projectiles have been developed that can be aerodynamically controlled in flight through the use of movable control surfaces. Given that a projectile can be controlled, it follows that there would be a way to do so such that some optimal result could be obtained. A wide variety of optimal control techniques have been developed and many of them can be applied to the problem at hand.

This dissertation documents research that was focused on the development of a method to estimate initial values of the costates for an indirect method application of an optimal control problem. The method developed relies on a reference trajectory to obtain the estimates and then employs a unique variation of the multiple-shooting method. This uniqueness is due to the fact that only the costates were "reset" throughout the optimization process; the state variables (position and velocity) were stable enough through the integration process. The method was applied to the problem of obtaining an optimal control history for a gun-launched guided projectile for which the angle of attack, α, was the only control and the

final impact velocity was the parameter to be maximized. Initially, an approximately optimal trajectory from a direct-method software application was used as the reference trajectory. Subsequently, other trajectories were examined to determine if the resulting coarser costate estimates could still enable a converged solution. In many cases, a purely ballistic trajectory produced costate estimates that did result in convergence. Variations on range extensions and time extensions were examined where the final, converged costates from one trajectory were used as the initial estimates for a slightly longer (in range or time) trajectory. An alternate method to estimate the costates, which involved the backwards integration of the trajectory, was also discussed.

The method was expanded to include a minimum dynamic pressure, q_{min}, constraint in the same optimal control guidance application. Such a constraint would likely be imposed on an aerodynamically-controlled vehicle to prevent the loss of control in the rarefied portions of the atmosphere. The method allows the determination of the single control (the angle of attack) while on the constraint and uses that with a formulation for α during unconstrained flight to determine the optimal control solution which maximizes the impact velocity while still maintaining the constraint. The higher the value of q_{min}, the longer the angle of attack is manipulated to keep the projectile on the constraint. Eventually, however, a large enough minimum value can be selected which prevents the methodology from functioning correctly. This optimization capability has application to high-velocity projectiles and can be extended to be applied to aerodynamically-controlled missiles as well as hypersonic vehicles.

The implementation of other constraints (such as a thermal limit) was discussed. If the constraint can be expressed as a function such that it or its derivatives explicitly contains the control, then an expression for that control can be derived and used in this indirect-method application to obtain an optimal control solution.

A history of the development of the software, starting with the initial development in MAT-

LAB and the subsequent migration to FORTRAN was discussed. The initial development was facilitated by MATLAB, which is designed to allow the easy manipulation of vectors and matrices. However, the initial run to achieve convergence took several days to execute and the need for a different programming language was obvious. Subsequently, all of the code was migrated to FORTRAN and the identical converged solution was produced in approximately 20 minutes. While this resulted in an outstanding increase in throughput, it highlights one of the down sides of using the indirect method compared to the direct method. In the direct method, the control is directly (hence the name) manipulated and the resulting impact to the cost determined. The control history is adjusted until the lowest cost is obtained. Since fewer parameters are being manipulated, the direct method can often result in much lower execution times. However, this time advantage comes at a price: if the initial estimate of the control is too far away from the optimal, it could take much longer to converge or it could converge to a value that is not the absolute optimal result. Another major difference between the direct and indirect methods is that the direct method usually results in a near-optimal solution while the indirect method results in an exactly optimal solution.

Another positive quality of the indirect method is the relatively straight-forward ability to add constraints like the minimum dynamic pressure constraint discussed previously in this dissertation. While path constraints can be added to direct method implementations, indirect methods offer a significant advantage.

Another potentially negative aspect of the indirect method is the fact that the costates can be *extremely* sensitive to the initial values. In some cases, converged solutions were obtained from relatively coarse initial estimates; in other cases, reference trajectories that appeared to be relatively close to the desired trajectory did not provide costate estimates sufficiently close for convergence. This resulted in the inability to predict ahead of time if the method would converge or not. Investigation of the sensitivity of convergence to the

initial costate estimates is a topic of potential follow-on research, as it the extension of the capability to address other potential constraints. The continuing development of very high speed projectiles, along with an almost global resurgence in the research, development, and production of hypersonic vehicles will necessitate the ability to balance controllability and thermal heating in the application of optimal control solutions.

The author recognizes that although there are numerous optimal control techniques, there is no "one-size-fits-all" solution. As a result, this research represents another tool in the optimal control solver's toolkit. However, the ability to develop an optimal control solution with the imposition of a minimum dynamic pressure constraint is unique to this research.

Chapter 7

References

[1] Rife, J. P. and Carlisle, R. P., *The Sound of Freedom: Naval Weapons Technology at Dahlgren, Virginia 1918-2006*, 1st ed., U. S. Government Printing Office, Washington, D. C., 2007.

[2] Niemann, R. A., *Dahlgren's Participation in the Development of Computer Technology*, Naval Surface Warfare Center Miscellaneous Publication, NSWC MP 81-416, 1982.

[3] *National Geospatial-Intelligence Agency (NGA) Standardization Document; Department of Defense World Geodetic System 1984; Its Definition and Relationships with Local Geodetic Systems*, Version 1.0.0, 7-8-2014, Office of Geomatics, NGA.STND.0036_1.0.0_WGS84

[4] Phillips, C. P., "Guidance Algorithm for Range Maximization and Time-of-Flight Control of a Guided Projectile", *Journal of Guidance, Control, and Dynamics*, Vol. 31, No. 5, AIAA, September - October 2008, p. 1447.

[5] Phillips, C. P., "Trajectory Optimization for a Missile Using a Multi-tier Approach", *Journal of Spacecraft and Rockets*, Vol. 37, No. 5, AIAA, September - October 2000, p. 653.

[6] Lin, C.-F., Bibel, J., Ohlmeyer, E., and Malyevac, S., "Optimal Design of Integrated Missile Guidance and Control", *AIAA and SAE, 1998 World Aviation Conference*, 985519, Anaheim, California, 28-30 September 1998.

[7] Ohlmeyer, E. J., Pepitone, T. R., and Miller, B. L., "Assessment of Integrated GPS/INS for the EX-171 Extended Range Guided Munition, *AIAA Guidance, Navigation, and Control*

Conference and Exhibit, AIAA-2000-6702, Boston, Massachusetts, 10-12 August 1998.

[8] Ohlmeyer, E., Bibel, J., Phillips, C., and Reiman, S., "Guidance and Navigation System Design for a Ship Self Defense Missile", *AIAA Guidance, Navigation, and Control Conference and Exhibit*, AIAA-2000-6702, Keystone, Colorado, 21-24 August 2006.

[9] Ohlmeyer, E., Pepitone, T., Miller B., Malyevac, D., Bibel, J., and Evans, A., "GPS-aided Navigation System Requirements for Smart Munitions and Guided Missiles", *AIAA Guidance, Navigation, and Control Conference*, AIAA-97-3683, New Orleans, Louisiana, 11-13 August 1997.

[10] Pamadi, K. B. and Ohlmeyer, E. J., "Evaluation of Two Guidance Laws for Controlling the Impact Flight Path Angle of a Naval Gun Launched Spinning Projectile", *AIAA Guidance, Navigation, and Control Conference and Exhibit*, AIAA-2000-6081, Keystone, Colorado, 21-24 August 2006.

[11] Kirk, D. E., *Optimal Control Theory: An Introduction*, 1st ed., Dover, New York, 2004, Chap. 1 "Introduction", p. 3.

[12] Leitmann, G., *Optimization Techniques*, 1st ed., Academic Press, New York, 1962, Chap. 7 "Pontryagin Maximum Principle", pp. 255-279.

[13] Betts, J. T., *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, 2nd ed., Society for Industrial and Applied Mathematics, Washington, D.C., 2010, Chap. 3 "Optimal Control Preliminaries", pp. 109-111.

[14] Hargraves, C. R. and Paris, S. W., "Direct Trajectory Optimization Using Nonlinear Programming and Collocation", *Journal of Guidance, Control, and Dynamics*, Vol. 10, No. 4, AIAA, July - August 1987, pp. 338-342.

[15] Dennis, M. E., Hager, W. W., and Rao, A. V., "Computational Method for Optimal Guidance and Control Using Adaptive Gaussian Quadrature Collocation", *Journal of Guidance, Control, and Dynamics*, Vol. 42, No. 9, AIAA, September 2019, pp. 2026-2041.

[16] Benson, D. A., Huntingdon, G. T., Thorvaldsen, T. P., and Rao, A. V., "Direct Trajectory Optimization and Costate Estimation via an Orthogonal Collocation Method", *Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 6, AIAA, November - December 2006, pp. 1435-1440.

[17] Betts, J. T., *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, 2nd ed., Society for Industrial and Applied Mathematics, Washington, D.C., 2010, Chap. 4 "The Optimal Control Problem", p. 129.

[18] Bryson, A. E. and Ho, Y.-C., *Applied Optimal Control Theory, Optimization, Estimation, and Control*, 2nd ed., Taylor and Francis, New York, 1975, Chap. 7 "Numerical Solution of Optimal Programming and Control Problems", p. 214.

[19] Saghamanesh, M. and Baoyin, H., "Optimal Gravity-Assist Low-Thrust Trajectory Using a Robust Homotopic Approach", *2018 IEEE CSAA Guidance, Navigation and Control Conference*, Xiamen, China, 2018, pp. 1-8, DOI: 10.1109/GNCC42960.2018.9018896.

[20] Bushong, P. M. and Lawton, J. A., "Fast Solution to an Optimal Trajectory Problem", *Conference Proceedings of the AAS/AIAA Spaceflight Mechanics Meeting held 14-16 February 1994, Cocoa Beach, Florida*, Advances in the Astronautical Sciences, Vol. 87, AAS 94-131, Univelt, San Diego, 1994, pp. 973-980.

[21] Jiang, F., Tang, G., and Li, J., "Improving Low-Thrust Trajectory Optimization by Adjoint Shape-Based Path", *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 12, AIAA, December 2017, pp. 3280-3287.

[22] Seywald, H. and Kumar, R. R., "Method for Automatic Costate Calculation", *Journal of Guidance, Control, and Dynamics*, Vol. 19, No. 6, AIAA, January 1999, pp. 490-492.

[23] Lee, D. and Bang, H., "Efficient Initial Costates Estimation for Optimal Spiral Orbit Transfer Trajectories Design", *Journal of Guidance, Control, and Dynamics*, Vol. 32, No. 6, AIAA, November - December 2009, pp. 1943-1947.

[24] Yan, H. and Wu, H., "Initial Adjoint-Variable Guess Technique and Its Application in Optimal Orbit Transfer", *Journal of Guidance*, Vol. 22, No. 3, AIAA, November - December 1995, pp. 1267-1272.

[25] Miswanto, Pranoto, I, Muhammad, H., and Mahayana, D., "The Application of the Steepest Gradient Descent for Control Design of Dubins Car for Tracking a Desired Path", *Journal of Math and Its Applications*, Vol. 4, No. 1, May 2007, pp. 1-8.

[26] Farhoo, F. and Ross, I. M., "Costate Estimation by a Legendre Pseudospectral Method", *AIAA Journal of Guidance, Control and Dynamics*, Vol. 24, No. 2, March - April 2001, pp. 270-277.

[27] Gong, Q., Ross, I. M., and Farhoo, F., "Costate Computation by a Chebyshev Pseudospectral Method", *AIAA Journal of Guidance, Control and Dynamics*, Vol. 33, No. 2, March - April 2010, pp. 623-628.

[28] Gath, P. F. and Well, K. H., "Trajectory Optimization Using a Combination of Direct

Multiple Shooting and Collocation", *AIAA Guidance, Navigation, and Control Conference and Exhibit*, AIAA-2001-4047, Montreal, Canada, 6-9 August 2001.

[29] Bulirsch, R., Nerz, E., and von Stryk, O., "Combining Direct and Indirect Methods in Optimal Control: Range Maximization of a Hang Glider", *International Series of Numerical Mathematics*, Vol. 111, 2008, pp. 273-288.

[30] Martell, C. A. and Lawton, J. A., "Adjoint Variable Solutions via an Auxiliary Optimization Problem", *Journal of Guidance, Control, and Dynamics*, Vol. 18, No. 6, AIAA, November - December 1995, pp. 1267-1272.

[31] Seywald, H. and Cliff, E., "Goddard Problem in the Presence of a Dynamic Pressure Limit", *Journal of Guidance, Control and Dynamics*, Vol. 16, No. 4, July - August 1993, pp. 776-781.

[32] Graichen, K. and Petit, N., "Solving the Goddard Problem with Thrust and Dynamic Pressure Constraints Using Saturation Functions", *Proceedings of the 17th World Congress, The International Federation of Automatic Control*, Seoul, Korea, July 2008, pp. 14301-14306.

[33] Graichen, K., Kugi, A., Petit, N., and Chaplais, F., "Handling Constraints in Optimal Control with Saturation Functions and System Extension", *Systems and Control Letters*, Vol. 59, 2010, pp. 671-679.

[34] Park, S-Y., "Launch Vehicle Trajectories with a Dynamic Pressure Constraint", *AIAA Journal of Spacecraft and Rockets*, Vol. 35, No. 6, November - December 1998, pp. 765-773.

[35] Bryson, A. E. and Ho, Y.-C., *Applied Optimal Control Theory, Optimization, Estimation, and Control*, 2nd ed., Taylor and Francis, New York, 1975., Chap. 2 "Optimization Problems for Dynamic Systems", pp. 55-89.

[36] Kirk, D. E., *Optimal Control Theory: An Introduction*, 1st ed., Dover, New York, 2004., Chap. 5 "The Variational Approach to Optimal Control Problems" , pp. 184-201

[37] Leitmann, G., *Calculus of Variations and Optimal Control*, 1st ed., Academic Press, New York, 1962, Chap. 2 "Problem Statement and Necessary Conditions for an Extremum", p. 7.

[38] Lee, E. B. and Markus, L., *Foundations of Optimal Control Theory*, 2nd ed., John Wiley and Sons, Inc., New York, 1967, Chap. 4 "Necessary and Sufficient Conditions for Optimal Control", p. 308.

[39] Ewing, G. M., *Calculus of Variations with Applications*, 2nd ed., W. W. Norton, New

York, 1975, Chap. 4 "The Non-parametric problem of Bolza", pp. 108-109.

[40] Morrison, D., Riley, J., and Zancanaro, J., *Multiple Shooting Method for Two-Point Boundary Value Problems*, Communications of the ACM, Association for Computing Machinery, 1962, 5 (12), pp.613-614. 10.1145/355580.369128 . hal-01634264

[41] Fox, L., The Numerical Solution of Two-point Boundary Problems in Ordinary Differential Equations. Oxford, 1957.

[42] Morris, A. H., *NSWC Library of Mathematics Subroutines*, NSWC TR 90-21, 1990.

[43] Bryson, A. E. and Ho, Y.-C., *Applied Optimal Control Theory, Optimization, Estimation, and Control*, 2nd ed., Taylor and Francis, New York, 1975, Chap. 3 "Optimization Problems for Dynamic Systems with Path Constraints", pp. 117-119.

[44] Bryson, A. E. and Ho, Y.-C., *Applied Optimal Control Theory, Optimization, Estimation, and Control*, 2nd ed., Taylor and Francis, New York, 1975, Chap. 4 "Optimization Problems for Dynamic Systems with Path Constraints", pp. 101-103.

[45] Seywald, H. and Cliff, E. M., *Short Communications on the Existence of Touch Points for First-Order State Inequality Constraints*, Optimal Control Applications and Methods, Vol. 17, pp. 357-366, 1996.

[46] Ross, I. M., Gong, Q., Karpenko, M., Proulx, R., J., "Scaling and Balancing for High-Performance Computation of Optimal Controls", *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 10, AIAA, 2018, pp. 2086-2097.

[47] Lawton, J. A., Pamadi, K. B., and Parks, D. J., "Optimal-Control-Based Trajectory Design for Gun-Launched Projectile Midcourse Guidance", *Proceedings of the 8th Annual Hypervelocity Gun Weapon System Workshop*, 19-20 September 2017

[48] Lawton, J. A., and Martell, C. A., "Hybrid Neighboring-Optimal-Control and Lambert-Based Interceptor Boost-Phase Guidance", 1997 AIAA/BMDO Technology Readiness Conference, paper no. 16-08.

Appendices

Appendix A

Minimum Induced-Drag Heading Error Guidance

This formulation utilizes a single control, $\dot{\gamma}$, where γ, the heading error angle, is defined positive clockwise from the x-axis to the velocity vector as shown in Figure A.1. Define the dynamics as follows:

$$\dot{x} = -V\cos\gamma \tag{A.1}$$

$$\dot{y} = V\sin\gamma \tag{A.2}$$

$$\dot{\gamma} = u \tag{A.3}$$

$$\dot{V} = 0 \tag{A.4}$$

where V is the initial speed (which remains constant). The initial state is given by:

$$\psi_0 = \begin{Bmatrix} x_0 \\ 0 \\ \gamma_0 \\ V_0 \end{Bmatrix} \tag{A.5}$$

and the final state is given by:

$$\psi_f = \begin{Bmatrix} 0 \\ 0 \\ \gamma_f \\ V_0 \end{Bmatrix} \tag{A.6}$$

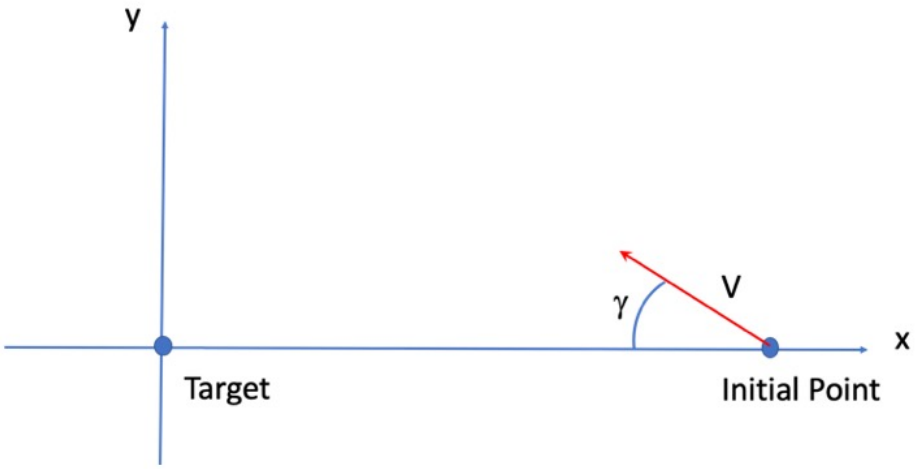

Figure A.1: Coordinate System Showing Definition of Heading Error Angle, γ

We define the cost function as:

$$J = \frac{1}{2} \int_{t_0}^{t_f} u^2 \, dt \tag{A.7}$$

and construct the Hamiltonian to yield:

$$\mathcal{H} = -\lambda_x V \cos\gamma + \lambda_y V \sin\gamma + \lambda_\gamma u + \lambda_V \cdot 0 - \frac{1}{2}u^2 \tag{A.8}$$

Taking the partial derivative of $\mathcal{H}$ with respect to the control and setting equal to zero gives:

$$\frac{\partial \mathcal{H}}{\partial u} = \lambda_\gamma - u = 0 \tag{A.9}$$

and hence:

$$\lambda_\gamma = u \tag{A.10}$$

which implies that $\dot{u} = \dot{\lambda_\gamma}$.

