MAXIMUM-RANGE TRAJECTORIES FOR AN UNPOWERED REUSABLE LAUNCH VEHICLE

A Thesis presented to the Faculty of the Mechanical
and Aerospace Engineering Department
at the University of Missouri

In Partial Fulfillment
of the Requirements for the Degree

Master of Science

---

by

JOSIAH A. BRYAN

Dr. Craig Kluever, Thesis Supervisor

DECEMBER 2011

The undersigned have examined the thesis entitled

MAXIMUM-RANGE TRAJECTORIES FOR AN UNPOWERED REUSABLE LAUNCH VEHICLE

presented by Josiah Bryan,

a candidate for the degree of Master of Science,

and hereby certify that, in their opinion, it is worthy of acceptance.

_____

Professor Craig Kluever

_____

Professor Douglas Smith

_____

Professor Carmen Chicone

ACKNOWLEDGEMENTS

I would like to thank Dr. Craig Kluever for his advisement and guidance, and for providing the inspiration for this research. It has been a privilege studying under him.

I would like to thank Dr. John B. Miles for informing me of the opportunity to be a student intern with the NASA-Missouri Space Grant Consortium and for assisting in the application process, and I thank the NASA-Missouri Space Grant Consortium for the opportunity to be a student intern and for supporting this research.

I would also like to thank my parents and family for their support and helpful observations and all the countless ways they have blessed me.

Most of all, I would like to thank my Lord and Savior Jesus Christ Who created the universe He helps me to study. Apart from Him I can do nothing.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii
LIST OF FIGURES v
LIST OF TABLES viii
NOMENCLATURE ix
ABSTRACT xiv
Chapter
1. INTRODUCTION 1
1.1. Reentry Background 1
1.2. Approaches to Range Maximization 5
1.3. Objectives 6
1.4. Selection of a Reusable Launch Vehicle 7
1.5. Implementation of Control System 8
1.6. Overview of the Paper 9
2. SYSTEM MODELS 12
2.1. Equations of Motion 12
2.2. Aerodynamic Model 16
2.3. Atmospheric Model 25
2.4. Controls Model 28
3. NUMERICAL OPTIMIZATION 34
3.1. Definition of Optimization Problem 34
3.2. Selection of the Number of Control Nodes 39
3.3. Comparison of Optimization Algorithms 53
4. QUASI-EQUILIBRIUM GLIDE (QEG) 59
4.1. Overview 59
4.2. Constant-Velocity Quasi-Equilibrium Glide (CVQEG) 63
4.3. Constant-Dynamic-Pressure Quasi-Equilibrium Glide (CDPQEG) 71
4.4. Control System for Maintaining Quasi-Equilibrium Glide Conditions 79
4.5. Control System to Follow the Bottom of the Drag Valley 103

5. OPTIMAL CONTROL THEORY 109
5.1. Two-Point Boundary Value Problem (2PBVP) 109
5.2. Approach to Solving the Two-Point Boundary Value Problem 118
6. CONCLUSION 120
6.1. Summary of Findings 120
6.2. Areas for Future Research 124
APPENDIX A: OPTIMAL-CONTROL DERIVATIONS 128
APPENDIX B: MATLAB CODE 130
B.1. Optimization Initialization and Execution (`RangeOpt.m`) 130
B.2. Max-*L/D* and QEG Control System Simulations (`RangeSim.m`) 142
B.3. Numerical Objective Function (`performance.m`) 147
B.4. Trajectory Setup and Integration (`traj.m`) 147
B.5. Equations of Motion Evaluation (`EOM.m`) 148
B.6. Nonlinear Constraints for Numerical Optimization (`nonlcon.m`) 150
B.7. Aerodynamic Model of X-34 (`AeroX34Piece.m`) 151
B.8. Angle of Attack for Max *L/D* (`AeroX34AlphaStar.m`) 156
B.9. Atmospheric Model (`atmos.m`) 157
B.10. QEG States and Drag Valley (`DragValleyNewton.m`) 160
B.11. Fast Linear Interpolation (`lininterp.m`) 166
B.12. Fast Bilinear Interpolation (`bilininterp.m`) 167
B.13. Two-Dimensional Numerical Optimization (`TwoDTest.m`) 169
B.14. Time-Trial/Validation of Aero Model (`plotAeroDataPiece.m`) 172
REFERENCES 178
VITA 181

LIST OF FIGURES

Figure Page

2.1. Lift-to-drag ratio of X-34 aerodynamic model……..……21

2.2. Angle of attack for maximum lift-to-drag ratio as a function of Mach number……25

2.3. Atmospheric density models as function of altitude……..27

2.4. Rate of change of atmospheric density models as function of altitude…….27

2.5. Speed of sound computed by piecewise thermal model as function of altitude…..28

2.6. Original control profile definition as function of time…….30

2.7. Modified control profile definition as function of energy height……..32

3.1. Side constraints on modified control profile……..36

3.2. Example of doubling the number of intervals between control nodes…….39

3.3. Optimal final range achieved for varying numbers of control nodes……....41

3.4. Percentage improvement of optimal ranges over max-*L/D* vs. computational time……42

3.5. Cumulative computational times to optimize with varied numbers of nodes……42

3.6. Velocity vs. energy height of optimal trajectories with varied numbers of nodes……47

3.7. Altitude vs. energy height of optimal trajectories with varied numbers of nodes…….47

3.8. Dynamic pressure vs. energy height of optimal trajectories with various nodes…..48

3.9. Mach number vs. energy height of optimal trajectories with various nodes…….48

3.10. Flight-path angle vs. energy height of optimal trajectories with various nodes……....50

3.11. Range flown vs. energy height of optimal trajectories with various nodes…….51

3.12. Control profiles vs. energy height of optimal trajectories with various nodes……51

3.13. Two-node control profile definition…….54

3.14. Range achieved for various values of each control in a two-node profile…….....54

3.15. Ranges achieved with PSO and gradient-based optimization with various nodes…….....57

4.1. CVQEG drag valley with contours of constant energy height ……..69

4.2. Flight-path angle along CVQEG drag valley with energy height contours…….....69

4.3. Angle of attack along CVQEG drag valley with energy height contours……..70

4.4. Dynamic pressure along CVQEG drag valley with energy height contours……..70

4.5. CDPQEG drag valley with contours of constant energy height ……..77

4.6. Flight-path angle along CDPQEG drag valley with energy height contours…….77

4.7. Angle of attack along CDPQEG drag valley with energy height contours…….....78

4.8. Dynamic pressure along CDPQEG drag valley with energy height contours…….....78

4.9. Velocity vs. energy height with QEG control systems ($K_\gamma = 0$)…….....90

4.10. Dynamic pressure vs. energy height with QEG control systems ($K_\gamma = 0$)……..90

4.11. Flight-path angle vs. energy height with QEG control systems ($K_\gamma = 0$)…….....91

4.12. Altitude vs. energy height with QEG control systems ($K_\gamma = 0$)……..91

4.13. Range vs. energy height with QEG control systems ($K_\gamma = 0$)…….92

4.14. Angle of attack vs. energy height with QEG control systems ($K_\gamma = 0$)……..92

4.15. Open-loop eigenvalue vs. energy height with QEG control systems ($K_\gamma = 0$)……...93

4.16. Flight-path-angle error vs. energy height with QEG control systems ($K_\gamma = 0$)……..93

4.17. Velocity vs. energy height with QEG control systems ($\lambda_{CL} = -0.05$ s^{-1})……..97

4.18. Dynamic pressure vs. energy height with QEG control systems ($\lambda_{CL} = -0.05$ s^{-1})……...97

4.19. Flight-path angle vs. energy height with QEG control systems ($\lambda_{CL} = -0.05$ s^{-1})…….98

4.20. Altitude vs. energy height with QEG control systems ($\lambda_{CL} = -0.05$ s^{-1})……98

4.21. Range vs. energy height with QEG control systems ($\lambda_{CL} = -0.05$ s^{-1})……..99

4.22. Angle of attack vs. energy height with QEG control systems ($\lambda_{CL} = -0.05$ s^{-1})……99

4.23. Flight-path-angle gain vs. energy height with QEG control systems ($\lambda_{CL} = -0.05$ s^{-1})..100

4.24. Flight-path-angle error vs. energy height with QEG control systems ($\lambda_{CL} = -0.05$ s^{-1}).100

4.25. Final range vs. closed-loop eigenvalue with QEG control systems……...102

4.26. Numerical optima converge on bottom of CVQEG drag valley……104

4.27. Numerical optima converge on bottom of CDPQEG drag valley……..105

4.28. Dynamic pressure of numerical optima compared to bottom of drag valleys…..106

4.29. Angle of attack of numerical optima compared to bottom of drag valleys…….106

4.30. Flight-path angle of numerical optimal compared to bottom of drag valleys……107

4.31. Ranges of numerically optimized trajectories with different initial conditions……107

LIST OF TABLES

Table Page

2.1. Comparison of times required to compute C_L and C_D using two approaches…….22

3.1. Initial and final states of numerical optimization trials……...45

4.1. Nominal parameters for constant drag-polar aerodynamic model of X-34 at Mach 0.6….....85

4.2. Initial and final conditions for trials using QEG control systems and max-*L/D*...……....86

4.3. Initial conditions of three numerically optimized trajectories converging on drag valleys..104

NOMENCLATURE

Acronyms

2PBVP	=	Two-Point Boundary Value Problem
A&L	=	Approach and Landing
CDPQEG	=	Constant-Dynamic-Pressure Quasi-Equilibrium Glide
CVQEG	=	Constant-Velocity Quasi-Equilibrium Glide
PSO	=	Particle-Swarm Optimization
QEG	=	Quasi-Equilibrium Glide
RLV	=	Reusable Launch Vehicle
TAEM	=	Terminal Area Energy Management

Symbols

α	=	angle of attack, deg
α^*	=	angle of attack corresponding to max *L/D*, deg
α_i	=	estimated angle of attack on *i*th iteration, deg
α_{pq}	=	*q*th-order regression coefficient of α^* as function of M between M_{α_p} and $M_{\alpha_{p+1}}$, deg^{-q}
α_{QEG}	=	reference angle of attack for QEG, deg
$\Delta\alpha$	=	perturbation in angle of attack, deg

$\Delta\gamma$	=	perturbation in flight-path angle, rad
$\delta\alpha$	=	deviation in angle of attack from α^*, deg
$\delta\gamma'$	=	linear approximation of $\partial\gamma/\partial e$, rad/ft
$\delta\hat{\alpha}$	=	perturbation in angle of attack from α_{QEG}, deg
$\delta\hat{\gamma}$	=	perturbation in flight-path angle from γ_{QEG}, rad
γ	=	flight-path angle, rad
γ_f	=	constrained final flight-path angle, rad
γ_g	=	adiabatic index of air
γ_i	=	estimated flight-path angle on *i*th iteration, rad
γ_{QEG}	=	reference flight-path angle for QEG, rad
$\{\lambda\}$	=	vector of costates
λ_{CL}	=	eigenvalue of closed-loop linearized control system, ft^{-1}
λ_{OL}	=	eigenvalue of open-loop linearized control system, ft^{-1}
λ_V	=	velocity costate, s
λ_γ	=	flight-path-angle costate, ft/rad
λ_h	=	altitude costate, dimensionless
λ_R	=	range costate, dimensionless
ρ	=	atmospheric density at current altitude, $slugs/ft^3$
ρ_0	=	atmospheric density at sea level, $slugs/ft^3$
ϕ	=	cost function of terminal states for optimal control
C_1	=	terminal state constraint for velocity, ft/s