Now determine the time derivatives of the adjoint variables by taking the appropriate partial derivatives of the Hamiltonian:

$$\dot{\lambda_x} = -\frac{\partial \mathcal{H}}{\partial x} = 0 \tag{A.11}$$

$$\dot{\lambda_y} = -\frac{\partial \mathcal{H}}{\partial y} = 0 \tag{A.12}$$

$$\dot{\lambda_\gamma} = -\frac{\partial \mathcal{H}}{\partial \gamma} = -\left[\lambda_x V \sin\gamma + \lambda_y V \cos\gamma\right] \tag{A.13}$$

$$\dot{\lambda_V} = -\frac{\partial \mathcal{H}}{\partial V} = \left[\lambda_x \cos\gamma - \lambda_y \sin\gamma\right] \tag{A.14}$$

Please notice that (A.11) and (A.12) show that λ_x and λ_y are constant throughout the flight.

At this point it convenient to express (A.13) as a γ-based derivative as opposed to a time-based one. We do this by using the following identity:

$$\frac{d()}{d\gamma} = \frac{1}{\dot{\gamma}}\frac{d()}{dt} = \frac{1}{u}\frac{d()}{dt} \tag{A.15}$$

Therefore,

$$\frac{du}{d\gamma} = \frac{1}{u}\dot{\lambda_\gamma} \tag{A.16}$$

Inserting (A.13) into (A.16) and rearranging terms yields:

$$u \cdot du = -\big[\lambda_x V \sin\gamma + \lambda_y V \cos\gamma\big] d\gamma \tag{A.17}$$

Integrating both sides:

$$\frac{u^2}{2} = \big[\lambda_x V \cos\gamma - \lambda_y V \sin\gamma + C\big] \tag{A.18}$$

and solving for u gives:

$$u = \pm\sqrt{2\lambda_x V \cos\gamma - 2\lambda_y V \sin\gamma + C} \tag{A.19}$$

Using $\mathcal{H} = 0$ and (A.19) to solve for C we find:

$$\mathcal{H} = -\lambda_x V \cos\gamma + \lambda_y V \sin\gamma + \lambda_\gamma u - \frac{1}{2}u^2 \tag{A.20}$$

$$= -\lambda_x V \cos\gamma + \lambda_y V \sin\gamma + u \cdot u - \frac{1}{2}u^2 \tag{A.21}$$

$$= -\lambda_x V \cos\gamma + \lambda_y V \sin\gamma + \frac{1}{2}u^2 \tag{A.22}$$

$$= -\lambda_x V \cos\gamma + \lambda_y V \sin\gamma + \lambda_x V \cos\gamma - \lambda_y V \sin\gamma + C \tag{A.23}$$

which indicates that C is identically equal to 0. Therefore, the expression for the control becomes:

$$u = \pm\sqrt{2\lambda_x V \cos\gamma - 2\lambda_y V \sin\gamma} \tag{A.24}$$

It will be convenient for us to express u in the form:

$$u = \pm b\sqrt{\sin\left(\gamma - \gamma_f\right)} \tag{A.25}$$

where γ_f is the final γ. Squaring both sides of (A.25),

$$u^2 = b^2 \sin\left(\gamma - \gamma_f\right) \tag{A.26}$$

and using the appropriate half angle formula, yields:

$$u^2 = b^2\big(\sin\gamma\cos\gamma_f - \cos\gamma\sin\gamma_f\big) \tag{A.27}$$

squaring (A.24) and equating to (A.27) gives:

$$\lambda_x = -\frac{b^2}{2V}\sin\gamma_f \tag{A.28}$$

$$\lambda_y = -\frac{b^2}{2V}\cos\gamma_f \tag{A.29}$$

Squaring and adding (A.28) and (A.29) allows the following determination of b:

$$\lambda_x{}^2 + \lambda_y{}^2 = \frac{b^4}{4V^2}\sin^2\gamma_f + \frac{b^4}{4V^2}\cos^2\gamma_f \tag{A.30}$$

$$= \frac{b^4}{4V^2} \tag{A.31}$$

Rearranging yields:

$$b^2 = 2V\sqrt{\lambda_x{}^2 + \lambda_y{}^2} \tag{A.32}$$

while dividing (A.28) by (A.29) gives:

$$\tan\gamma_f = \frac{-\lambda_x}{-\lambda_y} \tag{A.33}$$

where the negative signs are left in place to indicate the proper use of the four-quadrant arctangent in determining γ_f correctly.

Recall that V, λ_x, and λ_y are all constants. Equations (A.32) and (A.33) show that γ_f and b are also constant.

Returning to the flight dynamics, we can use (A.1) to find an expression for b. Begin by again changing the time derivative $\dot{x}$ to one in terms of γ.

$$\frac{dx}{d\gamma} = \frac{dt}{d\gamma}\frac{dx}{dt} \tag{A.34}$$

$$= \frac{1}{\dot{\gamma}}(-V\cos\gamma) \tag{A.35}$$

$$= \frac{-V}{u} \cos\gamma \tag{A.36}$$

Rearranging and utilizing (A.25):

$$dx = \frac{-V \cos\gamma d\gamma}{b\sqrt{\sin\left(\gamma - \gamma_f\right)}} \tag{A.37}$$

In order to integrate, let us utilize the following substitution:

$$U = \sin\left(\gamma - \gamma_f\right) \tag{A.38}$$

$$dU = \cos\left(\gamma - \gamma_f\right) d\gamma \tag{A.39}$$

which can be manipulated into:

$$dU = \left[\cos\gamma\cos\gamma_f + \sin\gamma\sin\gamma_f\right] d\gamma \tag{A.40}$$

Rearranging yields:

$$\cos\gamma d\gamma = \frac{dU - \sin\gamma\sin\gamma_f d\gamma}{\cos\gamma_f} \tag{A.41}$$

Using (A.38) and (A.41) in (A.37) gives:

$$dx = \frac{-V}{bU^{\frac{1}{2}}} \frac{\left[dU - \sin\gamma\sin\gamma_f d\gamma\right]}{\cos\gamma_f} \tag{A.42}$$

$$= \frac{-V\,dU}{b\cos\gamma_f U^{\frac{1}{2}}} + \frac{V}{b}\tan\gamma_f \frac{\sin\gamma d\gamma}{\sqrt{\sin\left(\gamma-\gamma_f\right)}} \tag{A.43}$$

Integration of the first term is straightforward. To integrate the second term, let us express (A.2) as a derivative in γ:

$$\frac{dy}{d\gamma} = \frac{1}{\dot{\gamma}}\dot{y} = \frac{V}{u}\sin\gamma \tag{A.44}$$

or

$$dy = \frac{V}{u}\sin\gamma d\gamma \tag{A.45}$$

Using (A.25) gives:

$$dy = \frac{V}{b}\frac{\sin\gamma d\gamma}{\sqrt{\sin\left(\gamma-\gamma_f\right)}} \tag{A.46}$$

The integral of the left-hand side yields $y_f - y_0$ which in our case is equal to 0. Therefore, the integral of the second term of (A.43) is also equal to 0. Continuing from (A.43) and integrating, we find:

$$x_f - x_0 = \frac{-2V}{b\cos\gamma_f}\left[{U_f}^{\frac{1}{2}} - {U_0}^{\frac{1}{2}}\right] \tag{A.47}$$

or, since x_f is zero and $\sin\left(\gamma_f-\gamma_f\right)$ is zero (which means ${U_f}^{\frac{1}{2}}$ is also zero), then

$$-x_0 = \frac{-2V}{b\cos\gamma_f}\left[-{U_0}^{\frac{1}{2}}\right] \tag{A.48}$$

$$= \frac{2V}{b} \frac{\sqrt{\sin\left(\gamma_0 - \gamma_f\right)}}{\cos\gamma_f} \tag{A.49}$$

Rearranging and solving for b yields:

$$b = \frac{-2V}{x_0} \frac{\sqrt{\sin\left(\gamma_0 - \gamma_f\right)}}{\cos\gamma_f} \tag{A.50}$$

Observation of (A.50) reveals that b is a constant since it is only dependent on other constant terms.

Let us now determine an expression for x in terms of γ. Returning to (A.37) and using the substitution $g = \gamma - \gamma_f$; $dg = d\gamma$ yields

$$dx = \frac{-V\cos(g + \gamma_f)}{b\sqrt{\sin g}} dg \tag{A.51}$$

Using the appropriate trig identity gives:

$$dx = \frac{-V}{b}\left[\frac{\cos g \cos\gamma_f dg}{\sqrt{\sin g}} - \sin\gamma_f \sqrt{\sin g} dg\right] \tag{A.52}$$

Making the substitution $u = \sin g$, and $du = \cos g dg$ and integrating both sides yields:

$$x_f - x_0 = \frac{-V}{b}\left[\cos\gamma_f \int_{u_0}^{u_f} \frac{du}{u^{\frac{1}{2}}} - \sin\gamma_f \int_{g_0}^{g_f} \sqrt{\sin g} dg\right] \tag{A.53}$$

Since x_f is known and the first integral on the right is easily completed, (A.53) becomes (ignoring the second integral for the moment...):

$$-x_0 = \frac{-V}{b}\Big[2\cos\gamma_f({u_f}^{\frac{1}{2}} - {u_0}^{\frac{1}{2}}) - \sin\gamma_f \int_{g_0}^{g_f} \sqrt{\sin g}dg\Big] \tag{A.54}$$

Resubstituting for u gives:

$$-x_0 = \frac{-V}{b}\Big[2\cos\gamma_f(\sqrt{\sin g_f} - \sqrt{\sin g_0}) - \sin\gamma_f \int_{g_0}^{g_f} \sqrt{\sin g}dg\Big] \tag{A.55}$$

Resubstituting for g and rearranging now results in:

$$\frac{bx_0}{V} = -\Big[2\cos\gamma_f\sqrt{\sin(\gamma_0 - \gamma_f)} - sin\gamma_f \int_{g_0}^{g_f} \sqrt{\sin g}dg\Big] \tag{A.56}$$

The integral of $\sqrt{\sin(g)}$ presents a problem and will be dealt with shortly.

Returning to (A.46) and again using the substitution $g = \gamma - \gamma_f$; $dg = d\gamma$ we have

$$dy = \frac{V}{b}\frac{\sin(g + \gamma_f)d\gamma}{\sqrt{\sin g}} \tag{A.57}$$

which again, with the appropriate trig identity yields:

$$dy = \frac{V}{b}\Big[\sqrt{\sin g}\cos\gamma_f + \frac{\cos g}{\sqrt{\sin g}}\sin_{\gamma_f}\Big]dg \tag{A.58}$$

Integrating both sides yields:

$$y_f - y_0 = \frac{V}{b}\Big[\cos\gamma_f \int_{g_0}^{g_f} \sqrt{\sin g}dg + \sin\gamma_f \int_{g_0}^{g_f} \frac{\cos g}{\sqrt{\sin g}}dg\Big] \tag{A.59}$$

Since $y_f = y_0 = 0$, we can rearrange to obtain:

$$\cos\gamma_f \int_{g_0}^{g_f} \sqrt{\sin g}dg = -\sin\gamma_f \int_{g_0}^{g_f} \frac{\cos g}{\sqrt{\sin g}}dg \tag{A.60}$$

Using the result from earlier, and dividing by $\cos\gamma_f$ yields:

$$\int_{g_0}^{g_f} \sqrt{\sin g}dg = 2\tan\gamma_f\sqrt{\sin\left(\gamma_0 - \gamma_f\right)} \tag{A.61}$$

And now for the integral of $\sqrt{\sin g}$...

Let us begin with the infinite series representation of $\sin g$:

$$\sin g = \sum_{k=0}^{\infty} \frac{(-1)^k g^{2k+1}}{(2k+1)!} = g - \frac{g^3}{3!} + \frac{g^5}{5!} - \frac{g^7}{7!}... \tag{A.62}$$

It follows that:

$$\sqrt{\sin g} = \sqrt{g - \frac{g^3}{3!} + \frac{g^5}{5!} - \frac{g^7}{7!}...} = \sqrt{g}\sqrt{1 - \frac{g^2}{3!} + \frac{g^4}{5!} - \frac{g^6}{7!}...} \tag{A.63}$$

It is also helpful to utilize the following infinite series representation:

$$(1+x)^{\frac{1}{2}} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{x^3}{16} - \frac{5x^4}{128}... \tag{A.64}$$

Examining (A.63) and letting:

$$x = -\frac{g^2}{3!} + \frac{g^4}{5!} - \frac{g^6}{7!}... \tag{A.65}$$

Allows us to write:

$$\sqrt{\sin g} = \sqrt{g}\Big[1+\frac{1}{2}\Big[-\frac{g^2}{3!}+\frac{g^4}{5!}-\frac{g^6}{7!}...\Big]-\frac{1}{8}\Big[-\frac{g^2}{3!}+\frac{g^4}{5!}-\frac{g^6}{7!}...\Big]^2+\frac{1}{16}\Big[-\frac{g^2}{3!}+\frac{g^4}{5!}-\frac{g^6}{7!}...\Big]^3\Big] \quad \text{(A.66)}$$

$$\approx \sqrt{g}\Big[1+\frac{1}{2}\Big[-\frac{g^2}{3!}+\frac{g^4}{5!}\Big]-\frac{1}{8}\Big[-\frac{g^2}{3!}+\frac{g^4}{5!}\Big]^2+\frac{1}{16}\Big[-\frac{g^2}{3!}+\frac{g^4}{5!}\Big]^3\Big] \quad \text{(A.67)}$$

$$= g^{\frac{1}{2}}+\frac{1}{2}\Big[-\frac{g^{\frac{5}{2}}}{3!}+\frac{g^{\frac{9}{2}}}{5!}\Big]-\frac{g^{\frac{1}{2}}}{8}\Big[\frac{g^4}{(3!)^2}-\frac{2g^6}{3!5!}+\frac{g^8}{(5!)^2}\Big] \quad \text{(A.68)}$$

$$= g^{\frac{1}{2}}+\frac{1}{2}\Big[-\frac{g^{\frac{5}{2}}}{3!}+\frac{g^{\frac{9}{2}}}{5!}\Big]-\frac{g^{\frac{1}{2}}}{8}\Big[\frac{g^4}{(3!)^2}-\frac{2g^6}{3!5!}+\frac{g^8}{(5!)^2}\Big] \quad \text{(A.69)}$$

$$= g^{\frac{1}{2}}+\frac{1}{2}\Big[-\frac{g^{\frac{5}{2}}}{3!}+\frac{g^{\frac{9}{2}}}{5!}\Big]-\frac{1}{8}\Big[\frac{g^{\frac{9}{2}}}{3!\cdot 3!}-\frac{2g^{\frac{13}{2}}}{3!\cdot 5!}+\frac{g^{\frac{17}{2}}}{5!\cdot 5!}\Big] \quad \text{(A.70)}$$

We will now use this result to compute a value for γ_f. We will use the Newton-Raphson method beginning with (A.61):

$$F(\gamma_f) = -\int_{g_0}^{g_f}\sqrt{\sin g}dg + 2\tan\gamma_f\sqrt{\sin(\gamma_0-\gamma_f)} \quad \text{(A.71)}$$

$$\approx -\int_{g_0}^{g_f}\Big[g^{\frac{1}{2}}+\frac{1}{2}\Big(\frac{-g^{\frac{5}{2}}}{3!}+\frac{g^{\frac{9}{2}}}{5!}\Big)-\frac{1}{8}\Big(\frac{g^{\frac{9}{2}}}{3!\cdot 3!}\Big)\Big]dg + 2\tan\gamma_f\sqrt{\sin(\gamma_0-\gamma_f)} \quad \text{(A.72)}$$

$$= \Big[-\frac{2g^{\frac{3}{2}}}{3} + \frac{g^{\frac{7}{2}}}{7 \cdot 3!} - \frac{g^{\frac{11}{2}}}{11 \cdot 5!} + \frac{1}{4}\frac{g^{\frac{11}{2}}}{11 \cdot 3! \cdot 3!}\Big]_0^{g_0} + 2\tan\gamma_f\sqrt{\sin\left(\gamma_0 - \gamma_f\right)} \tag{A.73}$$

$$= -\frac{2g_0{}^{\frac{3}{2}}}{3} + \frac{g_0{}^{\frac{7}{2}}}{7 \cdot 3!} - \frac{g_0{}^{\frac{11}{2}}}{11 \cdot 5!} + \frac{1}{4}\frac{g_0{}^{\frac{11}{2}}}{11 \cdot 3! \cdot 3!} + 2\tan\gamma_f\sqrt{\sin\left(\gamma_0 - \gamma_f\right)} \tag{A.74}$$

$$= -\frac{2}{3}(\gamma_0 - \gamma_f)^{\frac{3}{2}} + \frac{(\gamma_0 - \gamma_f)^{\frac{7}{2}}}{7 \cdot 3!} - \frac{(\gamma_0 - \gamma_f)^{\frac{11}{2}}}{11 \cdot 5!} + \frac{(\gamma_0 - \gamma_f)^{\frac{11}{2}}}{44 \cdot 3! \cdot 3!} + 2\tan\gamma_f\sqrt{\sin\left(\gamma_0 - \gamma_f\right)} \tag{A.75}$$

Taking the derivative yields:

$$\begin{aligned} F'(\gamma_f) = (\gamma_0 - \gamma_f)^{\frac{1}{2}} - \frac{(\gamma_0 - \gamma_f)^{\frac{5}{2}}}{2 \cdot 3!} + \frac{(\gamma_0 - \gamma_f)^{\frac{9}{2}}}{2 \cdot 5!} - \frac{(\gamma_0 - \gamma_f)^{\frac{11}{2}}}{8 \cdot 3! \cdot 3!} \\ + 2\Big[-\frac{1}{2}\tan\gamma_f\frac{\cos\left(\gamma_0 - \gamma_f\right)}{\sqrt{\sin\left(\gamma_0 - \gamma_f\right)}} + \sqrt{\sin\left(\gamma_0 - \gamma_f\right)}\sec^2\gamma_f\Big] \end{aligned} \tag{A.76}$$

The value of γ_f can then be iterated by making an initial guess for γ_f and using the formula:

$$\gamma_{f_{n+1}} = \gamma_{f_n} - \frac{F(\gamma_{f_n})}{F'(\gamma_{f_n})} \tag{A.77}$$

Note that only γ_0 and the initial guess for γ_f are needed to compute a final value of γ_f.

It is an interesting exercise to find the solution to the 1st-term approximation for $F(\gamma_f)$:

$$F(\gamma_f) \approx -\frac{2}{3}(\gamma_0 - \gamma_f)^{\frac{3}{2}} - 2\tan\gamma_f\sin^{\frac{1}{2}}(\gamma_0 - \gamma_f) = 0 \tag{A.78}$$

using a small-angle approximation (i.e., $\sin\alpha \approx \alpha, \tan\alpha \approx \alpha$), (A.78) becomes:

$$\approx -(\gamma_0 - \gamma_f)^{\frac{3}{2}} - 3\gamma_f(\gamma_0 - \gamma_f)^{\frac{1}{2}} = 0 \tag{A.79}$$

or:

$$(\gamma_0 - \gamma_f)^{\frac{3}{2}} = 3\gamma_f(\gamma_0 - \gamma_f)^{\frac{1}{2}} \tag{A.80}$$

and then squaring both sides yields:

$$(\gamma_0 - \gamma_f)^3 = 9{\gamma_f}^2(\gamma_0 - \gamma_f) \tag{A.81}$$

or:

$$(\gamma_0 - \gamma_f)^2 = 9{\gamma_f}^2 \tag{A.82}$$

expanding the term on the left:

$${\gamma_f}^2 - 2\gamma_0\gamma_f + {\gamma_0}^2 = 9{\gamma_f}^2 \tag{A.83}$$

or:

$$8{\gamma_f}^2 + 2\gamma_0\gamma_f - {\gamma_0}^2 = 0 \tag{A.84}$$

Utilizing the quadratic formula yields:

$$\gamma_f = \frac{-2\gamma_0 \pm \sqrt{4{\gamma^2}_0 + 32{\gamma^2}_0}}{16} \tag{A.85}$$

$$= \frac{-2\gamma_0 \pm \sqrt{36{\gamma^2}_0}}{16} \tag{A.86}$$

$$= \frac{-2\gamma_0 \pm 6\gamma_0}{16} \tag{A.87}$$

which yields two solutions:

$$\gamma_f = \frac{\gamma_0}{4} \tag{A.88}$$

which is not viable (γ_f must be negative given our problem setup), and:

$$\gamma_f = -\frac{\gamma_0}{2} \tag{A.89}$$

Returning to (A.54), and armed with our numerical approximation for $\int \sin^{\frac{1}{2}} g$ we can now find an expression for any x at any value of γ:

$$x - x_0 = \frac{-V}{b}\Big[\cos\gamma_f \int_{u_0}^{u} \frac{du}{u^{\frac{1}{2}}} - \sin\gamma_f \int_{g_0}^{g} \sqrt{\sin g}dg\Big] \tag{A.90}$$

or:

$$x = x_0 + \frac{-V}{b}\left[\Big[2\cos\gamma_f u^{\frac{1}{2}}\Big]_{u_0}^{u} - \sin\gamma_f \int_{g_0}^{g} \sqrt{\sin g}dg\right] \tag{A.91}$$

$$= x_0 + \frac{-V}{b}\left[\left[2\cos\gamma_f \sin^{\frac{1}{2}} g\right]_{g_0}^{g} - \sin\gamma_f \int_{g_0}^{g} \sqrt{\sin g}\, dg\right] \tag{A.92}$$

$$= x_0 + \frac{-V}{b}\left[2\cos\gamma_f\Big(\sin^{\frac{1}{2}}(\gamma-\gamma_f) - \sin^{\frac{1}{2}}(\gamma_0-\gamma_f)\Big) - \sin\gamma_f \int_{g_0}^{g} \sqrt{\sin g}\, dg\right] \tag{A.93}$$

$$= x_0 + \frac{-V}{b}\left[2\cos\gamma_f\Big(\sin^{\frac{1}{2}}(\gamma-\gamma_f) - \sin^{\frac{1}{2}}(\gamma_0-\gamma_f)\Big) - \sin\gamma_f \int_{g_0}^{g} \sqrt{\sin g}\, dg\right] \tag{A.94}$$

Letting $\Delta x = -\sin\gamma_f \int_{g_0}^{g} \sqrt{\sin g}\, dg$ and using the first 4 terms from (A.75) we obtain:

$$\begin{aligned}\Delta x = &-\sin\gamma_f\left[\frac{2}{3}(\gamma-\gamma_f)^{\frac{3}{2}} - \frac{(\gamma-\gamma_f)^{\frac{7}{2}}}{7\cdot 3!} + \frac{(\gamma-\gamma_f)^{\frac{11}{2}}}{11\cdot 5!} - \frac{(\gamma-\gamma_f)^{\frac{11}{2}}}{44\cdot 3!\cdot 3!}\right] \\ &+\sin\gamma_f\left[\frac{2}{3}(\gamma_0-\gamma_f)^{\frac{3}{2}} - \frac{(\gamma_0-\gamma_f)^{\frac{7}{2}}}{7\cdot 3!} + \frac{(\gamma_0-\gamma_f)^{\frac{11}{2}}}{11\cdot 5!} - \frac{(\gamma_0-\gamma_f)^{\frac{11}{2}}}{44\cdot 3!\cdot 3!}\right]\end{aligned} \tag{A.95}$$

and (A.94) becomes:

$$x = x_0 + \frac{-V}{b}\left[2\cos\gamma_f\Big(\sin^{\frac{1}{2}}(\gamma-\gamma_f) - \sin^{\frac{1}{2}}(\gamma_0-\gamma_f)\Big) + \Delta x\right] \tag{A.96}$$

In a similar fashion, we can obtain an expression for y for any γ by beginning with (A.59):

$$y = \frac{V}{b}\Big[\cos\gamma_f \int_{g_0}^{g} \sqrt{\sin g}\, dg + \sin\gamma_f \int_{g_0}^{g} \frac{\cos g}{\sqrt{\sin g}} dg\Big] \tag{A.97}$$

and using Δx from above:

$$y = \frac{V}{b}\Big[\frac{-\Delta x\cos\gamma_f}{\sin\gamma_f} + \sin\gamma_f \int_{u_0}^{u} \frac{du}{u^{\frac{1}{2}}}\Big] \tag{A.98}$$

$$= \frac{V}{b}\Big[- \Delta x \cot\gamma_f + 2\sin\gamma_f \Big[u^{\frac{1}{2}}\Big]_{u_0}^{u}\Big] \tag{A.99}$$

$$= \frac{V}{b}\Big[- \Delta x \cot\gamma_f + 2\sin\gamma_f \Big[\sin^{\frac{1}{2}} g\Big]_{g_0}^{g}\Big] \tag{A.100}$$

$$= \frac{V}{b}\Big[- \Delta x \cot\gamma_f + 2\sin\gamma_f \Big(\sin^{\frac{1}{2}}(\gamma-\gamma_f) - \sin^{\frac{1}{2}}(\gamma_0-\gamma_f)\Big)\Big] \tag{A.101}$$

We will now develop an expression for the total time of flight, t_f (or t_{go}). We begin with the dynamics for γ:

$$\dot{\gamma} = \frac{d\gamma}{dt} = u \tag{A.102}$$

Rearranging gives:

$$dt = \frac{1}{u} d\gamma = \frac{d\gamma}{b\sin^{\frac{1}{2}}(\gamma-\gamma_f)} \tag{A.103}$$

Integrating both sides:

$$\int_0^{t_f} dt = \frac{1}{b}\int_{\gamma_0}^{\gamma_f} \frac{d\gamma}{\sin^{\frac{1}{2}}(\gamma-\gamma_f)} \tag{A.104}$$

or using the substitution $g = \gamma - \gamma_f, dg = d\gamma$:

$$t_f - 0 = \frac{1}{b}\int_{g_0}^{g_f} \frac{dg}{\sin^{\frac{1}{2}} g} \tag{A.105}$$

Previously, we showed:

$$\sqrt{\sin g} \approx \sqrt{g}\sqrt{1 - \frac{g^2}{3!} + \frac{g^4}{5!} - \frac{g^6}{7!}...} = \sqrt{g}\sqrt{1+x} \tag{A.106}$$

where:

$$x = -\frac{g^2}{3!} + \frac{g^4}{5!} - \frac{g^6}{7!}... \tag{A.107}$$

It is also helpful to utilize the following infinite series representation:

$$\frac{1}{(1+x)^{\frac{1}{2}}} \approx 1 - \frac{x}{2} + \frac{3x^2}{8} - \frac{5x^3}{16}... \tag{A.108}$$

$$= 1 - \frac{1}{2}\Big(-\frac{g^2}{3!} + \frac{g^4}{5!} - \frac{g^6}{7!}\Big) + \frac{3}{8}\Big(-\frac{g^2}{3!} + \frac{g^4}{5!} - \frac{g^6}{7!}\Big)^2 + ... \tag{A.109}$$

$$= 1 + \frac{g^2}{12} - \frac{g^4}{240} + \frac{g^4}{96} + ... = 1 + \frac{g^2}{12} + \frac{g^4}{160} + ... \tag{A.110}$$

Substituting into (A.105) yields:

$$t_f \approx \frac{1}{b}\int_{g_0}^{g_f} \frac{1}{\sqrt{g}}\Big(1 + \frac{g^2}{12} + \frac{g^4}{160}\Big)dg \tag{A.111}$$

$$= \frac{1}{b}\int_{g_0}^{g_f}\Big(g^{-\frac{1}{2}} + \frac{g^{\frac{3}{2}}}{12} + \frac{g^{\frac{7}{2}}}{160}\Big)dg \tag{A.112}$$

We now integrate the right-hand side to obtain:

$$t_f = \frac{1}{b}\Big(2g^{\frac{1}{2}} + \frac{g^{\frac{5}{2}}}{30} + \frac{g^{\frac{9}{2}}}{720}\Big)_{g_0}^{g_f} \tag{A.113}$$

$$= \frac{1}{b}\Big[2(\gamma_0 - \gamma_f)^{\frac{1}{2}} + \frac{(\gamma_0 - \gamma_f)^{\frac{5}{2}}}{30} + \frac{(\gamma_0 - \gamma_f)^{\frac{9}{2}}}{720}\Big] \tag{A.114}$$

From the initial conditions, we can now find γ_f, t_f and the position (x, y) for any value of γ.

In summary, we have a solution for a optimal control, $\dot{\gamma}$, designed to minimize control input:

$$u = \pm b\sqrt{\sin(\gamma - \gamma_f)} \tag{A.115}$$

where b is given by:

$$b = \frac{-2V}{x_0}\frac{\sqrt{\sin\left(\gamma_0 - \gamma_f\right)}}{\cos\gamma_f} \tag{A.116}$$

and γ_f is:

$$\gamma_f \approx -\frac{\gamma_0}{2} \tag{A.117}$$

and the final time, t_f, is obtained from:

$$t_f \approx \frac{1}{b}\Big[2(\gamma_0 - \gamma_f)^{\frac{1}{2}} + \frac{(\gamma_0 - \gamma_f)^{\frac{5}{2}}}{30} + \frac{(\gamma_0 - \gamma_f)^{\frac{9}{2}}}{720}\Big] \tag{A.118}$$

Several simplifying assumptions result in a very powerful tool for addressing heading errors. If we assume the value of x_0 is the range, ρ, to the target, and that we reinitialize the problem at every time (such that γ is always γ_0, which is the heading error to the target at that time) then equations (A.115 and A.116) can be combined to produce:

$$u = -\frac{2V}{\rho}\frac{\sin(\gamma_0 - \gamma_f)}{\cos\gamma_f} \tag{A.119}$$

Combining this with approximation for γ_f in equation (A.117) we obtain:

$$u = -\frac{2V}{\rho}\frac{\sin\frac{3\gamma_0}{2}}{\cos\frac{\gamma_0}{2}} \tag{A.120}$$

which is a remarkably good approximation up to $\gamma_0 = 90^o$. The final time, t_f, is computed as before in (A.118); however, it is not necessary for the guidance solution.

This results in a very useful expression for the commanded acceleration of:

$$a_c = V\dot{\gamma} = -\frac{2V^2}{\rho}\frac{\sin\frac{3\gamma_0}{2}}{\cos\frac{\gamma_0}{2}} \tag{A.121}$$

Appendix B

Computer Code for Unconstrained Case

Main Code

```
program MultipleShootingProjectile
!
! Takes the control history and calls the flight module
!
use Physical
implicit none
external MS_function
integer             n_MS, writeme, writelambda, MS_count
double precision    delt, angle_l, angle_r, t_l, t_r, rt(2), &
theta0, theta, r0(2), v0(2), tf, L, D, q
common/GlobalStuff/ delt, angle_l, angle_r, t_l, t_r, rt   , &
```

```
                    theta0, theta, r0   , v0   , tf, n_MS, &
                    writeme, L, D, q, writelambda, MS_count
integer, parameter ::          n_MS_max = 160,             &
                               n_w_max  = n_MS_max*4 - 2,  &
                               L_max    = n_w_max*(3*n_w_max + 13)/2
logical from_previous
integer :: n_w, iflag, info, i, L_HBRD
double precision :: angles(n_MS_max), w0(n_w_max), wf(n_w_max), &
                    F(n_w_max), FF(n_w_max),  &
                    wk(L_max), eps, tol, alt_t, range_t, rm_t, ang_t
logical interpolate
common/Decisions/  interpolate
 300 format( 20x, g20.10 )
!
! Set convergence and tolerance values for the routine HBRD
!
eps = 1.d-12     ! Estimated computational accuracy of F()
tol = 1.d-06     ! Desired convergence accuracy of HBRD
writeme = 1
writelambda = 1
!writeme = 1 ! Don't use this flag for INIT.W except with extreme caution! &
(causes double output in MS_function)
!
! Open the input files
!
open( 10, file = '/Users/emmanuelskamangas/documents/matlab/&
```

```
        preserved code/ideal code with tf/lofted/flight_15.dat' )
!open( 11, file = '/Users/emmanuelskamangas/documents/matlab/&
        preserved code/ideal code with tf/angles_hi_runs.dat' )
open( 12, file = '/Users/emmanuelskamangas/documents/matlab/&
        preserved code/ideal code with tf/Decision.dat' )
open( 13, file = '/Users/emmanuelskamangas/documents/&
        matlab/preserved code/ideal code with tf/lofted/&
        win_15.dat' )
!
! Open the output files
!
open( 20, file = '/Users/emmanuelskamangas/documents/matlab/&
        preserved code/ideal code with tf/lofted/output_15.dat' )
!open( 21, file = '/Users/emmanuelskamangas/documents/matlab/&
        preserved code/ideal code 9-17-18/init_data_hand_hi_24.dat' )
open( 22, file = '/Users/emmanuelskamangas/documents/matlab/&
        preserved code/ideal code with tf/lofted/MS_15.dat' )
open( 23, file = '/Users/emmanuelskamangas/documents/matlab/&
        preserved code/ideal code with tf/lofted/w0_15.dat' )
open( 24, file = '/Users/emmanuelskamangas/documents/matlab/&
        preserved code/ideal code with tf/lofted/w_15.dat' )
!open( 25, file = '/Users/emmanuelskamangas/documents/matlab/&
        preserved code/ideal code 9-17-18/30 km/lfinal.dat' )
!open( 26, file = '/Users/emmanuelskamangas/documents/matlab/&
        preserved code/ideal code 9-17-18/30 km/l2final.dat' )
!
```

```
! Read in the desired time of flight and target position
!
read(10,*) range_t, alt_t, tf
ang_t = range_t / Re
rm_t  = Re + alt_t
rt(1) = rm_t * dcos( ang_t )
rt(2) = rm_t * dsin( ang_t )
write(*,*) tf, rt, vmag0
!
! Read in the decision to use a previous guess of w as w0, or create w0&
    with the new auxilliary method
read(12,*) from_previous
!
! Generate initial guess on w (called w0).  Either read it, or else generate it from a
! guessed trajectory
!
if ( from_previous ) then
    read(13, *) n_MS
    n_w = 4*n_MS + 1
    do i = 1, n_w
        read(13,*) w0(i)
    end do
    !
    ! Initial conditions of the state
    !
    theta0 = w0(n_w)
```

```
    r0(1) = Re
    r0(2) = 0.d0
    v0(1) = Vmag0*dcos(theta0)
    v0(2) = Vmag0*dsin(theta0)
else
    i = 0
    !
    ! Read in the approximate control angles
    !
    ! do while ( .not. eof(11) )
    1000 continue
        i = i + 1
        if ( i > n_MS_max ) stop ' > > > > Exceeded n_MS_max; need to increase it'
        read(11,*, err=2000, end=2000) angles(i)
        angles(i) = angles(i)*deg2rad
    goto 1000
    !end do
    2000 continue
    n_MS = i - 1
    n_w  = 4*n_MS + 1
    !
    ! Use these approx. control angles to create an initial guess on w
    !
    interpolate = .true.
    write(*,*) 'angles = '
    do i = 1, n_MS
```

```
        write(*,*) angles(i)*rad2deg
    end do
    write(*,*) 'number of angles = ', n_MS
    write(*,*) 'length of w = ', n_w
    call init_w(                    &
                angles, n_w,        & ! Given
                w0                  ) ! Yielded
end if
write(*,*) theta0*180/pi
write(*,'( " w0 from init_w =")')
write(*,300)  ((w0(i)), i = 1, n_w)
write(23,300) ((w0(i)), i = 1, n_w)
!
! Solve for the solution the the multiple-shooting problem
!
writeme = 0
interpolate = .false.
!write(*,*) theta0*180/pi
!call MS_function(                &
!                 n_w, w0,        &  ! Given
!                 F, 1            )  ! Yielded
!write(*,*) n_w
!write(*,*) 'F from MS_function'
!write(*,300) ((F(i)), i = 1, n_w)
!write(*,*) theta0*180/pi
wf = w0
```

```
writeme = 0
write(*,*) theta0*180/pi
!stop
MS_count = 0
call HBRD( MS_function, n_w, wf, FF, eps, tol, info, wk, L_max )
!
! Write out the results of HBRD
!
if ( info .ne. 1 ) then
     write(*,*) ' HBRD having difficulties.  info = ', info!
     write(24, '(i5, 100(/,e24.16))') n_MS, (wf(i), i=1, n_w)
else
     write(24, '(i5, 100(/,e24.16))') n_MS, (wf(i), i=1, n_w)
     write(*,  '(" HBRD converged!!!, /, wf =")')
     write(*,  '( 20x, g20.10 )') (wf(i), i = 1, n_w)
     write(*,  '("and FF =")')
     write(*,  '( 20x, g20.10 )') (FF(i), i = 1, n_w)
     write(20, '(" HBRD converged!!!, /, wf =")')
     write(20, '( 20x, g20.10 )') (wf(i), i = 1, n_w)
     write(20, '("and FF =")')
     write(20, '( 20x, g20.10 )') (FF(i), i = 1, n_w)
end if
write(*,*) theta0*180/pi
writeme = 1
writelambda = 1
call MS_function(                &
```

```
                  n_w, wf,        &  ! Given
                  F, 1            )  ! Yielded
write(*,*) 'Final F ='
write(*,300) ((F(i)), i = 1, n_w)
write(*,*) theta0*180/pi
end
```