C_2	=	terminal state constraint for flight-path angle, rad
C_3	=	terminal state constraint for altitude, ft
C_D	=	coefficient of drag
C_{D_0}	=	zero-lift drag coefficient for drag-polar model
$C_{D_{mn}}$	=	*m*th-order regression coefficient of C_D as function of α at Mach number M_n, $\deg^{-m}$
C_L	=	coefficient of lift
$C_{L_{mn}}$	=	*m*th-order regression coefficient of C_L as function of α at Mach number M_n, $\deg^{-m}$
C_{L_α}	=	"lift slope" coefficient for drag-polar model, $\deg^{-1}$
C_L^*	=	coefficient of lift corresponding to max *L/D*
D	=	drag force, lb_f
e	=	specific mechanical energy (energy height), ft
e_0	=	initial energy height, ft
$e_1, e_2, \ldots, e_N$	=	energy heights of control nodes, ft
e_f	=	final energy height, ft
F	=	objective function for numerical optimization, ft
$\{ f \}$	=	vector of state derivatives with respect to time
$\{ \hat{f} \}$	=	vector of state derivatives with respect to energy height
$\hat{f}_1$	=	derivative of velocity with respect to energy height, s^{-1}
$\hat{f}_2$	=	derivative of flight-path angle with respect to energy height, rad/ft
$\hat{f}_3$	=	derivative of altitude with respect to energy height,

		dimensionless
$\hat{f}_4$	=	derivative of range with respect to energy height, dimensionless
g	=	acceleration due to gravity at sea level, ft/s^2
H	=	Hamiltonian of optimal control problem
H_s	=	scale height of exponential atmosphere, ft
h	=	altitude, ft
h_f	=	constrained final altitude, ft
$[\mathbf{J}]$	=	Jacobian matrix
J	=	performance index of optimal control problem
$\tilde{J}$	=	augmented performance index of optimal control problem
K	=	induced-drag coefficient for drag model
K_γ	=	gain on error in flight-path angle, deg/rad
L	=	lift force, lb_f
$\hat{L}$	=	integrated cost function for optimal control
M	=	Mach number
M_{α_p}	=	Mach number at lower bound of *p*th piece of piecewise α^* function
$M_{\alpha_{p+1}}$	=	Mach number at upper bound of *p*th piece of piecewise α^* function
M_g	=	molar mass of air, kg/mol
M_n	=	Mach number of *n*th set of aerodynamic data
m	=	vehicle mass, slugs

N_i	=	number of control nodes in trial number i
$\bar{q}$	=	dynamic pressure, lb_f/ft^2
R	=	horizontal range flown, ft
R_g	=	molar gas constant, J/(mol-K)
S	=	wing reference area, ft^2
T	=	final time for optimal control
t	=	time elapsed from start of TAEM, s
$t_1, t_2, \ldots, t_N$	=	times of control nodes, s
t_f	=	estimated final flight time, s
u	=	control input
V	=	velocity, ft/s
V_f	=	constrained final velocity, ft/s
$\{\mathbf{x}_i\}$	=	estimated state vector on *i*th iteration
x	=	vector of state values

MAXIMUM-RANGE TRAJECTORIES FOR AN UNPOWERED REUSABLE LAUNCH VEHICLE

Josiah A. Bryan

Dr. Craig Kluever, Thesis Supervisor

ABSTRACT

A software package has been developed that numerically maximizes the range of an unpowered reusable launch vehicle (RLV) during the Terminal Area Energy Management (TAEM) phase of reentry into Earth's atmosphere by adjusting the angle-of-attack control profile at preselected energy heights along its trajectory. The software computes the optimal trajectory in terms of angle-of-attack deviations from a maximum lift-to-drag trajectory, which is the traditional trajectory used to maximize range of an unpowered aerial vehicle. In order to test the optimization software, an aerodynamic model of the X-34 launch vehicle was developed to calculate lift and drag coefficients for a given angle of attack and Mach number. Consideration of different numbers of control nodes is made, primarily with gradient-based optimization, though particle-swarm optimization is briefly tested. The merits of alternative control laws, such as constant-velocity or constant-dynamic-pressure quasi-equilibrium glide (QEG) algorithms, have also been investigated in an attempt to find a control law that does not require the inherent computational costs associated with numerical optimization. A two-point boundary-value problem is set up using optimal control theory to describe the optimization problem with simplified aerodynamic and atmospheric models.

CHAPTER 1: INTRODUCTION

1.1. Reentry Background

Recent years have seen a significant amount of interest in the development of a reliable and less expensive means of reaching space. The primary means of entering space throughout the world, apart from the Space Shuttle, has been to use expendable launch vehicles, which cannot be reused [1]. Launch vehicles are by nature very expensive, so it is highly desirable to develop a Reusable Launch Vehicle (RLV) such that its fixed cost pays for numerous launches, making the cost of each flight much less expensive. The Space Shuttle was partially reusable (it required a new external tank and refurbishment of the solid rocket boosters and orbiter following each flight) and was used in 135 missions from April 12, 1981, to July 21, 2011, given a fleet of five vehicles (Challenger, Columbia, Discovery, Atlantis, Endeavour) [2]. This type of reusability reduces cost immensely from the same number of missions with expendable launch vehicles. The cost, however, could be reduced even more by developing a fully reusable launch vehicle.