Code to find initial costates

```
subroutine init_w(                    &
                  angles, n_w,        & ! Given
                  w                   ) ! Yielded
!
! projectile parameters
!
use Physical
implicit none
integer, parameter ::    n_MS_max = 40,              &
                         n_w_max  = n_MS_max*4 - 2
integer n_w
double precision w(n_w), angles(n_MS_max)
integer n_MS, writeme, writelambda
double precision    delt, angle_l, angle_r, t_l, t_r, rt(2), theta0,&
        theta, r0(2), v0(2), tf, L, D, q
!common/GlobalStuff/ delt, angle_l, angle_r, t_l, t_r, rt    , theta0,&
```

```
        theta, r0   , v0   , tf, n_MS, writeme
common/GlobalStuff/ delt, angle_l, angle_r, t_l, t_r, rt   , theta0,&
        theta, r0   , v0   , tf, n_MS, &
        writeme, L, D, q, writelambda
integer npanel, num, TotalNum, i, ii, j, jj, kk, n_RK
double precision :: interval, I4(4,4), lambda_input(4,n_MS*4), z(2), t, r(2), v(2),&
                    angle, state(8), dt_des, h, downrange, height, tkeep, lamdar1, &
                    lamdar2, lamdav1, lamdav2, thetakeep, gamma, tnew, statenew(8),&
                    A(4,4), B(4,n_w_max), dpsi_dx(2,4), dphi_dx(1,4), dot2D, mag2D,&
                    dphi_dx_trn(4), Sbar(4,4), alf(4,4), dpsi_dx_trn(4,2),&
                    temp1(2,4), temp2(2,2), temp2_trn(2,2), temp3(2,4), negz(2),&
                    temp4(4), lamdafinal(4), angle_grid(n_MS), lamda0(4), hdot,&
                    lamda(4), told, angle_time(n_MS), rtest(2), rhat(2), vf_mag,&
                    alfinv(4,4,n_MS), SS(4,4,n_MS-1), Sbar_trn(4,4), Sbar_trnBYB(4),&
                    lamdareset(4), B_trnBYSbar(1,4), SUM(4,4), negSbar(4,4),&
                    dSUM(4,4), temp2_inv(2,2), B_trn(1,4), mtest(2,2), nu0, xhat(2),&
                    uhat(2), vhat(2), alphaprint, sinalpha, cosalpha
external deqs_2D   !!!!!
character *2 passname
logical interpolate
common/Decisions/  interpolate
300 format( 20x, g20.10 )
theta0 = angles(1)
npanel = n_MS - 1
interval = tf / dble(npanel)
n_RK = 8
```

```
nu0 = 0.d0
do i = 1, n_MS
angle_grid(i) = angles(i)
    angle_time(i) = (i - 1)*interval
end do
call ident( I4, 4)
!
! Initial conditions
!
r0(1)     = Re
r0(2)   = 0.d0
v0(1)     = Vmag0*dcos(theta0)
v0(2)     = Vmag0*dsin(theta0)
do i = 1, 4
lamda0(i) = 0.d0
end do
t = 0.d0
r = r0
v = v0
lamda = lamda0
state        = (/ r, v, lamda /)
dt_des   = .001d0
num      = int(interval/dt_des) + 1
TotalNum = int( (num * npanel) / 100 ) + 1
delt     = interval/num
h         = mag2D(r) - Re
```

```
i = 1
kk = 1
if ( mod(i,100) == 0 .and. writeme == 1) then
    downrange = 0.d0
    xhat = (/ dcos(angles(1)), dsin(angles(1)) /)
    vhat = v/mag2D( v )
    uhat = (/ vhat(2), -vhat(1) /)
    sinalpha = dot2D(uhat, xhat)
    cosalpha = dot2D(vhat, xhat)
    alphaprint = datan2(sinalpha, cosalpha)
    write( 21, '(20e24.14)') t, h, downrange, angles(1), state, alphaprint
end if
do ii = 1, npanel
    told = t
    angle_l = angle_grid(ii)
    angle_r = angle_grid(ii + 1)
    t_l     = angle_time(ii)
    t_r     = angle_time(ii + 1)
    call getAandB_2D(                &
            state, t,    &    ! Given
            A, B(:,ii)   )    ! Yielded
    alf  =  -(I4 + A*interval)              !!!
    call inverse( alf, alfinv(:,:,ii), 4 )     !!!
    do jj = 1, num
            tnew  = t + delt
            call RungaKutta(                                        &
```

```
                                deqs_2D, n_RK, state, t, tnew,    &  ! Given
                                statenew                          )  ! Yielded
            state    = statenew
            t    = tnew
            i     = i + 1
            rtest = state(1:2)
            h     = mag2D(rtest) - Re
            if ( mod(i,100) == 0 .and. writeme == 1) then
                angle         = dacos(dot2D(r0, rtest)/(mag2D(r0)*mag2D(rtest)))
                downrange = Re * angle
!                call get_theta(           &
!                                  state,    & ! Given
!                                    theta   ) ! Yielded
                xhat = (/ dcos(theta), dsin(theta) /)
                v = state(3:4)
                vhat = v/mag2D( v )
                uhat = (/ vhat(2), -vhat(1) /)
                sinalpha = dot2D(uhat, xhat)
                cosalpha = dot2D(vhat, xhat)
                alphaprint = datan2(sinalpha, cosalpha)
                write( 21, '(20e24.14)') t, h, downrange, theta, state, alphaprint
            end if
    end do
end do
angle         = dacos(dot2D(r0, rtest)/(mag2D(r0)*mag2D(rtest)))
downrange = Re * angle
```

```
write(*,*) 't, h, downrange, theta, state = '
write(*,300) t, h, downrange, theta, state
!write( 22,'(12g20.10)') t, h, downrange, theta, state
write(*,*) 'rtest = ', rtest
vf_mag = dsqrt( state(3)**2 + state(4)**2 )
CALL getAandB_2D(                       &  ! Need to confirm inputs and outputs!
                    state, t,           &  ! Given
                    A, B(:, npanel + 1)     )  ! Yielded
alfinv(:,:,npanel + 1) = I4
Sbar = -alfinv(:,:,npanel)
SS(:,:,npanel) = Sbar
call trn(Sbar, Sbar_trn, 4, 4)
call trn( B(:,npanel), B_trn, 4, 1 )
call matmult( Sbar_trn, B(:,npanel), Sbar_trnBYB, 4, 4, 1 ) !!!
call matmult( B_trn, Sbar, B_trnBYSbar, 1, 4, 4 )
call matmult( Sbar_trnBYB, B_trnBYSbar, SUM, 4, 1, 4 )
do i = npanel - 1, 1, -1
    call matmult( alfinv(:,:,i), Sbar, negSbar, 4, 4, 4 )
    Sbar = -negSbar
    SS(:,:,i) = Sbar
    call trn(Sbar, Sbar_trn, 4, 4)
    call trn( B(:,i), B_trn, 4, 1 )
    call matmult( Sbar_trn, B(:,i), Sbar_trnBYB, 4, 4, 1 )
    call matmult( B_trn, Sbar, B_trnBYSbar, 1, 4, 4 )
    call matmult( Sbar_trnBYB, B_trnBYSbar, dSUM, 4, 1, 4 )
    SUM = SUM + dSUM
```

```
end do
dpsi_dx      = 0.d0
dpsi_dx(1,1) = 1.d0
dpsi_dx(2,2) = 1.d0
dphi_dx(1,1) = 0.d0
dphi_dx(1,2) = 0.d0
dphi_dx(1,3) = -statenew(3)
dphi_dx(1,4) = -statenew(4)
call matmult( dpsi_dx, SUM, temp1, 2, 4, 4 )         !!!
call trn( dpsi_dx, dpsi_dx_trn, 2, 4 )              !!!
call matmult( temp1, dpsi_dx_trn, temp2, 2, 4, 2 )   !!!
call inverse( temp2, temp2_inv, 2)                 !!!
call matmult( temp2_inv, temp1, temp3, 2, 2, 4)      !!!
call trn( dphi_dx, dphi_dx_trn, 1, 4 )
call matmult( temp3, dphi_dx_trn, negz, 2, 4, 1 )        !!!
z = -negz                                           !!!
call matmult( dpsi_dx_trn, z, temp4, 4, 2, 1 )       !!!
lamdafinal = dphi_dx_trn + temp4                         !!!
write(25, 300) lamdafinal
write(*,*) 'lamdafinal = ', lamdafinal
do i = 1, npanel
    call matmult( SS(:,:,i), lamdafinal, lamdareset, 4, 4, 1 )    !!!
    w((4*i-3):(4*i)) = lamdareset                            !!!
end do
w((n_w-4):(n_w-3)) = z       !!!
w(n_w-2) = nu0
```

```
w(n_w-1) = theta0
w(n_w)   = tf/100.d0
write(*,*) 'lamdafinal = ', lamdafinal
end subroutine init_w


Code to execute the Multiple Shooting Method

subroutine MS_function(                         &
                       n_w, w,                  &  ! Given
                       F, iflag                 )  ! Yielded
!
! projectile parameters
!
use Physical
implicit none
integer n_w, iflag, writeme, writelambda
double precision w(n_w), F(n_w)
integer             n_MS, MS_count
double precision    delt, angle_l, angle_r, t_l, t_r, rt(2), theta0, theta,&
                    r0(2), v0(2), tf, L, D, q
common/GlobalStuff/ delt, angle_l, angle_r, t_l, t_r, rt   , theta0, theta,&
                    r0,    v0,    tf, n_MS, writeme, L, D, q, writelambda, MS_count
integer npanel, num, TotalNum, i, ii, j, jj, kk, last_start, n_RK
double precision :: interval, I4(4,4), lamda_input(4,n_MS), z(2), t, r(2), v(2),&
                    lamdafinal(4),  state(8), dt_des, h, height, tkeep, lamdar1,&
```

```
                    lamdar2, lamdav1, rtest(2), lamdav2, thetakeep, htest, gamma,&
                    tnew, statenew(8), angle, mag2D, dot2D, downrange, dpsi_dx(4,2),&
                    dphi_dx(4),dpsi_dxBYz(4), dr(2), nu0, Ham, vdot(2), alphaold,&
                    alphanew, Hold, Hnew, dH_dt, du_dt
double precision alpha_plot, Hu, Huu, timeplot, dot1, dot2
common/Checkout/ alpha_plot, Hu, Huu, timeplot, dot1, dot2
logical interpolate
integer print_theta
common/Decisions/  interpolate, print_theta
external deqs_2D
300 format( 20x, g20.10 )
write(*,*) 'Beginning of MS_function'
MS_count = MS_count + 1
write(*,*) MS_count
print_theta = 0
npanel = n_MS - 1
n_RK = 8
call ident( I4, 4)
!
! Initial conditions
!
do i = 1, npanel
    lamda_input(:,i) = w((i*4-3):(i*4))
end do
z = w(n_w-4:n_w-3)
nu0 = w(n_w-2)
```

```
theta0 = w(n_w-1)
tf     = w(n_w)*100.d0
interval = tf / dble(npanel)
t = 0.d0
timeplot = 0.d0
r0(1) = Re
r0(2) = 0.d0
v0(1) = Vmag0*dcos(theta0)
v0(2) = Vmag0*dsin(theta0)
r = r0
v = v0
state(1:2) = r
state(3:4) = v
state(5:8) = lamda_input(:,1)
dt_des   = .001d0
num      = int(interval/dt_des) + 1
TotalNum = int( (num * npanel) / 100 ) + 1
delt     = interval/num
h         = dsqrt( r(1)**2 + r(2)**2 ) - Re
downrange     = 0.d0
i = 1
kk = 1
htest = 10
call get_a_2D(                &
                state, t,     &   ! Given
                vdot          )   ! Yielded
```

```
Ham        = state(5)*state(3) + state(6)*state(4) + state(7)*vdot(1)&
            + state(8)*vdot(2)
Hold = Ham
alphaold = alpha_plot
dH_dt = 0.d0
du_dt = 0.d0
! if ( mod(i,1) == 0 .and. writeme == 1) then
if (writeme == 1) then
    write( 22,'(25e24.14)')  t, h, downrange, theta, state, Ham, alpha_plot, Hu,&
    Huu, L, D, q, dot1, dot2, dH_dt, du_dt, vdot(1), vdot(2)
end if
do ii = 1, npanel
    do jj = 1, num
        tnew  = t + delt
        call RungaKutta(                                &
                        deqs_2D, n_RK, state, t, tnew,  &  ! Given
                        statenew                        )  ! Yielded
        state    = statenew
        t    = tnew
        timeplot = t
        i     = i + 1
        rtest = state(1:2)
        h     = mag2D(rtest) - Re
        if ( mod(i,1) == 0 .and. writeme == 1) then
            angle      = dacos(dot2D(r0, rtest)/(mag2D(r0)*mag2D(rtest)))
            downrange  = Re * angle
```

```
            call get_a_2D(                  &
                           state, t,        &   ! Given
                           vdot             )   ! Yielded
            Ham  = state(5)*state(3) + state(6)*state(4) + state(7)*vdot(1)&
                    + state(8)*vdot(2)
            Hnew = Ham
            alphanew = alpha_plot
            dH_dt = (Hnew - Hold)/delt
            du_dt = (alphanew - alphaold)/delt
            Hold = Ham
            alphaold = alpha_plot
            write( 22,'(25e24.14)')  t, h, downrange, theta, state, Ham, alpha_plot,&
            Hu, Huu, L, D, q, dot1, dot2, dH_dt, du_dt, vdot(1), vdot(2)
        end if
    end do
    if ( ii < npanel ) then
        F(ii*4-3:ii*4) = state(5:8) - lamda_input(:,ii+1)
        state(5:8) = lamda_input(:,ii+1)
    end if
end do
    angle         = dacos(dot2D(r0, rtest)/(mag2D(r0)*mag2D(rtest)))
    downrange     = Re * angle
    call get_theta(          &
                    state,  & ! Given
                    theta   ) ! Yielded
    call get_a_2D(                 &
```

```
                    state, t,       &   ! Given
                    vdot            )   ! Yielded
    Ham         = state(5)*state(3) + state(6)*state(4) + state(7)*vdot(1)&
                 + state(8)*vdot(2)
    Hnew = Ham
    alphanew = alpha_plot
    dH_dt = (Hnew - Hold)/delt
    du_dt = (alphanew - alphaold)/delt
if ( writeme == 1) then
    write( 22,'(25e24.14)')  t, h, downrange, theta, state, Ham, alpha_plot, Hu,&
            Huu, L, D, q, dot1, dot2, dH_dt, du_dt, vdot(1), vdot(2)
end if
dpsi_dx      = 0.d0
dpsi_dx(1,1) = 1.d0
dpsi_dx(2,2) = 1.d0
dphi_dx(1:2) = 0.d0
dphi_dx(3:4) = -statenew(3:4)
call matmult( dpsi_dx, z, dpsi_dxBYz, 4, 2, 1 )
lamdafinal = dphi_dx + dpsi_dxBYz
if (writelambda == 1) then
    write(26, 300) lamdafinal
    write(*,*) 'lamdafinal = ', lamdafinal
    write(*,*) 'tf, Ham', tf, Ham
endif
dr = state(1:2) - rt
! Final stuff into F(w)
```

```
last_start = 4*npanel-3
F(last_start:last_start+1)   = dr / 100.d0
F(last_start+2:last_start+5) = state(5:8) - lamdafinal
F(last_start+6) = lamda_input(3,1) - nu0*dcos(theta0)
F(last_start+7) = lamda_input(4,1) - nu0*dsin(theta0)
F(last_start+8) = Ham*200.d0
end subroutine MS_function
```

Code containing the state differential equations

```
subroutine deqs_2D(                    &
                   statecurrent, t,    &  ! Given
                   stateprime          )  ! Yielded
! Calculates the state derivatives.  Uses the get_a routine to find vdot.
implicit none
double precision statecurrent(8), t, stateprime(8)
integer             n_MS
double precision    delt, angle_l, angle_r, t_l, t_r, rt(2), theta0, theta, r0(2),&
                    v0(2), tf
common/GlobalStuff/ delt, angle_l, angle_r, t_l, t_r, rt   , theta0, theta, r0   ,&
                    v0   , tf, n_MS
double precision :: x(4), v(2), lamda_r(2), lamda_v(2), vdot(2), da_a_dx(2,4),&
                    da_dr(2,2), da_dr_trn(2,2), da_dv_trn(2,2), prod(2),&
                    da_dv(2,2), lamdadot_r(2), lamdadot_v(2)
x = statecurrent(1:4)
```

```
v = statecurrent(3:4)
lamda_r = statecurrent(5:6)
lamda_v = statecurrent(7:8)
call get_a_2D(                    &
               statecurrent, t,   &   ! Given
               vdot               )   ! Yielded
call partials_full_2D(                     &
                       statecurrent, t,    &   ! Given
                       da_a_dx             )   ! Yielded
da_dr = da_a_dx(1:2, 1:2)
da_dv = da_a_dx(1:2, 3:4)
call trn( da_dr, da_dr_trn, 2, 2 )
call matmult( -da_dr_trn, lamda_v, lamdadot_r, 2, 2, 1 )
call trn( da_dv, da_dv_trn, 2, 2 )
call matmult( da_dv_trn, lamda_v, prod, 2, 2, 1 )
lamdadot_v = - lamda_r - prod
stateprime = (/ v , vdot, lamdadot_r, lamdadot_v /)   !
end
```

Code to calculate the accelerations (vdot)

```
subroutine get_a_2D(                    &
                       xcurrent, t,     &   ! Given
                       vdot             )   ! Yielded
use Physical
```

```
implicit none
! Takes the current position(r) and velocity (v) and angle of attack
! and calculates the derivative (v and vdot)
!
double precision :: xcurrent(8), t, vdot(2)
integer             n_MS, writeme
double precision    delt, angle_l, angle_r, t_l, t_r, rt(2), theta0, theta, r0(2), v0(2)
common/GlobalStuff/ delt, angle_l, angle_r, t_l, t_r, rt   , theta0, theta, r0   , v0
                    writeme, L, D, q
double precision :: r(2), v(2), interval, slope, xhat(2), vhat(2), uhat(2),
                    sinalpha, cosalpha, alpha, h, rho, drhodh, sos, dsosdh, A, Ahat(2),
                    N, Nhat(2), Nvec(2), g(2), dot2D, mag2D, h_send, h_use
logical interpolate
integer print_theta
common/Decisions/  interpolate, print_theta
r = xcurrent(1:2)
v = xcurrent(3:4)
if (print_theta == 1) then
    write(*,*) 'first print in a_2D', theta
endif
!write(*,*) 'r, v = ', r, v
! Calculate the control (interpolate == 1; get_theta <> 1)
if ( interpolate ) then
    interval = t_r - t_l
    slope = (angle_r - angle_l) / interval
    theta = angle_l + slope*(t - t_l)
```

```
else
    if (print_theta == 1) then
        write(*,*) '2nd print', theta
    endif
    call get_theta(              &
                   xcurrent,     & ! Given
                   theta         ) ! Yielded
    if (print_theta == 1) then
        write(*,*) '3rd print', theta
    endif
end if
xhat = (/ dcos(theta), dsin(theta) /)
if (print_theta == 1) then
    write(*,*) '4th print', theta
endif
vhat = v/mag2D( v )
!write(*,*) 'vhat, mag(v) = ', vhat, mag2D(v)
uhat = (/ vhat(2), -vhat(1) /)
!write(*,*) 'uhat = ', uhat
sinalpha = dot2D(uhat, xhat)
cosalpha = dot2D(vhat, xhat)
alpha = datan2(sinalpha, cosalpha)
!write(*,*) 'alpha = ', alpha
! alphadeg = alpha*180/pi
!
! Calculate height
```

```
!
h = mag2D(r) - Re
h_send = h
!write(*,*) 'height = ', h
!
! Calculate density
!
call shau62( h_send*1000.d0 , rho, drhodh, sos, dsosdh)
if ( h_send < 0.d0 ) then
    h_use = h_send*1000.d0
    rho = 1.3032437899813000d0 - h_use*1.196173013681500d-04&
        + .5d0*h_use*h_use*(4.758851579998464d-09)
    drhodh = -1.196173013681500d-04 + h_use*(4.758851579998464d-09)
!    write(*,*) 't, h', t, h
endif
!
! Calculate axial force
!
A = .5d0 * rho * dot2D(v,v) * S * Ca * 1000d0 ! magnitude of axial force
Ahat = -xhat
Avec = A * Ahat
!write(*,*) 'Avec = ', Avec
! Avec = -A * vhat; only if alpha = 0!!!!!
!
! Calculate normal force vector
!
```

```
N = .5d0 * rho * dot2D(v, v) * S * Cna * alpha * 1000d0 ! magnitude of normal force
Nhat = (/ -Ahat(2), Ahat(1) /)
Nvec = N * Nhat
!write(*,*) 'Nvec = ', Nvec
!
! gravitational force
!
g = -mu * r / (mag2D(r)**3)
!
! total acceleration
vdot = (Avec + Nvec)/m + g
!write(*,*) 'vdot in get_a_2D', vdot
if (print_theta == 1) then
    write(*,*) '5th print', theta
endif
if (writeme == 1) then
    L = N*dcos(alpha) - A*dsin(alpha)
    D = A*dcos(alpha) + N*dsin(alpha)
    q = .5*rho*dot2D(v*1000.d0, v*1000.d0)
endif
end subroutine get_a_2D
```

Code to generate the A and B matrices used to for the costate estimates

```
subroutine getAandB_2D(               &
```

```
                        state, t,      & ! Given
                        A, B           ) ! Yielded
! generates the A and B matricies
implicit none
double precision state(8), t, A(4,4), B(4)
integer             n_MS
double precision    delt, angle_l, angle_r, t_l, t_r, rt(2), theta0, theta, r0(2),&
                    v0(2), tf
common/GlobalStuff/ delt, angle_l, angle_r, t_l, t_r, rt   , theta0, theta, r0   ,&
                    v0   , tf, n_MS
double precision :: I2(2,2), x(8), da_a_dx(2,4), da_dr(2,2), da_dv(2,2),&
                    da_dtheta(2), Z2(2,2)
logical interpolate
common/Decisions/  interpolate
call ident(I2, 2)
x = state
call partials_full_2D(          &   ! da_a_dx = partials_full_2D(x, t)
                       x, t,    &   ! Given
                       da_a_dx  )   ! Yielded
300 format( 20x, g20.10 )
da_dr = da_a_dx(:, 1:2)
da_dv = da_a_dx(:, 3:4)
!write(*,*) 'da_a_dx'
!write(*,300) da_a_dx
A(1:2,1:2) = 0.d0
A(1:2,3:4) = -da_dr
```

```
A(3:4,1:2) = -I2
A(3:4,3:4) = -da_dv
call getpartialda_dtheta_2D(            &     !
                            x, t,       &     ! Given
                            da_dtheta )     ! Yielded
!write(*,*) 'da_dtheta'
!write(*,300) da_dtheta
B = (/ 0.d0, 0.d0, da_dtheta(1:2) /)
end
```

Code to calculate the angle of attack

```
subroutine get_theta(           &
                     state,     & ! Given
                     control    ) ! Yielded
use Physical
implicit none
double precision state(8), control
integer             n_MS, iter
integer, parameter  :: itmax = 100
double precision    delt, angle_l, angle_r, t_l, t_r, rt(2), theta0, theta, r0(2),&
                    v0(2), tf
common/GlobalStuff/ delt, angle_l, angle_r, t_l, t_r, rt   , theta0, theta, r0   ,&
                    v0   , tf, n_MS
double precision :: v(2), lamda_v(2), gamma, epsilon, check, diff, vhat(2),&
```

```
                    uhat(2), dot1, dot2, alpha, alpha1, alpha2, F_alpha,&
                    F_prime, alpha_new, mag2D, dot2D, alpha_old, check2
double precision alpha_plot, Hu, Huu, timeplot
common/Checkout/ alpha_plot, Hu, Huu, timeplot, dot1, dot2
!
! Initialize variables
!
v       = state(3:4)
lamda_v = state(7:8)
gamma   = datan2(v(2), v(1))
epsilon = 1.d-08
check   = 1.d-16
check2  = 1.d-12
!epsilon = 0.00001d0
!check   = 1.d-12
diff    = 1000.d0
vhat    = v / mag2D(v)
uhat    = (/ vhat(2), -vhat(1) /)
dot1    = dot2D(lamda_v, vhat)
dot2    = dot2D(lamda_v, uhat)
!
! Solve for angle of attack
!
if ( dabs(dot2) < epsilon ) then
    alpha = 0.d0
else
```

```
    !
    ! Two roots of the approximate solution of alpha (small angle approx.)
    !
    alpha1 = (-dot1*(2*Cna - Ca) - dsqrt(((2*Cna - Ca)*dot1)**2  &
        + 4*dot2**2*Cna*(Cna - Ca)))/(2*dot2*Cna)
    alpha2 = (-dot1*(2*Cna - Ca) + dsqrt(((2*Cna - Ca)*dot1)**2  &
        + 4*dot2**2*Cna*(Cna - Ca)))/(2*dot2*Cna)
    !
    ! Choose the root with the smallest magnitude to initialize&
    Newton's Method
    !
    if ( dabs(alpha1) < dabs(alpha2)) then
        alpha = alpha1
    else
        alpha = alpha2
    end if
endif
    !
    ! Newton's method iteration to solve for angle of attack
    !
iter = 0
do while ( diff > check .and. iter < itmax)
iter = iter + 1
    F_alpha = dot1*(Ca - Cna)*dsin(alpha) - dot2*Cna*alpha*dsin(alpha) &
            - dot1*Cna*alpha*dcos(alpha) + dot2*(Cna - Ca)*dcos(alpha)
    F_prime = dot1*(Ca - Cna)*dcos(alpha) - dot2*Cna*dsin(alpha)       &
```

```
            - dot2*Cna*alpha*dcos(alpha) - dot1*Cna*dcos(alpha)        &
            + dot1*Cna*alpha*dsin(alpha) - dot2*(Cna - Ca)*dsin(alpha)
    alpha_new = alpha - F_alpha/F_prime
    diff      = dabs(alpha - alpha_new)
    alpha_old = alpha
    alpha     = alpha_new
end do
if ( alpha > 2*pi ) then
    alpha = 2*pi - alpha
end if
!
! Set the control (theta) based on the angle of attack
!
control = gamma - alpha
alpha_plot = alpha
Hu  = F_alpha
Huu = F_prime
end subroutine get_theta


Code to calculate the partial derivative of the acceleration with&
respect to the control

subroutine getpartialda_dtheta_2D(         &
                                  x, t,    &  ! Given
                                  da_dtheta )  ! Yielded
```

```
! finds the partial of acceleration wrt theta
use Physical
implicit none
double precision data, x(8), t, da_theta(2)
integer             n_MS
double precision    delt, angle_l, angle_r, t_l, t_r, rt(2), theta0, theta, r0(2),&
                    v0(2), tf
common/GlobalStuff/ delt, angle_l, angle_r, t_l, t_r, rt   , theta0, theta, r0   ,&
                    v0   , tf, n_MS
double precision :: r(2), v(2), slope, xhat(2), int, vhat(2), uhat(2), sinalpha,&
                    cosalpha, h, dt_des, tkeep, lamdar1, lamdar2, lamdav1, lamdav2,&
                    thetakeep, htest, dot2D, alpha, A, N, dN_da, dA_da, dL_da,&
                    dD_da, rho, drhodh, sos, dsosdh, da_dtheta(2), mag2D, h_send,&
                    h_use
r = x(1:2)
v = x(3:4)
! Calculate the control
int = t_r - t_l
slope = (angle_r - angle_l) / int
theta = angle_l + slope*(t - t_l)
xhat = (/ dcos(theta), dsin(theta) /)
vhat = v / mag2D(v)
uhat = (/ vhat(2), -vhat(1) /)
sinalpha = dot2D(uhat, xhat)
cosalpha = dot2D(vhat, xhat)
alpha = datan2(sinalpha, cosalpha)
```

```
!
! Calculate height
!
h = mag2D(r) - Re
h_send = h
!
! Calculate density
!
call shau62( h_send*1000.d0 , rho, drhodh, sos, dsosdh)
if ( h_send < 0.d0 ) then
    h_use = h_send*1000.d0
    rho = 1.3032437899813000d0 - h_use*1.196173013681500d-04&
            + .5d0*h_use*h_use*(4.758851579998464d-09)
    drhodh = -1.196173013681500d-04 + h_use*(4.758851579998464d-09)
!    h_send = 0.d0
!    write(*,*) 't, h', t, h
endif
!
! Calculate axial force
!
A = .5d0 * rho * dot2D(v, v) * S * Ca * 1000d0    ! magnitude of axial force
!
! Calculate normal force vector
!
N = .5d0 * rho * dot2D(v, v) * S * Cna * alpha*1000   ! magnitude of normal force
! Calculate the partials of the normal force, axial force, lift and drag wrt alpha
```

```
dN_da = .5d0 * rho * dot2D(v, v) * S * Cna * 1000.d0
dA_da = 0.d0
dL_da = (dN_da - A) * cosalpha - (N + dA_da) * sinalpha
dD_da = (dN_da - A) * sinalpha + (N + dA_da) * cosalpha
! calculate the partial of acceleration wrt theta
da_dtheta = -(dL_da * uhat - dD_da * vhat) / m
end subroutine getpartialda_dtheta_2D


Code to calculate the partial derivatives of the acceleration
with respect to the state

subroutine partials_full_2D(          &
                            x, t,     & ! Given
                            da_a_dx   ) ! Yielded

! function takes the current state and returns partials of the total
! acceleration
implicit none
double precision x(8), t, da_a_dx(2,4)
integer             n_MS
double precision    delt, angle_l, angle_r, t_l, t_r, rt(2), theta0, theta, r0(2),&
                    v0(2), tf
common/GlobalStuff/ delt, angle_l, angle_r, t_l, t_r, rt   , theta0, theta, r0   ,&
                    v0   , tf, n_MS
integer i, n
```

```
double precision :: del, s, xkeep(8), xpass(8), a_f(2), a_b(2)
s = .00001
xkeep = x
n = 4        ! Need to define
do i = 1, n
    xpass = xkeep
    del = max(1.d0, dabs(xpass(i))) * s    ! Need the max function
    xpass(i) = xkeep(i) + del
    call get_a_2D(               &
                   xpass, t,     &  ! Given
                   a_f           )  ! Yielded
    xpass(i) = xkeep(i) - del
    call get_a_2D(               &
                   xpass, t,     &  ! Given
                   a_b           )  ! Yielded
    da_a_dx(:,i) = (a_f - a_b) / (2 * del)
end do
end subroutine partials_full_2D
```

Code with the Runga-Kutta integration routine

```
subroutine RungaKutta(             &
            defun, n, x0, t0, t,   &  ! Given
            xnext                  )  ! Yielded
implicit none
```

```
integer :: n
double precision :: x0(n), t0, t, xnext(n)
double precision :: delt, halfdt, thalf, K1(n), K2(n), K3(n), K4(n),  &
    xmid1(n), xmid2(n), xend(n)
external defun
delt   = t - t0
halfdt = delt / 2.d0
thalf  = t0 + halfdt
call defun(x0, t0, K1)
xmid1 = x0 + K1*halfdt
call defun(xmid1, thalf, K2)
xmid2 = x0 + K2*halfdt
call defun(xmid2, thalf, K3)
xend = x0 + K3*delt
call defun(xend, t, K4)
xnext = x0 + delt*(K1 + 2.d0*K2 + 2.d0*K3 + K4)/6.d0
!Write(*,*) 'new state in Runga-Kutta', xnext
end subroutine RungaKutta
```

Subroutine with the atmospheric model

```
C-----------------------------------------------------------------------
C     MODEL IS SHAU'S APPROXIMATION TO 62 ATMOSPHERE (WITH AND WITHOUT
C       PARTIAL DERIVS)
```

```
C-----------------------------------------------------------------------
      SUBROUTINE  SHAU62(H, RHO,  DRHODH,  ASP, DASPDH)
C-----------------------------------------------------------------------
C     PURPOSE:  TO APPROXIMATE THE 1962 STANDARD ATMOSPHERE BY
C  A RATIONAL POLYNOMIAL FIT FOR BOTH DENSITY AND
C  AIR SPEED. FIRST DERIVATIVES WITH RESPECT TO H
C  ARE COMPUTED.
C
C     REMARK:    IN DR. SHAU'S RATIONAL POLYNOMIALS ALTITUDE IS
C  IN <KM>.  THE CONVERSION FROM <M> TO <KM> IS MADE
C  WITHIN ATMSHU.  THE USER SUPPLIES ALTITUDE IN <M>.
C
C     INPUT:  H   ALTITUDE (IN <M>)
C
C     OUTPUT:  RHO   DENSITY  <KG*S**2/M**4>
C  DRHODH   D(RHO)/D(H)  <KG*S**2/M**5>
C  ASP   AIR SPEED  <M/S>
C  DASPDH   D(ASP)/D(H)  <M/S**2>
C-----------------------------------------------------------------------
C     COMMON BLOCKS:  NONE
C     SUBROUTINES REFERENCED: *RAFKT (POLY13)     * ==> DIRECT CALL
C-----------------------------------------------------------------------
C     PROGRAMMED BY: DR. G. SHAU, DFVLR-OBERPFAFFENHOFEN
C     STATUS:      JUNE, 1986
C-----------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)
```

```
      DIMENSION AZ(6),AN(6),RZ(6),RN(6)
C-----------------------------------------------------------------------
C     DATA:  SCHALLGESCHWINDIGKEIT(M/S) ALS FUNKTION VON H(KM)
C-----------------------------------------------------------------------
      DATA NAZ/6/,NAN/1/
      DATA AZ/
     1        -0.16216846484D-06,    0.62918764872D-04,
     2        -0.82202821737D-02,    0.43395312463D+00,
     3        -0.85038553728D+01,    0.34780109374D+03/
      DATA AN/  0.10000000000D+01, 5*0.D0/
C-----------------------------------------------------------------------
C     DATA:  LUFTDICHTE ALF FUNKTION VON H(KM)
C-----------------------------------------------------------------------
      DATA RHOC/.1111111111111111D0/, NRZ/4/,NRN/6/
      DATA RZ/
     1       -0.687778748290D-07,   0.231496308862D-04,
     2       -0.246069335862D-02,   0.860913681426D-01, 2*0.D0/
      DATA RN/
     1       -0.164740052195D-08,   0.533547525341D-06,
     2       -0.557673532116D-04,   0.219980670895D-02,
     3       -0.310471410480D-01,   0.648041700234D+00/
C
C     NEUW WERTE MIT MAX REL. ERROR ABOUT 12%.
C-----------------------------------------------------------------------
C     DENSITY MODEL:
C-----------------------------------------------------------------------
```

```
      HKM = H/1000.D0
      IF (H .LT. 0.D0)  HKM = 0.D0
      CALL  RAFKT(HKM,RZ(1),RN(1),NRZ,NRN,RHO0,RHO1,RHO2,RHO3)
      S1=DEXP(-HKM*RHOC)
      RHO=RHO0*S1
      S2=RHO1*S1
      S1=-RHO*RHOC
      RHO1=S1+S2
C-----------------------------------------------------------------------
C     AIR SPEED:
C-----------------------------------------------------------------------
      CALL  RAFKT(HKM,AZ,AN,NAZ,NAN,ASP,DASP,D2ASP,D3ASP)
C-----------------------------------------------------------------------
C     CONVERT DERIVATIVES TO M
C-----------------------------------------------------------------------
      DRHODH = RHO1*1.D-03
      DASPDH = DASP*1.D-03
C
      ge = 9.81d0
      RHO = RHO * ge
      DRHODH = DRHODH * ge
C
      RETURN
      END
C
C     END OF SUBROUTINE: SHAU62
```

```
C
C-----------------------------------------------------------------------
      SUBROUTINE  RAFKT(A,CZ,CN,IZ,IN,F,F1,F2,F3)
C-----------------------------------------------------------------------
C     PURPOSE:   TO EVALUATE A RATIONAL POLYNOMIAL ALONG WITH
C   DERIVATIVES
C
C     INPUT:   A    INDEPENDENT VARIABLE
C   CZ    COEFFICIENTS OF NUMERATOR POLYNOMIAL
C   CN    COEFFICIENTS OF DENOMINATOR POLYNOMIAL
C   IZ    NUMBER OF CZ COEFFICIENTS
C   IN    NUMBER OF CN COEFFICIENTS
C
C     OUTPUT:   F    RATIONAL POLYNOMIAL AT A
C   F1    FIRST  DERIVATIVE OF F AT A
C   F2    SECOND DERIVATIVE OF F AT A
C   F3    THIRD  DERIVATIVE OF F AT A
C-----------------------------------------------------------------------
C     COMMON BLOCKS:  NONE
C     SUBROUTINES REFERENCED: *RAFKT (POLY13)     * ==> DIRECT CALL
C-----------------------------------------------------------------------
C     PROGRAMMED BY: DR. G. SHAU, DFVLR-OBERPFAFFENHOFEN
C     STATUS:      JUNE, 1986
C-----------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION  CZ(6),CN(6)
```

```
REAL*8  N,N1,N2,N3
Z=0.D0
Z1=0.D0
Z2=0.D0
Z3=0.D0
N=0.D0
N1=0.D0
N2=0.D0
N3=0.D0
CALL  POLY13(A,CZ,Z,Z1,Z2,Z3,IZ)
IF(IN.EQ.1)  GOTO  1
CALL  POLY13(A,CN,N,N1,N2,N3,IN)
S1=N*Z1
S2=Z*N1
P1=S1-S2
S1=N*Z2
S2=Z*N2
P2=S1-S2
S1=N*P2
S2=P1*N1
S2=S2+S2
P3=S1-S2
S1=N1*Z2
S2=N*Z3
S3=Z1*N2
S4=Z*N3
```

```
      P4=S1+S2-S3-S4
      S1=N*P4
      S2=P2*N1
      S3=P1*N2
      S3=S3+S3
      P5=S1-S2-S3
      S1=N*P5
      S2=P3*N1
      S2=S2+S2+S2
      P6=S1-S2
      S1=1.D0/N
      S2=S1*S1
      S3=S1*S2
      S4=S2*S2
      F=Z*S1
      F1=P1*S2
      F2=P3*S3
      F3=P6*S4
      RETURN
    1 F=Z
      F1=Z1
      F2=Z2
      F3=Z3
      RETURN
      END
C
```

```
C     END OF SUBROUTINE: RAFKT
C
C-----------------------------------------------------------------------
      SUBROUTINE  POLY13(A,C,V,VA1,VA2,VA3,NA)
C-----------------------------------------------------------------------
C     V = OUTPUT = V(A)
C     VA1 = DV / DA
C     VA2 = DVA1 / DA = DV2 / DA2
C     VA3 = DVA2 / DA = DV3 / DA3
C
C
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION  C(6)
      V=0.D0
      VA1=0.D0
      VA2=0.D0
      VA3=0.D0
      NA1=NA-1
      NA2=NA-2
      NA3=NA-3
      DO  1  I=1,NA
      S=C(I)
      V=V*A+S
      IF(I.GT.NA1)  GOTO  1
      S1=DBLE(NA-I)
```

```
      VA1=VA1*A+S1*S
      IF(I.GT.NA2)  GOTO  1
      S1=DBLE(NA1-I)*S1
      VA2=VA2*A+S1*S
      IF(I.GT.NA3)  GOTO  1
      S1=DBLE(NA2-I)*S1
      VA3=VA3*A+S1*S
    1 CONTINUE
      RETURN
      END
C
C     END OF SUBROUTINE: POLY13
C
```