The 2nd Generation RLV Program, as a part of NASA's Integrated Space Transportation Plan, budgeted $4.5 billion from 2001 to 2005 for the purpose of developing "a commercially competitive, privately owned and operated RLV that serves both commercial and NASA human space flight and other government needs," emphasizing increased safety and reduced cost of reaching space [1]. In particular, the goals of the 2nd Generation RLV Program were to improve the safety of RLVs, defined in terms of the risk of crew loss, to less than 1-in-10,000 missions, and "to decrease the cost

by a factor of 10 – to approximately $1,000 per pound of payload launched to Low Earth Orbit (LEO)" [3]. Compare this safety goal to the estimated safety of the Shuttle at the time, which was considered to be about 1-in-500 missions in 2002 [4], noting that this was before the Columbia disaster in 2003.

It has been argued that advanced guidance and control systems are an important part of improving the safety of RLVs, allowing the vehicle to return under unexpected conditions for which it otherwise could not, and reducing the cost of using such vehicles [4]. Landing-site targeting systems, for instance, were considered for use in both the X-33 and X-34 RLVs, and in the X-34 the system was intended to find an appropriate and reachable site for an abort landing in the event that propulsion was lost [4]. Such systems contrast that of the Shuttle, in which ground-based programs are responsible for calculating abort possibilities, and large amounts of time are required to calculate safe abort trajectories before the vehicle is launched. Changes to the mission requirements may require adjustments in the control gains or guidance modes [4].

Among the situations that might be safely escaped using advanced guidance and control systems are aborts during ascent, vehicle dispersions greater than what was expected (deviating severely from nominal conditions), vehicle mismodeling, aerosurface failures, or other reductions in performance that could have unknown causes [5]. The goal has been proposed [5] of returning a vehicle safely whenever it is physically possible—i.e., an onboard control system can meet any situation that does not render the vehicle uncontrollable (e.g., an explosion could damage the control surfaces or control system). It has also been argued that 42.5 % of the failures of Russian, Japanese, European, and U.S. expendable launch vehicles from 1990 to 2003 occurred in failure

modes that would have been addressed by advanced guidance and control technologies, had the same failures occurred in RLVs [5].

A critical component of any RLV reentry or abort guidance system would be maximizing the flyable range of the RLV so that it could reach the greatest number of landing sites or, viewed another way, so that it could have the greatest probability of being able to reach a landing site. If the RLV reentered with severely off-nominal conditions, it is desired that the vehicle still have the greatest ability to reach a landing site as possible. The RLV would typically be unpowered in reentry, having spent its fuel in ascending, adjusting its orbit, and deorbiting. During an abort the RLV might also be unpowered, especially if the reason for aborting involves an engine malfunction. As an abort could occur at any point in the ascent, some abort scenarios may look very similar to off-nominal reentry scenarios. Furthermore, some of the approaches used for range maximization during reentry can be applied to range maximization at low altitudes as well. Therefore, a significant problem in RLV guidance is to maximize the range of an unpowered RLV during reentry into Earth's atmosphere.

The Terminal Area Energy Management (TAEM) flight phase of the Space Shuttle begins upon termination of the atmospheric entry phase (velocity of 1500 ft/s and altitude of ~70,000 ft) and continues to the Approach and Landing (A&L) interface (altitude of 10,000 ft) [6-7]. The purpose of TAEM is to achieve the velocity, altitude, and runway alignment necessary for the vehicle to enter the A&L phase. Depending on the results of the entry, though, the initial velocity and altitude of TAEM may vary. Moreover, the TAEM guidance system for the Shuttle was required to accommodate extreme winds and turbulence, up to three standard deviations of variation in initial

states, and be able to use the entire available range of the orbiter [6]. The TAEM guidance system for the Shuttle took the approach of conserving energy, given that the pilot can dissipate surplus energy by maneuvering if need be. An unpowered vehicle cannot generate additional energy if it comes up short. The TAEM guidance system attempted to track an energy profile as a function of range to go. Various NASA reports exist that describe the Space Shuttle TAEM guidance system [7-9]. An important part of the Shuttle TAEM phase includes using a Heading Alignment Cylinder to direct the orbiter toward the runway [6].

Some research has been conducted in developing advanced guidance and control concepts for A&L. A neural network has been employed to store optimal trajectories over a range of expected flight conditions in A&L, from which the trajectory to be flown can be reshaped to improve on range flown [10]. An algorithm was also developed for the X-34 to select a trajectory with a constant-dynamic-pressure steep glide slope for A&L [11]. Another algorithm finds a trajectory for A&L by iteratively seeking to satisfy a final-flare flight-path-angle constraint [12].

Attention has also been given to developing advanced guidance systems for the TAEM phase. One algorithm recalculates the TAEM reference trajectory iteratively, seeking to satisfy geometric constraints [13]. Other algorithms make use of fuzzy logic [14], nonlinear programming [15], and adaptive critic neural networks [16] to design TAEM trajectories. One approach applies to A&L and TAEM and allows the reference trajectory to be defined with only a few parameters. The trajectory can also be reshaped to accommodate aerosurface failures [17]. Another guidance method considers selecting a trajectory in the event of limited banking capabilities, as in the case of an aerosurface

failure [18]. Trajectory optimization that minimizes dynamic pressure for TAEM is also proposed [19].