File with the physical parameters of the problem

```
module Physical
!
!  Module to define the physical parameters of the simulation
!
implicit none
double precision, parameter :: &
    Re  = 6371.d0,        &   ! Radius of the Earth (km)
    mu  = 398600.5d0,     &   ! Gravitational Constant (km^3/s^2)
    S   = 0.005352d0,     &   ! Reference area of the projectile (m^2)
    m   = 12.d0,          &   ! Mass of the projectile (kg)
```

```
    Ca  = 0.15d0,          &   ! Axial Coefficient of the projectile
    vmag0 = 1.26d0,        &   ! km/s
    Cna = 5.d0,            &   ! Normal-force curve-slope &
                                 coefficient of the projectile (per radian)
    pi = 3.141592653589793d0  ! pi (of course)
double precision, parameter :: deg2rad = 0.017453292519943294d0, &
                               rad2deg = 57.295779513082323d0
end module Physical
```

angles.dat input file

```
       74.2500000000
       75.5647578719
       77.0162254523
       78.6006293255
       80.3149650228
       82.1569597566
       84.1215259903
       86.2053021559
       88.4032726076
       90.7093442029
       93.1161969477
       95.6151934858
       98.1963589162
      100.8484396497
```

```
       103.5590449600
       106.3160734600
       109.1031909671
       111.9073646969
       114.7144495476
       117.5107389803


flight.dat input file

         43.7200000000
         25.0272763962
          0.0000000000


decision.dat input file

.true.
```

Appendix C

Computer Code for Dynamic Pressure Constraint

Subroutines that are identical to the unconstrained case are not listed.

Main Code

```
program MultipleShootingProjectile

!
! Takes the control history and calls the flight module
!

use Physical
implicit none
external MS_function
```

```
integer             n_MS, writeme, npanels(3), MS_iter
double precision    delt(3), angle_l, angle_r, t_l, t_r, rt(2), theta0,&
                    theta, r0(2), v0(2), tf, L, D, q, qmin, t1, t2
common/GlobalStuff/ delt,    angle_l, angle_r, t_l, t_r, rt   , theta0,&
                    theta, r0   , v0   , tf, L, D, q, qmin, t1, t2, writeme,&
                    n_MS, npanels, MS_iter
integer, parameter ::          n_MS_max = 40,                    &
                               n_w_max  = 4*(n_MS_max - 1) +7 ,  &
                               L_max    = n_w_max*(3*n_w_max + 13)/2
logical from_previous
integer :: n_w, iflag, info, i, L_HBRD
double precision :: angles(n_MS_max), w0(n_w_max), wf(n_w_max), F1(n_w_max),&
                    FF(n_w_max), wk(L_max), eps, tol, alt_t, range_t, rm_t,&
                    ang_t, nu_s, GG(n_w_max)
logical interpolate
common/Decisions/  interpolate
double precision wprevious(n_w_max), Fprevious(n_w_max)
common/Fcheck/ wprevious, Fprevious
 300 format( 20x, g20.10 )
!
! Set convergence and tolerance values for the routine HBRD
!
eps = 1.d-12     ! Estimated computational accuracy of F()
tol = 1.d-06     ! Desired convergence accuracy of HBRD
writeme = 0
!writeme = 1 ! Don't use this flag for INIT.W except with extreme caution!&
```

```
(causes double output in MS_function)
!
! Open the input files
!
open(10,file= '/Users/emmanuelskamangas/documents/matlab/preserved code/&
                single point with H/runs/2 kms/40 km/flight_20f_2625.dat')
!open( 11, file = '/Users/emmanuelskamangas/documents/matlab/preserved code/&
                single point with H/angles.dat' )
!open( 12, file = '/Users/emmanuelskamangas/documents/matlab/preserved code/&
                single point with H/Decision.dat' )
!open( 13, file = '/Users/emmanuelskamangas/documents/matlab/preserved code/&
                single point with H/win.dat' )
open(13,file= '/Users/emmanuelskamangas/documents/matlab/preserved code/&
                single point with H/runs/2 kms/40 km/win20f_2625.dat' )



!
! Open the output files
!
open(20,file= '/Users/emmanuelskamangas/documents/matlab/preserved code/&
                single point with H/runs/2 kms/40 km/output_20f_2625.dat')
!open(21, file = '/Users/emmanuelskamangas/documents/matlab/preserved code/&
                single point with H/init_data_H6a.dat' )
open(22,file= '/Users/emmanuelskamangas/documents/matlab/preserved code/&
                    single point with H/runs/2 kms/40 km/MS_20f_2625.dat' )
!open( 23, file = '/Users/emmanuelskamangas/documents/matlab/preserved code/&
```

```
                    single point with H/w0.dat' )
open( 24, file = '/Users/emmanuelskamangas/documents/matlab/preserved code/&
                single point with H/runs/2 kms/40 km/w_20f_2625.dat' )
!open( 25, file = '/Users/emmanuelskamangas/documents/matlab/preserved code/&
                single point with H/newton_H6a.dat' )
!open( 26, file = '/Users/emmanuelskamangas/documents/matlab/preserved code/&
                single point with H/alpha_data_H6a.dat' )
!open( 27, file = '/Users/emmanuelskamangas/documents/matlab/preserved code/&
                single point with H/qdot_H6a.dat' )
!open( 28, file = '/Users/emmanuelskamangas/documents/matlab/preserved code/&
                single point with H/alpha_check_H6a.dat' )
!open( 29, file = '/Users/emmanuelskamangas/documents/matlab/preserved code/&
                single point with H/lambdacheck_H6a.dat' )
!open( 30, file = '/Users/emmanuelskamangas/documents/matlab/preserved code/&
                single point with H/wf_Ham.dat' )
!open( 31, file = '/Users/emmanuelskamangas/documents/matlab/preserved code/&
                single point with H/alpha_compare_Ham.dat' )
!open( 32, file = '/Users/emmanuelskamangas/documents/matlab/preserved code/&
                single point with H/Ham_check.dat' )
!
! Read in the desired time of flight and target position
!
read(10,*) range_t
read(10,*) alt_t
read(10,*) vmag0
read(10,*) qmin          ! qmin is read in in Pascals (N/m^2)
```

```
qmin = qmin * 1000.d0    ! converts to kN/km^2
write(*,*) range_t, alt_t, vmag0, qmin
ang_t = range_t / Re
rm_t  = Re + alt_t
rt(1) = rm_t * dcos( ang_t )
rt(2) = rm_t * dsin( ang_t )
MS_iter = 1
!write(*,*) tf, rt
!
! Read initial guess on w (called w0)
!
read(13, *) npanels(1), npanels(2)
read(13, *) n_MS
write(24, '(2i10, /, i10)') npanels(1), npanels(2), n_MS
n_w = 4*(n_MS-1) + 8
write(*,*) n_w
do i = 1, n_w
    read(13, *) w0(i)
end do
theta0 = w0(n_w-4)
nu_s = w0(n_w-3)
t1   = w0(n_w-2)*100.d0      ! 100 is the scaling factor for time
tf   = w0(n_w)*100.d0
write(*,*) 'npanels(1, 2), t1, tf, n_MS, n_w', npanels(1), npanels(2), t1, tf, n_MS, n_w
!
! Initial conditions of the state
```

```
!
r0(1) = Re
r0(2) = 0.d0
v0(1) = Vmag0*dcos(theta0)
v0(2) = Vmag0*dsin(theta0)
write(*,*) theta0*180/pi
do i =1, n_w
    wprevious(i) = 0.d0
    Fprevious(i) = 0.d0
end do
!write(*,*) theta0*180/pi
write(*,'( " w0 =")')
write(*,300)  ((w0(i)), i = 1, n_w)
write(23,300) ((w0(i)), i = 1, n_w)
!
! Solve for the solution the the multiple-shooting problem
!
interpolate = .false.
!write(*,*) theta0*180/pi
writeme = 0
write(*,*) 'before MS_Function', eps, tol
!call MS_function(                &
!                 n_w, w0,        &  ! Given
!                 F1, 1            )  ! Yielded
!write(*,*) 'after MS_Function', eps, tol
!write(*,*) n_w
```

```
!write(*,*) 'F from MS_function'
!write(*,300) ((F1(i)), i = 1, n_w)
!write(*,*) n_w
wf = w0
writeme = 0
call HBRD( MS_function, n_w, wf, FF, eps, tol, info, wk, L_max )
!
! Write out the results of HBRD
!
if ( info .ne. 1 ) then
    write(*,*) ' HBRD having difficulties.  info = ', info
    write(*,'( " wf =")')
    write(*,300)  ((wf(i)), i = 1, n_w)
    write(*,*) 'Final F out of HBRD ='
    write(*,300) ((FF(i)), i = 1, n_w)
else
!     write(24, '(i5, 100(/,e24.16))') n_MS, (wf(i), i=1, n_w)
     write(*,  '(" HBRD converged!!!, /, wf =")')
     write(*,  '( 20x, g20.10 )') (wf(i), i = 1, n_w)
     write(*,  '("and FF =")')
     write(*,  '( 20x, g20.10 )') (FF(i), i = 1, n_w)
     write(20, '(" HBRD converged!!!, /, wf =")')
     write(20, '( 20x, g20.10 )') (wf(i), i = 1, n_w)
     write(20, '("and FF =")')
     write(20, '( 20x, g20.10 )') (FF(i), i = 1, n_w)
end if
```

```
!write(*,*) 'MS function iterations = ', MS_iter
writeme = 1
call MS_function(                &
                 n_w, wf,        &  ! Given
                 GG, 1            )  ! Yielded
write(24, '( 100(e24.16, /))') (wf(i), i = 1, n_w)
write(*,'( " wf =")')
write(30,'( " wf =")')
write(*,300)  ((wf(i)), i = 1, n_w)
write(30,300)  ((wf(i)), i = 1, n_w)
write(*,*) 'Final F ='
write(*,300) ((GG(i)), i = 1, n_w)
write(30,*) 'Final F ='
write(30,300) ((GG(i)), i = 1, n_w)
write(*,*) theta0*180/pi
end
```

Code to find initial costates

```
subroutine init_w(                  &
                  angles, n_w,      & ! Given
                  w                 ) ! Yielded
!
! projectile parameters
!
```

```
use Physical
implicit none
integer, parameter ::    n_MS_max = 40,              &
                         n_w_max  = n_MS_max*4 + 1
integer n_w
double precision w(n_w), angles(n_MS_max)
integer             n_MS, writeme, npanels(3)
double precision    delt(3), angle_l, angle_r, t_l, t_r, rt(2), theta0, theta,&
                    r0(2), v0(2), tf, L, D, q, t1, t2
common/GlobalStuff/ delt,    angle_l, angle_r, t_l, t_r, rt   , theta0, theta,&
                    r0   , v0   , tf, L, D, q, t1, t2, writeme, n_MS, npanels
integer npanel, num, TotalNum, i, ii, j, jj, kk, n_RK
double precision :: interval, I4(4,4), lambda_input(4,n_MS*4), z(2), t, r(2), v(2),&
                    angle, state(8), dt_des, h, downrange, height, tkeep, lamdar1,&
                    lamdar2, lamdav1, lamdav2, thetakeep, gamma, tnew, statenew(8),&
                    A(4,4), B(4,n_w_max), dpsi_dx(2,4), dphi_dx(1,4), dot2D, mag2D,&
                    dphi_dx_trn(4), Sbar(4,4), alf(4,4), dpsi_dx_trn(4,2), temp1(2,4),&
                    temp2(2,2), temp2_trn(2,2), temp3(2,4), negz(2), temp4(4),&
                    lamdafinal(4), angle_grid(n_MS), lamda0(4), hdot, lamda(4), told,&
                    angle_time(n_MS), rtest(2), rhat(2), vf_mag, alfinv(4,4,n_MS),&
                    SS(4,4,n_MS-1), Sbar_trn(4,4), Sbar_trnBYB(4), lamdareset(4),&
                    B_trnBYSbar(1,4), SUM(4,4), negSbar(4,4), dSUM(4,4),&
                    temp2_inv(2,2), B_trn(1,4), mtest(2,2), nu0, nus
external deqs_2D   !!!!!
character *2 passname
logical interpolate
```

```
common/Decisions/  interpolate
300 format( 20x, g20.10 )
theta0 = angles(1)
npanel = n_MS - 1
interval = tf / dble(npanel)
n_RK = 8
nu0 = 0.d0
nus = 0.d0
do i = 1, n_MS
angle_grid(i) = angles(i)
    angle_time(i) = (i - 1)*interval
end do
call ident( I4, 4)
!
! Initial conditions
!
r0(1)     = Re
r0(2)   = 0.d0
v0(1)     = Vmag0*dcos(theta0)
v0(2)     = Vmag0*dsin(theta0)
do i = 1, 4
lamda0(i) = 0.d0
end do
t = 0.d0
r = r0
v = v0
```

```
lamda = lamda0
state        = (/ r, v, lamda /)
dt_des   = .001d0
num      = int(interval/dt_des) + 1
TotalNum = int( (num * npanel) / 100 ) + 1
delt     = interval/num
h         = mag2D(r) - Re
i = 1
kk = 1
if ( mod(i,100) == 0 .and. writeme == 1) then
    downrange = 0.d0
    write( 21,'(12e24.14)') t, h, downrange, angles(1), state
end if
do ii = 1, npanel
    told = t
    angle_l = angle_grid(ii)
    angle_r = angle_grid(ii + 1)
    t_l     = angle_time(ii)
    t_r     = angle_time(ii + 1)
    call getAandB_2D(                &
            state, t,    &    ! Given
            A, B(:,ii)   )    ! Yielded
    alf  =  -(I4 + A*interval)              !!!
    call inverse( alf, alfinv(:,:,ii), 4 )     !!!
    do jj = 1, num
            tnew  = t + delt(1)
```

```
            call RungaKutta(                                        &
                                deqs_2D, n_RK, state, t, tnew,      &  ! Given
                                statenew                            )  ! Yielded
            state    = statenew
            t    = tnew
            i     = i + 1
            rtest = state(1:2)
            h     = mag2D(rtest) - Re
            if ( mod(i,100) == 0 .and. writeme == 1) then
                angle         = dacos(dot2D(r0, rtest)/(mag2D(r0)*mag2D(rtest)))
                downrange = Re * angle
                write( 21,'(12e24.14)') t, h, downrange, theta, state
            end if
    end do
end do
angle         = dacos(dot2D(r0, rtest)/(mag2D(r0)*mag2D(rtest)))
downrange = Re * angle
write(*,*) 't, h, downrange, theta, state = '
write(*,300) t, h, downrange, theta, state
!write( 22,'(12g20.10)') t, h, downrange, theta, state
write(*,*) 'rtest = ', rtest
vf_mag = dsqrt( state(3)**2 + state(4)**2 )
CALL getAandB_2D(                       &  !
                    state, t,           &  ! Given
                    A, B(:, npanel + 1)     )  ! Yielded
alfinv(:,:,npanel + 1) = I4
```

```
Sbar = -alfinv(:,:,npanel)
SS(:,:,npanel) = Sbar
call trn( B(:,npanel + 1), B_trn, 4, 1 )
call matmult( B(:, npanel + 1), B_trn, SUM, 4, 1, 4 )
do i = npanel, 1, -1
    call matmult( alfinv(:,:,i), Sbar, negSbar, 4, 4, 4 )
    Sbar = -negSbar
    SS(:,:,i) = Sbar
    call trn(Sbar, Sbar_trn, 4, 4)
    call trn( B(:,i), B_trn, 4, 1 )
    call matmult( Sbar_trn, B(:,i), Sbar_trnBYB, 4, 4, 1 )
    call matmult( B_trn, Sbar, B_trnBYSbar, 1, 4, 4 )
    call matmult( Sbar_trnBYB, B_trnBYSbar, dSUM, 4, 1, 4 )
    SUM = SUM + dSUM
end do
dpsi_dx      = 0.d0
dpsi_dx(1,1) = 1.d0
dpsi_dx(2,2) = 1.d0
dphi_dx(1,1) = 0.d0
dphi_dx(1,2) = 0.d0
dphi_dx(1,3) = -statenew(3)
dphi_dx(1,4) = -statenew(4)
call matmult( dpsi_dx, SUM, temp1, 2, 4, 4 )         !!!
call trn( dpsi_dx, dpsi_dx_trn, 2, 4 )              !!!
call matmult( temp1, dpsi_dx_trn, temp2, 2, 4, 2 )   !!!
call inverse( temp2, temp2_inv, 2)                 !!!
```

```
call matmult( temp2_inv, temp1, temp3, 2, 2, 4)      !!!
call trn( dphi_dx, dphi_dx_trn, 1, 4 )
call matmult( temp3, dphi_dx_trn, negz, 2, 4, 1 )        !!!
z = -negz                                              !!!
call matmult( dpsi_dx_trn, z, temp4, 4, 2, 1 )       !!!
lamdafinal = dphi_dx_trn + temp4                          !!!
do i = 1, npanel
    call matmult( SS(:,:,i), lamdafinal, lamdareset, 4, 4, 1 )    !!!
    w((4*i-3):(4*i)) = lamdareset                              !!!
end do
w((n_w-6):(n_w-5)) = z       !!!
w(n_w-4) = nu0
w(n_w-3) = theta0
w(n_w-2) = nus
w(n_w-1) = t1
w(n_w) = t2
end subroutine init_w
```

Code to execute the Multiple Shooting Method

```
subroutine MS_function(                          &
                        n_w, w,                  &  ! Given
                        F, iflag                 )  ! Yielded
!
! projectile parameters
```

```
!
use Physical
implicit none
integer n_w, iflag, writeme
double precision w(n_w), F(n_w)
integer             n_MS, npanels(3), MS_iter
double precision    delt(3), angle_l, angle_r, t_l, t_r, rt(2), theta0, theta,&
                    r0(2), v0(2), tf, L, D, q, qmin, t1, t2
common/GlobalStuff/ delt,    angle_l, angle_r, t_l, t_r, rt   , theta0, theta,&
                    r0   , v0   , tf, L, D, q, qmin, t1, t2, writeme, n_MS,&
                    npanels, MS_iter
integer             npanel, num(3), TotalNum, i, ii, j, jj, last_start, n_RK, jjj,&
                    Hflag1, Hflag2, Hflag3, Sflag, n

double precision :: interval(3), I4(4,4), lamda_input(4,n_MS), z(2), t, r(2), v(2),&
                    lamdafinal(4), state(8), dt_des, h, height, tkeep, lamdar1,&
                    lamdar2, lamdav1, rtest(2), lamdav2, thetakeep, htest, gamma,&
                    tnew, statenew(8), angle, mag2D, dot2D, downrange, dpsi_dx(4,2),&
                    dphi_dx(4), dpsi_dxBYz(4), dr(2), nu0, Ham, vdot(2), H_t1plus,&
                    H_t2plus, H_t1minus, H_t2minus, S_t1, dS_dx(4), nu_s1, v0mag,&
                    nu_s2, dSdotdr(2), dSdotdv(2), vmag2, da_a_dx(2,4), da_dr(2,2),&
                    da_dv(2,2), prod1(2), prod2(2), rmag, dN_dx_t(4,2), nu_s(2),&
                    prod3(4), hdotmin, alphastar, alphaapprox, alphasol, rhat(2),&
                    Sdot, vmag, hdot, rho, drhodh, sos, dsosdh, d2rhodh2, qtest, v2,&
                    rmag2
integer alphacheck, kk
```

```
double precision alpha_plot, alpha_plot2, Hu, Huu
common/Checkout/ alpha_plot, alpha_plot2, Hu, Huu, alphacheck, kk
double precision wprevious(163), Fprevious(163)
common/Fcheck/ wprevious, Fprevious
logical interpolate
integer print_theta
common/Decisions/  interpolate, print_theta
external deqs_2D
300 format(20x, g20.10 )
alphacheck = 0
!
! Initial conditions
!
npanel = npanels(1) + npanels(2)! + npanels(3)
do i = 1, npanel
    lamda_input(:,i) = w((i*4-3):(i*4))
end do
lamda_input(1,:) = lamda_input(1,:)!/100.d0
lamda_input(2,:) = lamda_input(2,:)!/100.d0
z      = w(n_w-7:n_w-6)!/100.d0
nu0    = w(n_w-5)
theta0 = w(n_w-4)
nu_s1  = w(n_w-3)
t1     = w(n_w-2)*100.d0 !scaling factor for time
tf     = w(n_w)*100.d0
nu_s2  = w(n_w-1)
```

```
print_theta = 0
H_t1minus = 0.d0
H_t1plus  = 0.d0
S_t1      = 0.d0
interval(1) = t1 / dble(npanels(1))
interval(2) = (tf - t1)/dble(npanels(2))
!write(*,*) rt
n_RK = 8
call ident( I4, 4)
t = 0.d0
r0(1) = Re
r0(2) = 0.d0
v0(1) = Vmag0*dcos(theta0)
v0(2) = Vmag0*dsin(theta0)
v0mag = mag2D(v0)
r = r0
v = v0
state(1:2) = r
state(3:4) = v
state(5:8) = lamda_input(:,1)
dt_des   = .001d0
num(1)      = int(interval(1)/dt_des) + 1
num(2)      = int(interval(2)/dt_des) + 1
delt(1)     = interval(1)/num(1)
delt(2)     = interval(2)/num(2)
h         = dsqrt( r(1)**2 + r(2)**2 ) - Re
```

```
downrange     = 0.d0
rmag  = mag2D(r)
rmag2 = rmag*rmag
vmag  = mag2D(v)
vmag2 = vmag*vmag
rhat = r/rmag
hdot = dot2d(rhat, v)
h = rmag - Re
i = 1
htest = 10
alphastar = alpha_plot*180.d0/pi
alphasol = alpha_plot2*180.d0/pi
if (writeme == 1) then
    write(31, '(3e24.14)') t, alphastar, alphasol
endif
call get_a_2D(                &
                state, t,     &   ! Given
                vdot          )   ! Yielded
call shau62( h, rho, drhodh, sos, dsosdh, d2rhodh2)
Ham        = state(5)*state(3) + state(6)*state(4) + state(7)*vdot(1)&
             + state(8)*vdot(2)
hdotmin = (-.5d0*rho**2*vmag**3*S*Ca)/(m*((rho*mu/rmag2) - (.5d0*vmag2*drhodh)))
if (writeme == 1) then
    write( 22,'(22e24.14)')  t, h, downrange, theta, state, Ham, alpha_plot, Hu, Huu,&
                L, D, q, hdot, hdotmin
    write(32, '(9e24.14)') t, state(5), state(3), state(6), state(4), state(7),&
```

```
                vdot(1), state(8), vdot(2)
end if
jjj = 1 ! set the panel counter to 1
do kk = 1, 2 ! loop through the 2 sections
    if (kk == 2) then
        call get_a_2D(                &
                        state, t,     &   ! Given
                        vdot          )   ! Yielded
        H_t1plus = state(5)*state(3) + state(6)*state(4) + state(7)*vdot(1)&
                + state(8)*vdot(2)
    end if
    do ii = 1, npanels(kk)
        do jj = 1, num(kk)
            tnew  = t + delt(kk)
            call RungaKutta(                                &
                            deqs_2D, n_RK, state, t, tnew,  &  ! Given
                            statenew                        )  ! Yielded
            state    = statenew
            t     = tnew
            i     = i + 1
            rtest = state(1:2)
            v     = state(3:4)
            rmag  = mag2D(rtest)
            vmag  = mag2D(v)
            vmag2 = vmag*vmag
            rmag2 = rmag*rmag
```

```
h     = mag2D(rtest) - Re
rhat = rtest/rmag
hdot = dot2d(rhat, v)
alphastar = alpha_plot*180.d0/pi
alphasol = alpha_plot2*180.d0/pi
if (writeme == 1) then
    write(31, '(3e24.14)') t, alphastar, alphasol
endif
call get_a_2D(                &
               state, t,      &   ! Given
               vdot           )   ! Yielded
Ham = state(5)*state(3) + state(6)*state(4) + state(7)*vdot(1)&
    + state(8)*vdot(2)
if (kk == 1 .and. jj == num(1) .and. ii == npanels(1)) then
    H_t1minus = Ham
end if
if ( mod(i,1) == 0 .and. writeme == 1) then
    angle      = dacos(dot2D(r0, rtest)/(mag2D(r0)*mag2D(rtest)))
    downrange  = Re * angle
    call shau62( h, rho, drhodh, sos, dsosdh, d2rhodh2)
    hdotmin = (-.5d0*rho**2*vmag**3*S*Ca)/(m*((rho*mu/rmag2)&
            - (.5d0*vmag2*drhodh)))
    write( 22,'(22e24.14)')  t, h, downrange, theta, state, Ham,&
            alpha_plot, Hu, Huu, L, D, q, hdot, hdotmin
    write(32, '(9e24.14)') t, state(5), state(3), state(6), state(4),&
            state(7), vdot(1), state(8), vdot(2)
```

```
            end if
        end do
        if ( jjj < npanel ) then
            if (kk == 1 .and. ii == npanels(1) ) then
                rmag = mag2D(rtest)
                rhat = rtest/rmag
                v = state(3:4)
                rmag2 = rmag*rmag
                vmag = mag2D(v)
                vmag2 = vmag*vmag
                hdot = dot2d(rhat, v)
                call shau62( h, rho, drhodh, sos, dsosdh, d2rhodh2)
                v2 = vmag*vmag
                q = .5d0*rho*v2
                write(*,*) 't, q, qmin, rho, v^2 = ', t, q, qmin, rho, v2
!                write(*,*)
                S_t1 = qmin - q
                Sdot = -.5d0*v2*drhodh*hdot - rho*dot2d(v, vdot)
                dS_dx(1:2) = -.5d0*drhodh*v2*rhat
                dS_dx(3:4) = -rho*v
                call partials_full_2D(              &   !
                                      state, t,     &   ! Given
                                      da_a_dx       )   ! Yielded
                da_dr = da_a_dx(:,1:2)
                da_dv = da_a_dx(:,3:4)
                call matmult(v, da_dr, prod1, 1, 2, 2)
```

```
            dSdotdr      = -(.5d0*vmag2*hdot*d2rhodh2 &
                    + drhodh*dot2D(v, vdot))*rhat - rho*prod1 -&
                    .5d0*vmag2*drhodh*(v - hdot*rhat)/rmag
            call matmult(v, da_dv, prod2, 1, 2, 2)
            dSdotdv      = -rho*vdot - rho*prod2 - drhodh*hdot*v -&
                        .5d0*vmag2*drhodh*rhat
            dN_dx_t(1,1) = dS_dx(1)
            dN_dx_t(2,1) = dS_dx(2)
            dN_dx_t(3,1) = dS_dx(3)
            dN_dx_t(4,1) = dS_dx(4)
            dN_dx_t(1,2) = dSdotdr(1)
            dN_dx_t(2,2) = dSdotdr(2)
            dN_dx_t(3,2) = dSdotdv(1)
            dN_dx_t(4,2) = dSdotdv(2)
            nu_s(1) = nu_s1
            nu_s(2) = nu_s2
            call matmult(dN_dx_t, nu_s, prod3, 4, 2, 1)
            F(jjj*4-3:jjj*4) = state(5:8) - lamda_input(:,jjj+1) - prod3
            state(5:8) = lamda_input(:,jjj+1)
        else
            F(jjj*4-3:jjj*4) = state(5:8) - lamda_input(:,jjj+1)
            state(5:8) = lamda_input(:,jjj+1)
        end if
    end if
    jjj = jjj + 1
end do
```

```
end do
!write(*,*) rt
if ( writeme == 1) then
    angle         = dacos(dot2D(r0, rtest)/(mag2D(r0)*mag2D(rtest)))
    downrange     = Re * angle
!    call get_theta(          &
!                     state,  & ! Given
!                     theta   ) ! Yielded
!    call get_a_2D(                &
!                    state, t,      &   ! Given
!                    vdot           )   ! Yielded
    call shau62( h, rho, drhodh, sos, dsosdh, d2rhodh2)
    hdotmin = (-.5d0*rho**2*vmag**3*S*Ca)/(m*((rho*mu/rmag2) - (.5d0*vmag2*drhodh)))
    Ham        = state(5)*state(3) + state(6)*state(4) + state(7)*vdot(1)&
                + state(8)*vdot(2)
    write( 22,'(22e24.14)')  t, h, downrange, theta, state, Ham, alpha_plot, Hu,&
            Huu, L, D, q, hdot, hdotmin
    write(32, '(9e24.14)') t, state(5), state(3), state(6), state(4), state(7),&
                        vdot(1), state(8), vdot(2)
end if
dpsi_dx      = 0.d0
dpsi_dx(1,1) = 1.d0
dpsi_dx(2,2) = 1.d0
dphi_dx(1:2) = 0.d0
dphi_dx(3:4) = -statenew(3:4)
call matmult( dpsi_dx, z, dpsi_dxBYz, 4, 2, 1 )
```

```
lamdafinal = dphi_dx + dpsi_dxBYz
dr = state(1:2) - rt
! Final stuff into F(w)
last_start = 4*npanel-3
F(last_start:last_start+1)   = dr / 100.d0
F(last_start+2:last_start+3) = (state(5:6) - lamdafinal(1:2))*100.d0
F(last_start+4:last_start+5) = state(7:8) - lamdafinal(3:4)
F(last_start+6)  = lamda_input(3,1) - nu0*dcos(theta0)
F(last_start+7)  = lamda_input(4,1) - nu0*dsin(theta0)
F(last_start+8)  = S_t1/qmin
F(last_start+9)  = (H_t1minus - H_t1plus)*100.d0
F(last_start+10) = Sdot/5.d06
F(last_start+11) = Ham*200.d0
write(*, '( "call #, t1, tf, Ham = ", i5, x, 2f12.5, e20.5)') MS_iter, t1, tf, Ham
MS_iter = MS_iter + 1
end subroutine MS_function
```