Little attention, however, has been given to maximizing the range of an RLV in the TAEM phase, which is the focus of this study. This capability is critical for an RLV guidance system if the vehicle is to have the greatest potential for recovering from extreme initial conditions and safely gliding to a landing site. A notable investigation in this field is by De Ridder [20], using genetic algorithms to maximize range in TAEM. He also computes the so-called "drag valley" and considers constant-velocity QEG as an approach to maximizing range [20]. These approaches will be discussed at greater length throughout this study (see Sections 3.3, 4.1, and 4.2).

This study considers approaches to finding optimal trajectories in the vertical plane and does not consider the effects of banking or aligning the vehicle's heading with the runway. The Heading Alignment Cylinder introduces more complexities to the optimization problem and is neglected for now. Its effects can be introduced later if desired, once the basic optimization algorithm has been developed.

1.2. Approaches to Range Maximization

The traditional approach for maximizing the range of an unpowered aerial vehicle is to minimize the ratio of descent-rate to horizontal velocity—i.e., to minimize the absolute value of the flight-path angle (where positive indicates above the horizon, negative indicates below the horizon). This approach assumes the vehicle is flying in steady-state conditions, and the maximum range is accomplished by flying at the maximum lift-to-drag ratio (L/D). In reality, however, the atmospheric density increases

as the vehicle descends, and the vehicle's velocity is typically decreasing. If the dynamic pressure were kept constant throughout the descent (e.g., by balancing the increase in density with a decrease in the square of velocity), this approach would still work very well, resulting in a quasi-equilibrium glide. One motivation for pursuing this research is the investigation of such a quasi-equilibrium glide approach to range maximization.

Though in reality a vehicle may not have steady-state conditions due to the changing dynamic pressure, one means of approximating steady-state conditions is to adjust the angle of attack at every point along the trajectory (assuming an instantaneous pitch control and response) to maximize *L/D* of the vehicle. For any given Mach number there exists an angle of attack for which *L/D* is maximized, so if this relationship is known, a steady-state approximation can be made to find a near-optimal trajectory. The steady-state approximation works well enough at subsonic speeds, but there is no guarantee that it works for transonic or supersonic speeds. In this study the maximum lift-to-drag approach is used as an initial approximation of the optimal trajectory, and the optimal trajectory is then expressed as a set of deviations from the maximum *L/D* trajectory.

1.3. Objectives

The purpose of this research is to obtain the trajectory of an unpowered RLV that achieves a maximum horizontal range during the TAEM phase of reentry into Earth's atmosphere. The optimization algorithm should be flexible enough to accommodate a variety of initial conditions (altitude, velocity, and flight-path angle), and it should be able to consider constraints on the final altitude, velocity, and flight-path angle of the

vehicle. Moreover, it is desired that the algorithm be as computationally cost efficient as possible. Therefore, a control law that is easily evaluated at a given altitude and velocity is preferable to a holistic numerical optimization approach, provided that the simplified control law does not result in significantly lower maximum ranges. A secondary objective that is necessary in order to test the optimization algorithm is the development of a software model that computes the lift and drag coefficients of a specific launch vehicle for a given angle of attack and Mach number. In addition, a model is also needed to compute the angle of attack corresponding to the maximum lift-to-drag ratio of the vehicle at a given Mach number.

1.4. Selection of a Reusable Launch Vehicle

For the purpose of testing various optimization approaches on a realistic RLV model, the X-34 has been selected as the test model for this study. This selection is largely due to the availability of detailed aerodynamic data for the vehicle, as described in Section 2.2. NASA began developing the X-34 in 1996 as a test vehicle capable of launching into space with greater reliability and much lower cost than other vehicles, aiming to reduce launch costs from $10,000/lb to $1,000/lb [21]. The vehicle was designed to have a reusable Fastrac rocket motor and be capable of reaching Mach 8 and an altitude of 50 miles. The unfueled vehicle weighs 18,000 lb (the value used in this study, given that the vehicle glides upon reentry and does not need thrust). Three captive-carry test flights were conducted in 1999 before the program was reviewed in 2000 and it was determined that more funding would be necessary for the program to be successful. NASA did not approve the increased funding, and the program expired in

March 2001. In 2002 two existing vehicles and parts of a third were stored at Edwards Air Force Base. These vehicles were moved in November 2010 to Mojave Air and Spaceport where they were to be examined for possible further testing [22].

1.5. Implementation of Control System

Brief mention may be made here of the mechanism for implementing optimized trajectories in the guidance and control system onboard an RLV. If it is found that optimal TAEM trajectories are similar enough to one another over the expected initial conditions, there may be no need to place the optimization software directly onboard the RLV. An optimization-based algorithm may not even be fast enough or reliable enough for use onboard the vehicle, but the software could serve to produce a database of trajectories that may be stored onboard. A neural network could store pre-computed trajectories and produce interpolated solutions for intermediate initial conditions. This type of database of generalized scenarios may be a faster and more reliable mechanism for implementing nearly-optimal trajectories in the guidance system without a substantial loss of range compared to the true optimum. Moreover, such a database could also be used to test simpler control laws by observing similarities between the numerically optimized solution and a proposed alternate control algorithm, in order to determine which alternate algorithm most closely approximates the numerical solution. If an alternate control algorithm were found that were simpler (or, in particular, faster) yet still sufficiently accurate, such an algorithm could be implemented in an onboard guidance control system as a much faster and more reliable alternative to numerical optimization and, perhaps, as a faster, simpler, and more accurate alternative to a neural network.

Either a neural network or, especially, a simple control law could be implemented in such a way as to provide real-time control inputs for the vehicle, whereas numerical optimization may not be reliable enough or fast enough to do so. The attractiveness of having a simple control law for a real-time control system prompts the investigations into QEG and optimal control theory in Chapters 4 and 5.

1.6. Overview of the Paper

Chapter 2 describes the system models used for simulating glide trajectories in the TAEM phase, including the equations of motion of the vehicle (Section 2.1), the aerodynamics of an aerial vehicle and, in particular, the aerodynamics of the X-34 (Section 2.2), the atmospheric density and speed of sound as functions of altitude (Section 2.3), and the method of defining the control input for the vehicle.

Chapter 3 discusses the approaches used to numerically optimize the range of the vehicle. First the numerical optimization problem is defined (Section 3.1), then the effect of different numbers of control nodes on maximum range is considered (Section 3.2). The chapter closes with a brief consideration of alternate numerical optimization algorithms, with a focus on Particle-Swarm Optimization (PSO) (Section 3.3).

Chapter 4 addresses the possibility of using a QEG trajectory (holding flight-path angle and either velocity or dynamic pressure constant) to approximate maximum-range trajectories. The chapter begins with an overview of the rationale behind flying at maximum lift-to-drag ratios, as well as a broad introduction to QEG trajectories (Section 4.1). Two different QEG approaches are derived and discussed, along with numerical methods of solving for the states that satisfy the conditions of each approach. Each

approach can be used to generate a surface of drag values that satisfy QEG conditions over a range of velocities and altitudes. The resulting surface is called a drag valley because of its shape. The two QEG approaches examined here are called Constant-Velocity Quasi-Equilibrium Glide (CVQEG) (Section 4.2) and Constant-Dynamic-Pressure Quasi-Equilibrium Glide (CDPQEG) (Section 4.3) after the respective conditions that differentiate them. A control system for enforcing QEG conditions by tracking the QEG flight-path angle is discussed, along with preliminary results of implementing the control system (Section 4.4). The chapter ends by briefly considering a more sophisticated control system that would track the bottom of the QEG drag valley, noting the tendency of numerically optimized trajectories to converge on the bottom of the CDPQEG drag valley (Section 4.5).

Chapter 5 considers an alternate approach to maximizing range by defining the range-maximization problem as a Two-Point Boundary Value Problem (2PBVP) using optimal control theory (Section 5.1). To simplify the derivation, the aerodynamic and atmospheric models used in this chapter are less complex than those used for the numerical optimization. This chapter only sets up the problem, briefly mentioning that multiple shooting methods might be employed for its solution (Section 5.2).

Chapter 6 summarizes the findings of the previous chapters (Section 6.1), and it provides a discussion of areas for future research (Section 6.2). Appendix A provides more detailed derivations of some of the partial derivatives used in the optimal control problem of Chapter 5. Appendix B lists the code implemented in MATLAB to conduct the numerical range optimization, the simulation of glide trajectories (including

maximum lift-to-drag trajectories), the evaluation of the aerodynamic and atmospheric models, the solution of QEG conditions, and the simulation of QEG control systems.

CHAPTER 2: SYSTEM MODELS

2.1. Equations of Motion

The traditional dynamic model of motion of an unpowered aerial vehicle in the vertical plane was used, which includes four state variables: velocity V, flight-path angle γ, altitude h, and range R flown across the Earth. This section will discuss the vehicle dynamics as functions of lift and drag forces without explaining how to compute these forces. This computation is explained in Sections 2.2 and 2.3 with the aerodynamic and atmospheric models. The equations of motion of the vehicle are

$$\dot{V} = -\frac{D}{m} - g \sin\gamma \tag{2.1}$$

$$\dot{\gamma} = \frac{L}{mV} - \frac{g}{V} \cos\gamma \tag{2.2}$$

$$\dot{h} = V \sin\gamma \tag{2.3}$$

$$\dot{R} = V \cos\gamma \tag{2.4}$$

where V is the velocity, γ is the flight-path angle (positive above horizon, negative below horizon), h is the altitude above the runway, R is the range (horizontal distance flown), L and D are the lift and drag forces, respectively, g is the acceleration due to gravity (it is assumed to be constant at 32.174 ft/s^2 throughout flight), and m is the mass of the vehicle

(18,000 lbm or 8200 kg for the X-34 [21]. Note: m must be converted to slugs before use in the equations (~560 slugs)). Over-dots indicate derivatives with respect to time.

Equations (2.1)-(2.4) are given with time as the independent variable. The exact time of arrival at the A&L interface is not known, however, so there is no known time interval over which to integrate. Therefore, the independent variable selected for integration of the equations of motion is energy height, e, which is the total mechanical energy of the vehicle divided by its weight (where weight is constant because no fuel is consumed), defined as

$$e = \frac{V^2}{2g} + h \tag{2.5}$$

The final mechanical energy of the vehicle, e_f (sum of final kinetic and potential energies), can be computed from the velocity and altitude that designate the onset of the A&L phase. The initial mechanical energy, e_0, can be found in the same way from the velocity and altitude that designate the onset of the TAEM phase. Mechanical energy of the vehicle is always decreasing due to aerodynamic drag, so energy height is monotonic, making it a possible independent variable for integration instead of time. Defining the final energy height of the vehicle does not guarantee that the final velocity and altitude of the vehicle will be those of the A&L interface. In practice, though, the error between the final velocity and altitude of a simulated trajectory and the A&L interface remains relatively low. If it is critical that the final states be constrained, such constraints can be incorporated into the optimization problem quite easily.