Code containing the state differential equations

```
subroutine deqs_2D(                    &
                   statecurrent, t,    &  ! Given
                   stateprime          )  ! Yielded
! Calculates the state derivatives.  Uses the get_a routine to find vdot.
use Physical
implicit none
```

```
double precision statecurrent(8), t, stateprime(8)
integer             n_MS, writeme, npanels(3)
double precision    delt(3), angle_l, angle_r, t_l, t_r, rt(2), theta0, theta,&
                    r0(2), v0(2), tf, L, D, q, t1, t2
common/GlobalStuff/ delt,    angle_l, angle_r, t_l, t_r, rt   , theta0, theta,&
                    r0   , v0   , tf, L, D, q, t1, t2, writeme, n_MS, npanels
double precision :: x(4), v(2), r(2), rhat(2), lamda_r(2), lamda_v(2), vdot(2),&
                    da_a_dx(2,4), da_dr(2,2), da_dr_trn(2,2), da_dv_trn(2,2),&
                    prod(2), da_dv(2,2), lamdadot_r(2), lamdadot_v(2), h, vmag,&
                    nhat(2), dSdot_dalpha, da_dalpha(2), mu_const, dSdotdv(2),&
                    dSdotdr(2), hdot, rho, drhodh, d2rhodh2, asp, daspdh, rmag,&
                    vmag2, xhat(2), theta_keep, mag2D, dot2D
logical interpolate
integer print_theta
common/Decisions/  interpolate, print_theta
300 format(20x, g20.10 )
x = statecurrent(1:4)
v = statecurrent(3:4)
lamda_r = statecurrent(5:6)
lamda_v = statecurrent(7:8)
call get_a_2D(                  &   !
               statecurrent, t, &   ! Given
               vdot             )   ! Yielded
theta_keep = theta
call partials_full_2D(                  &
                      statecurrent, t,  &   ! Given
```

```
                        da_a_dx           )   ! Yielded
da_dr = da_a_dx(1:2, 1:2)
da_dv = da_a_dx(1:2, 3:4)
call trn( da_dr, da_dr_trn, 2, 2 )
call matmult( -da_dr_trn, lamda_v, lamdadot_r, 2, 2, 1 )
call trn( da_dv, da_dv_trn, 2, 2 )
call matmult( da_dv_trn, lamda_v, prod, 2, 2, 1 )
lamdadot_v = - lamda_r - prod
stateprime = (/ v , vdot, lamdadot_r, lamdadot_v /)
end subroutine deqs_2D


Code to calculate the accelerations (vdot)


subroutine get_a_2D(                  &
                        xcurrent, t,   &   ! Given
                        vdot)! anvec, nnvec     )   ! Yielded
use Physical
implicit none
! Takes the current position(r) and velocity (v) and angle of attack
! and calculates the derivative (v and vdot)
!
double precision :: xcurrent(8), t, vdot(2)
integer             n_MS, writeme, npanels(3)
double precision    delt(3), angle_l, angle_r, t_l, t_r, rt(2), theta0, theta,&
                    r0(2), v0(2), tf, L, D, q, t1, t2
```

```
common/GlobalStuff/ delt,    angle_l, angle_r, t_l, t_r, rt   , theta0, theta,&
                    r0   , v0   , tf, L, D, q, t1, t2, writeme, n_MS, npanels
double precision :: r(2), v(2), interval, slope, xhat(2), vhat(2), uhat(2),&
                    sinalpha, cosalpha, alpha, h, rho, drhodh, sos, dsosdh, A,&
                    Ahat(2), Avec(2), N, Nhat(2), Nvec(2), g(2), dot2D, mag2D,&
                    h_send, anvec(2), nnvec(2), d2rhodh2, qdot, vmag, rhat(2),&
                    hdot, vmag2, rmag, F_alpha, rmag2, qdot1, qdot2, F_alpha1,&
                    F_alpha2, qdotdiff1, qdotdiff2, qdotg1, qdotg2, qdota1,&
                    qdota2, Nalpha, Aforce, gamma, F11, F12, F21, F22, F1, F2,&
                    theta2, alpha_Sdot, alpha_star
integer alphacheck, kk
double precision alpha_plot, alpha_plot2, Hu, Huu
common/Checkout/ alpha_plot, alpha_plot2, Hu, Huu, alphacheck, kk
logical interpolate
integer print_theta
common/Decisions/  interpolate, print_theta
r     = xcurrent(1:2)
v     = xcurrent(3:4)
vmag  = mag2D(v)
vmag2 = vmag**2
rmag  = mag2d(r)
rmag2 = rmag**2
rhat  = r / rmag
hdot  = dot2D(v, rhat)
alpha_plot2 = 0.d0
gamma   = datan2(v(2), v(1))
```

```
!write(*,*) 'r, v = ', r, v
! Calculate the control (interpolate == 1; get_theta <> 1)
call get_theta(              &
                xcurrent,    & ! Given
                theta        ) ! Yielded
xhat = (/ dcos(theta), dsin(theta) /)
if (print_theta == 1) then
    write(*,*) '4th print', theta
endif
vhat = v/vmag
!write(*,*) 'vhat, mag(v) = ', vhat, mag2D(v)
uhat = (/ vhat(2), -vhat(1) /)
!write(*,*) 'uhat = ', uhat
sinalpha = dot2D(uhat, xhat)
cosalpha = dot2D(vhat, xhat)
alpha = datan2(sinalpha, cosalpha)
alpha_plot = alpha
!write(*,*) alpha
!if (alphacheck == 1) then
!    write(27, '(20e24.14)') t, vhat, uhat, sinalpha, cosalpha, gamma, theta
!    write(27, '(20e24.14)') g, mu, rmag, N, A, Avec, Nvec
!end if
!if(on_constraint) then
!    write(28, '(20e24.14)') alpha, alpha_plot, alpha - alpha_plot
!send if
!write(*,*) 'alpha = ', alpha
```

```
! alphadeg = alpha*180/pi
!
! Calculate height
!
h = mag2D(r) - Re
h_send = h
!
! Calculate density
!
call shau62( h_send, rho, drhodh, sos, dsosdh, d2rhodh2)
if ( h_send < 0.d0 ) then
    rho = 1.303243789981300d09 - h_send*1.196173013681500d08 +     &
        .5d0*h_send*h_send*4.758658604189700d06                    &
        + h_send*h_send*h_send*3.859588831998408d05/3.d0
    drhodh = -1.196173013681500d08 + h_send*4.758658604189700d06 + &
        .5d0*h_send*h_send*3.859588831998408d05
    d2rhodh2 = 4.758658604189700d06 + h_send*3.859588831998408d05
endif
!
! Calculate axial force
!
A = .5d0 * rho * vmag2 * S * Ca   ! magnitude of axial force
Ahat = -xhat
Avec = A * Ahat
anvec = Avec/m
!
```

```
! Calculate normal force vector
!
N = .5d0 * rho * vmag2 * S * Cna * alpha  ! magnitude of normal force
!write(*,*) 'rho, v, S, Cna, alpha = ', rho, v, S, Cna, alpha
!write(*,*) 'N = ', N
Nhat = (/ -Ahat(2), Ahat(1) /)
Nvec = N * Nhat
nnvec = Nvec/m
!write(*,*) 'Nvec = ', Nvec
Aforce = .5d0*rho*vmag2*S*Ca
Nalpha = .5d0*rho*vmag2*S*Cna
!
! gravitational force
!
g = -mu * r / (mag2D(r)**3)
!
! total acceleration
vdot = (Avec + Nvec)/m + g
!write(*,*) 'vdot in get_a_2D', vdot
if (print_theta == 1) then
    write(*,*) '5th print', theta
end if
if (writeme == 1) then
    L = N*dcos(alpha) - A*dsin(alpha)
    D = A*dcos(alpha) + N*dsin(alpha)
end if
```

```
q = .5d0*rho*vmag2
!if (alphacheck == 1) then
!      write(27, '(20e24.14)') t, q, F_alpha2, rho, h_send, vdot, drhodh, hdot,  &
!                              Avec, Nvec, g, N, A, alpha
!      write(27, '(20e24.14)') t, rho, vmag2, S, Cna, alpha
!      write(27, '(20e24.14)') g, mu, rmag, N, A, Avec, Nvec

!       write(27, '(20e24.14)') F11, F21, F12, F22, F1, F2, F_alpha2 - F_alpha
!end if
end subroutine get_a_2D


Code to generate the A and B matrices used to for the costate estimates

subroutine getAandB_2D(               &
                       state, t,      & ! Given
                       A, B           ) ! Yielded
! generates the A and B matricies
implicit none
double precision state(8), t, A(4,4), B(4)
integer             n_MS, writeme, npanels(3)
double precision    delt(3), angle_l, angle_r, t_l, t_r, rt(2), theta0, theta,&
                    r0(2), v0(2), tf, L, D, q, t1, t2
common/GlobalStuff/ delt,    angle_l, angle_r, t_l, t_r, rt   , theta0, theta,&
                    r0   , v0   , tf, L, D, q, t1, t2, writeme, n_MS, npanels
double precision :: I2(2,2), x(8), da_a_dx(2,4), da_dr(2,2), da_dv(2,2),&
```

```
                    da_dtheta(2), Z2(2,2)
logical interpolate
common/Decisions/  interpolate
call ident(I2, 2)
x = state
call partials_full_2D(           &   ! da_a_dx = partials_full_2D(x, t)
                      x, t,      &   ! Given
                      da_a_dx    )   ! Yielded
300 format( 20x, g20.10 )
da_dr = da_a_dx(:, 1:2)
da_dv = da_a_dx(:, 3:4)
!write(*,*) 'da_a_dx'
!write(*,300) da_a_dx
A(1:2,1:2) = 0.d0
A(1:2,3:4) = -da_dr
A(3:4,1:2) = -I2
A(3:4,3:4) = -da_dv
call getpartialda_dtheta_2D(            &   !    da_dtheta = getpartialda_dtheta_2D(x, t
                            x, t,       &   ! Given
                            da_dtheta )     ! Yielded
!write(*,*) 'da_dtheta'
!write(*,300) da_dtheta
B = (/ 0.d0, 0.d0, da_dtheta(1:2) /)
end
```

```
Code to calculate the angle of attack


subroutine get_theta(           &
                      state,    & ! Given
                      control   ) ! Yielded
use Physical
implicit none
double precision state(8), control
integer             n_MS, writeme, npanels(3), iter
integer, parameter :: itmax = 100
double precision    delt(3), angle_l, angle_r, t_l, t_r, rt(2), theta0, theta,&
                    r0(2), v0(2), tf, L, D, q, t1, t2
common/GlobalStuff/ delt,    angle_l, angle_r, t_l, t_r, rt   , theta0, theta,&
                    r0   , v0   , tf, L, D, q, t1, t2, writeme, n_MS, npanels
double precision :: v(2), lamda_v(2), gamma, epsilon, check, diff, vhat(2), uhat(2),&
                    dot1, dot2, alpha, alpha1, alpha2, F_alpha, F_prime, alpha_new,&
                    mag2D, dot2D, r(2), x(4), vmag, vmag2, rmag, rmag2, h, rhat(2),&
                    hdot, rho, drho_dh, sos, dsosdh, d2rhodh2, Aforce, Nalpha,&
                    hdotmin, term, alpha_con, alpha_new_con, F_alpha2, F_prime2,&
                    check2, alpha_old
integer alphacheck, kk
double precision alpha_plot, alpha_plot2, Hu, Huu
common/Checkout/ alpha_plot, alpha_plot2, Hu, Huu, alphacheck, kk
!
! Initialize variables
!
```

```
x       = state(1:4)
r       = state(1:2)
v       = state(3:4)
vmag    = mag2D(v)
vmag2   = vmag**2
rmag    = mag2D(r)
rmag2   = rmag**2
h       = rmag - Re
lamda_v = state(7:8)
gamma   = datan2(v(2), v(1))
rhat    = r / mag2D(r)
hdot = dot2D(v, rhat)
epsilon = 1.d-08
check   = 1.d-16
check2  = 1.d-12
diff    = 1000.d0
vhat    = v / mag2D(v)
uhat    = (/ vhat(2), -vhat(1) /)
dot1    = dot2D(lamda_v, vhat)
dot2    = dot2D(lamda_v, uhat)
!
! Solve for unconstrained angle of attack
!
if ( dabs(dot2) < epsilon ) then
    alpha = 0
else
```

```
    !
    ! Two roots of the approximate solution of alpha (small angle approx.)
    !
    alpha1 = (-dot1*(2*Cna - Ca) - dsqrt(((2*Cna - Ca)*dot1)**2  &
        + 4*dot2**2*Cna*(Cna - Ca)))/(2*dot2*Cna)
    alpha2 = (-dot1*(2*Cna - Ca) + dsqrt(((2*Cna - Ca)*dot1)**2  &
        + 4*dot2**2*Cna*(Cna - Ca)))/(2*dot2*Cna)
    !
    ! Choose the root with the smallest magnitude to initialize the Newton's Method
    !
    if ( dabs(alpha1) < dabs(alpha2)) then
        alpha = alpha1
    else
        alpha = alpha2
    end if
end if
    !
    ! Newton's method iteration to solve for angle of attack
    !
iter = 0
do while ( diff > check .and. iter < itmax)
    iter = iter + 1
    F_alpha = dot1*(Ca - Cna)*dsin(alpha) - dot2*Cna*alpha*dsin(alpha) &
            - dot1*Cna*alpha*dcos(alpha) + dot2*(Cna - Ca)*dcos(alpha)
    F_prime = dot1*(Ca - Cna)*dcos(alpha) - dot2*Cna*dsin(alpha)       &
            - dot2*Cna*alpha*dcos(alpha) - dot1*Cna*dcos(alpha)        &
```

```
            + dot1*Cna*alpha*dsin(alpha) - dot2*(Cna - Ca)*dsin(alpha)
    alpha_new = alpha - F_alpha/F_prime
    diff      = dabs(alpha - alpha_new)
    alpha_old = alpha
    alpha     = alpha_new
end do
if ( alpha > 2*pi ) then
    alpha = 2*pi - alpha
end if
!write(*,*) 'alpha*'
!write(*,*) alpha
alpha_plot2 = 0.d0
!
! solve for the constrained angle of attack if on the second panel
!
if (kk == 2) then
    diff = 1000.d0
    call shau62(h, rho, drho_dh, sos, dsosdh, d2rhodh2)
    if ( h < 0.d0 ) then
        rho = 1.303243789981300d09 - h*1.196173013681500d08 +     &
                .5d0*h*h*4.758658604189700d06 + h*h*h*3.859588831998408d05/3.d0
        drho_dh = -1.196173013681500d08 + h*4.758658604189700d06 + &
                .5d0*h*h*3.859588831998408d05
        d2rhodh2 = 4.758658604189700d06 + h*3.859588831998408d05
    endif
!
```

```
! Solve for angle of attack
!
    Aforce = .5d0*rho*vmag2*S*Ca
    Nalpha = .5d0*rho*vmag2*S*Cna
    hdotmin = (-.5d0*rho**2*vmag**3*S*Ca)/(m*((rho*mu/rmag2) - (.5d0*vmag2*drho_dh)))
!write(*,*) 'hdot min = ', hdotmin
    term = (-(m/(rho*vmag))*((rho*mu/rmag2) - (.5d0*vmag2*drho_dh))*hdot&
            - Aforce)/Nalpha
!write(*,*) 'term = ', term
    if (term < 0.d0) then
        alpha_con = 0.d0
    else
    alpha_con = dsqrt(term)
    end if
    alpha_plot2 = alpha_con
!    write(*,*) 'alpha_approx'
!    write(*,*) alpha_con
!write(*,*) 'alpha guess = ', alpha*180.d0/pi
!
! Newton's method iteration to solve for angle of attack
!
    iter = 0
    if (abs(alpha_con) > epsilon) then
        do while ( diff > check2 .and. iter < itmax)
        iter = iter + 1
            F_alpha2   = (rho*mu/rmag2 - .5d0*vmag2*drho_dh)*hdot +     &
```

```
                        rho*vmag*(Nalpha*alpha_con*dsin(alpha_con) +    &
                        Aforce*dcos(alpha_con))/m  ! m included because
!   forces, not accelerations, are used in F_alpha and F_prime
            F_prime2   = (rho*vmag*(Nalpha*(alpha_con*dcos(alpha_con) + &
                        dsin(alpha_con)) - Aforce*dsin(alpha_con)))/m
            alpha_new_con = alpha_con - F_alpha2/F_prime2
            diff      = dabs(alpha_con - alpha_new_con)
            alpha_con     = alpha_new_con
        !write(*,*) F_alpha, F_prime, alpha, diff, alpha
        !stop
        end do
    end if
!write(*,*) 'Solution = ', alpha*180/pi
    if (alpha_con > 2.d0*pi ) then
        alpha_con = alpha_con - 2.d0*pi
    end if
!write(*,*) 'alpha_Sdot'
!write(*,*) alpha_con
!
! Determine which alpha to use
!
    if (abs(alpha) > abs(alpha_con)) then
        alpha = dsign(alpha_con, alpha)
    end if
end if
!write(*,*) 'alpha at end'
```

```
!write(*,*) alpha
!
! Set the control (theta) based on the angle of attack
!
control = gamma - alpha
alpha_plot = alpha
Hu  = F_alpha
Huu = F_prime
end subroutine get_theta


Code to calculate the partial derivative of the acceleration with&
respect to the control

subroutine getpartialda_dtheta_2D(         &
                                  x, t,    &  ! Given
                                  da_dtheta )  ! Yielded
! finds the partial of acceleration wrt theta
use Physical
implicit none
double precision data, x(8), t, da_theta(2)
integer             n_MS, writeme, npanels(3)
double precision    delt(3), angle_l, angle_r, t_l, t_r, rt(2), theta0, theta,&
                    r0(2), v0(2), tf, L, D, q, t1, t2
common/GlobalStuff/ delt,    angle_l, angle_r, t_l, t_r, rt   , theta0, theta,&
                    r0   , v0   , tf, L, D, q, t1, t2, writeme, n_MS, npanels
```

```
double precision :: r(2), v(2), slope, xhat(2), int, vhat(2), uhat(2), sinalpha,&
                    cosalpha, h, dt_des, tkeep, lamdar1, lamdar2, lamdav1, lamdav2,&
                    thetakeep, htest, dot2D, alpha, A, N, dN_da, dA_da, dL_da, dD_da,&
                    rho, drhodh, sos, dsosdh, da_dtheta(2), mag2D, h_send, d2rhodh2,&
                    h_use
r = x(1:2)
v = x(3:4)
! Calculate the control
int = t_r - t_l
slope = (angle_r - angle_l) / int
theta = angle_l + slope*(t - t_l)
xhat = (/ dcos(theta), dsin(theta) /)
vhat = v / mag2D(v)
uhat = (/ vhat(2), -vhat(1) /)
sinalpha = dot2D(uhat, xhat)
cosalpha = dot2D(vhat, xhat)
alpha = datan2(sinalpha, cosalpha)
! alphadeg = 180*alpha/pi
!
! Calculate height
!
h = mag2D(r) - Re
h_send = h
!
! Calculate density
```

```
!
call shau62( h_send , rho, drhodh, sos, dsosdh, d2rhodh2)
if ( h_send < 0.d0 ) then
    rho = 1.303243789981300d09 - h_send*1.196173013681500d08 +     &
             .5d0*h_send*h_send*4.758658604189700d06                &
             + h_send*h_send*h_send*3.859588831998408d05/3.d0

    drhodh = -1.196173013681500d08 + h_send*4.758658604189700d06 + &
             .5d0*h_send*h_send*3.859588831998408d05

    d2rhodh2 = 4.758658604189700d06 + h_send*3.859588831998408d05
endif
!
! Calculate axial force
!
A = .5d0 * rho * dot2D(v, v) * S * Ca    ! magnitude of axial force
!
! Calculate normal force vector
!
N = .5d0 * rho * dot2D(v, v) * S * Cna * alpha  ! magnitude of normal force
! Calculate the partials of the normal force, axial force, lift and drag wrt alpha
dN_da = .5d0 * rho * dot2D(v, v) * S * Cna
dA_da = 0.d0
dL_da = (dN_da - A) * cosalpha - (N + dA_da) * sinalpha
dD_da = (dN_da - A) * sinalpha + (N + dA_da) * cosalpha
! calculate the partial of acceleration wrt theta
```

```
da_dtheta = -(dL_da * uhat - dD_da * vhat) / m
end subroutine getpartialda_dtheta_2D


Code to calculate the partial derivatives of the acceleration
with respect to the state

subroutine partials_full_2D(          &
                            x, t,     & ! Given
                            da_a_dx   ) ! Yielded
! function takes the current state and returns partials of the total
! acceleration
implicit none
double precision x(8), t, da_a_dx(2,4)
integer             n_MS, writeme, npanels(3)
double precision    delt(3), angle_l, angle_r, t_l, t_r, rt(2), theta0, theta,&
                    r0(2), v0(2), tf, L, D, q, t1, t2
common/GlobalStuff/ delt,    angle_l, angle_r, t_l, t_r, rt   , theta0, theta,&
                    r0   , v0   , tf, L, D, q, t1, t2, writeme, n_MS, npanels
integer i, n
double precision :: del, s, xkeep(8), xpass(8), a_f(2), a_b(2)
s = .00001
xkeep = x
n = 4        ! Need to define
do i = 1, n
    xpass = xkeep
```

```
    del = max(1.d0, dabs(xpass(i))) * s    ! Need the max function
    xpass(i) = xkeep(i) + del
    call get_a_2D(               &
                   xpass, t,     &  ! Given
                   a_f           )  ! Yielded
    xpass(i) = xkeep(i) - del
    call get_a_2D(               &
                   xpass, t,     &  ! Given
                   a_b           )  ! Yielded
    da_a_dx(:,i) = (a_f - a_b) / (2 * del)
end do
end subroutine partials_full_2D

Time function to prevent time on constraint from being negative

subroutine tfunc(        &
                      w, c,  & ! Given
                      time   ) ! Yielded
implicit none
double precision w, c, a, b, b2, time, f1, cw
a = .5d0
f1 = dsqrt(1.d0 + c**2) - 1.d0;
cw = c*w
b  = a*c / f1
b2 = b**2
time = ( cw + dsqrt( cw**2 + 2.d0*b2*f1 ) ) / 2.d0
```

```
end subroutine tfunc
```

Code with the Runga-Kutta integration routine is identical to the listing&
in Appendix B.

Subroutine with the atmospheric model is the same as in Appendix B.

File with the physical parameters of the problem

```
module Physical
!
!  Module to define the physical parameters of the simulation
!
implicit none
double precision, parameter :: &
    Re  = 6371.d0,        &   ! Radius of the Earth (km)
    mu  = 398600.5d0,     &   ! Gravitational Constant (km^3/s^2)
    S_m2  = 0.005352d0,   &   ! Reference area of the projectile (m^2)
    S   = 0.005352d0,     &   ! Reference area of the projectile (m^2)
    m   = 12.d0,          &   ! Mass of the projectile (kg)
    Ca  = 0.15d0,         &   ! Axial Coefficient of the projectile
    vmag0 = 1.26d0,       &   ! km/s
    Cna = 5.d0,           &   ! Normal-force curve-slope &
```

```
                                coefficient of the projectile (per radian)
    pi = 3.1415926535897930d0  ! pi (of course)
double precision, parameter :: deg2rad = 0.0174532925199433294d0, &
                               rad2deg = 57.2957795130823230d0
end module Physical
```

angles.dat input file

```
10    10    10
       66.8079354465
       68.4947622801
       70.4974176100
       72.7945112350
       75.3715129528
       78.2150540073
       81.3118650764
       84.6434442619
       88.1896270537
       91.9244090442
       95.8175840342
       99.8310199756
      103.9262434122
      108.0621428610
      112.1993133974
      116.2977679975
      120.3254546561
```

```
      124.2549619242
      128.0661023676
      131.7416083441


flight.dat input file

      30.0
       0.0
       1.26
  1000000.0


decision.dat input file

.true.
```