Another benefit to using energy height as the variable of integration is that it constrains the final flight time to be whatever the time is at the final energy state. This removes one independent variable from the optimization problem. If time were the variable of integration, the final flight time would need to be a free variable in the optimization problem to allow for it to vary as needed in order for the final flight conditions to be met. By integrating over energy height instead and thereby reducing the number of independent variables in the optimization problem, fewer function evaluations are necessary to calculate the numerical gradient and Hessian, which typically means the optimization will take less computational time per iteration and fewer iterations to converge.

To convert the system from time derivatives to energy derivatives, the rate of change of energy height with respect to time, $\dot{e}$, is required. Differentiating Eq. (2.5) with respect to time and substituting (2.1) and (2.3) into it results in

$$\dot{e} = \frac{V}{g}\dot{V} + \dot{h} = -\frac{DV}{mg} \tag{2.6}$$

The equations of motion (Eqs. (2.1)-(2.4)) can now be rewritten as derivatives with respect to energy height by dividing each equation by Eq. (2.6), thereby canceling out the differential time terms in each time derivative. The new system of equations is

$$\frac{dV}{de} = \frac{\dot{V}}{\dot{e}} = \frac{-\frac{D}{m} - g\sin\gamma}{-\frac{DV}{mg}} = \frac{g}{V} + \frac{mg^2}{DV}\sin\gamma \tag{2.7}$$

$$\frac{d\gamma}{de} = \frac{\dot{\gamma}}{\dot{e}} = \frac{\frac{L}{mV} - \frac{g}{V}\cos\gamma}{-\frac{DV}{mg}} = -\frac{gL}{DV^2} + \frac{mg^2}{DV^2}\cos\gamma \tag{2.8}$$

$$\frac{dh}{de} = \frac{\dot{h}}{\dot{e}} = \frac{V\sin\gamma}{-\frac{DV}{mg}} = -\frac{mg}{D}\sin\gamma \tag{2.9}$$

$$\frac{dR}{de} = \frac{\dot{R}}{\dot{e}} = \frac{V\cos\gamma}{-\frac{DV}{mg}} = -\frac{mg}{D}\cos\gamma \tag{2.10}$$

One additional equation is integrated to find the time elapsed from the beginning of TAEM, *t*, at each energy step in order to plot the optimal trajectory with respect to time:

$$\frac{dt}{de} = \frac{1}{\dot{e}} = -\frac{mg}{DV} \tag{2.11}$$

To compute the states throughout a trajectory, Eqs. (2.7), (2.8), (2.10), and (2.11) are integrated from an initial energy height (computed from an initial velocity and altitude) to a final energy height (computed from velocity and altitude for A&L interface). Note that Eq. (2.9) is not needed because h can be determined from V and e by solving Eq. (2.5). The initial states should reflect the states at the beginning of the TAEM phase. Range and time are both zero at the beginning of the simulation, so the values of R and t at the end of the simulation indicate the range flown and time elapsed, respectively. The MATLAB `ode45` function is used to integrate the system using 4th-

order Runge-Kutta to compute the state values and 5th-order Runge-Kutta to check errors and determine an appropriate integration step size.

2.2. Aerodynamic Model

The lift and drag forces, L and D, respectively, that appeared in Eqs. (2.1)-(2.2) and (2.6)-(2.11) are

$$L = \bar{q}SC_L \tag{2.12}$$

$$D = \bar{q}SC_D \tag{2.13}$$

where $\bar{q}$ is the dynamic pressure, S is the wing reference area (357.5 ft^2 or 33.21 m^2 for the X-34 [23]), and C_L and C_D are the lift and drag coefficients, respectively. Dynamic pressure is defined as

$$\bar{q} = \frac{1}{2}\rho V^2 \tag{2.14}$$

where ρ is the atmospheric density at the current altitude (see Section 2.3 for details of atmospheric model).

The traditional approach to calculating the lift and drag coefficients of a vehicle is the "drag-polar" approach, in which the lift coefficient C_L is modeled as a linear function of angle of attack:

$$C_L = C_{L_0} + C_{L_\alpha}\alpha \tag{2.15}$$

where C_{L_0} is the lift coefficient at zero angle of attack, C_{L_α} is the "lift slope" coefficient, and α is the angle of attack. In the drag-polar approach, the drag coefficient C_D is modeled as a quadratic function of C_L with a zero first-order term:

$$C_D = C_{D_0} + KC_L{}^2 \tag{2.16}$$

where C_{D_0} is the zero-lift drag coefficient, and K is the induced-drag coefficient. The lift-to-drag ratio (L/D) is defined as the ratio of L (Eq. (2.12)) to D (Eq. (2.13)), which is equivalent to the ratio of C_L (Eq. (2.15)) to C_D (Eq. (2.16)):

$$\frac{L}{D} = \frac{\bar{q}SC_L}{\bar{q}SC_D} = \frac{C_L}{C_D} = \frac{C_L}{C_{D_0} + KC_L{}^2} \tag{2.17}$$

A common approach to maximizing the range of a glider is to maintain maximum L/D by changing C_L. One benefit of the drag-polar model is that C_L for maximum L/D is constant. To find maximum L/D, Eq. (2.17) is differentiated with respect to C_L, and the derivative is set equal to zero:

$$\frac{d}{dC_L}\left(\frac{L}{D}\right) = \frac{d}{dC_L}\left(\frac{C_L}{C_{D_0} + KC_L{}^2}\right) = \frac{\left((C_{D_0}+KC_L^2)-2KC_L^2\right)}{\left(C_{D_0}+KC_L^2\right)^2} = 0 \tag{2.18}$$

Equation (2.18) can be simplified to

$$C_{D_0} - KC_L^2 = 0 \tag{2.19}$$

Equation (2.19) can now be solved for the value of C_L corresponding to maximum *L/D*, denoted C_L^*, and is

$$C_L^* = \sqrt{\frac{C_{D_0}}{K}} \tag{2.20}$$

Note that the negative root of Eq. (2.19) is ignored because C_L must be positive in order to have a positive lift force. *L/D* would be negative and not a maximum if C_L were negative. The angle of attack corresponding to C_L^*, denoted α^*, is found by substituting C_L^* for C_L in Eq. (2.15):

$$\alpha^* = \frac{C_L^* - C_{L_0}}{C_{L_\alpha}} \tag{2.21}$$

The constant drag-polar approach allows much simpler derivation of controls laws and much faster evaluation of aerodynamic coefficients in general. In particular this approach allows faster evaluation of the angle of attack for maximum *L/D*. The drag-polar approach is most suitable for subsonic flight, over which the modeling parameters (C_{L_0}, C_{L_α}, C_{D_0}, and K) can be assumed to be constant. For transonic and supersonic

flight, however, the modeling parameters do not remain constant, so a more complex model must be developed that can account for these drag effects.

As mentioned in Section 1.4, the X-34 was chosen as the RLV in this study because of the detailed X-34 aerodynamic data available. These data were made available by Pamadi [23] and constitute numerous tables of aerodynamic coefficients across a wide range of Mach numbers (ranging from 0.3 up to 10), angles of attack, sideslip angles, and control-surface deflections. The data used to develop the model for this study are for zero-sideslip, normal cruise conditions and include C_L and C_D for thirteen Mach numbers from 0.3 to 2.5 and ten angles of attack from -6 deg to 21 deg. C_L and C_D are both functions of Mach number and angle of attack, where Mach number M is

$$M = \frac{V}{V_s} \tag{2.22}$$

where V_s is the speed of sound in air, which is a function of temperature (see Eq. (2.28)).

To minimize the computational cost of interpolating among the data, various attempts were made to find two-dimensional polynomial fits of C_L and C_D as functions of Mach number and angle of attack. These attempts largely resulted in severe discrepancies between the polynomial fit and the data (particularly when evaluating *L/D*, since the individual inaccuracies C_L and C_D compounded to produce a highly inaccurate ratio). Low-order polynomial fits neglected certain features of the data, but high-order fits rippled and folded unstably, adding features that did not exist in the data.

The solution settled upon, which results in a reasonably accurate model of the data, is to find a piecewise set of polynomial fits of C_L and C_D, each as a function of angle

of attack at a specific Mach number given in the data. Second-order polynomial fits are used to model C_L, and third-order polynomial fits are used to model C_D (except at Mach = 0.3, where a fourth-order fit was necessary), each as a function of angle of attack at each Mach number given in the data:

$$C_L = f(M,\alpha) = \begin{cases} C_{L_{01}} + C_{L_{11}}\alpha + C_{L_{21}}\alpha^2, & M = 0.3 \\ C_{L_{02}} + C_{L_{12}}\alpha + C_{L_{22}}\alpha^2, & M = 0.4 \\ C_{L_{03}} + C_{L_{13}}\alpha + C_{L_{23}}\alpha^2, & M = 0.6 \\ \vdots & \\ C_{L_{0n}} + C_{L_{1n}}\alpha + C_{L_{2n}}\alpha^2, & M = M_n \end{cases} \tag{2.23}$$

$$C_D = f(M,\alpha) = \begin{cases} C_{D_{01}} + C_{D_{11}}\alpha + C_{D_{21}}\alpha^2 + C_{D_{31}}\alpha^3 + C_{D_{41}}\alpha^4, & M = 0.3 \\ C_{D_{02}} + C_{D_{12}}\alpha + C_{D_{22}}\alpha^2 + C_{D_{32}}\alpha^3, & M = 0.4 \\ C_{D_{03}} + C_{D_{13}}\alpha + C_{D_{23}}\alpha^2 + C_{D_{33}}\alpha^3, & M = 0.6 \\ \vdots & \\ C_{D_{0n}} + C_{D_{1n}}\alpha + C_{D_{2n}}\alpha^2 + C_{D_{3n}}\alpha^3, & M = M_n \end{cases} \tag{2.24}$$

where M is Mach number, M_n is the nth Mach number for which aerodynamic data is known, $C_{L_{mn}}$ is the regression coefficient of degree m for C_L at Mach number M_n, and $C_{D_{mn}}$ is the regression coefficient of degree m for C_D at Mach number M_n. C_L and C_D at intermediate Mach numbers are evaluated by finding a weighted average of the polynomial fits at the two nearest Mach numbers given in the data.

Figure 2.1 displays the resulting *L/D* surface computed by the aerodynamic model as a function of M and α, as well as points that represent linear interpolations of the original data. It was found that the piecewise polynomial aerodynamic model was typically about ten times faster to evaluate than a traditional two-dimensional table

lookup using linear interpolation. It seems this is partly due to the ability to multithread calculations for different angles of attack and Mach numbers on different computer processor cores, as this factor-of-ten improvement was witnessed on a machine with an AMD Athlon II X4 630 quad-core processor running MATLAB R2010a in Windows 7, both of which have multithreading capabilities.

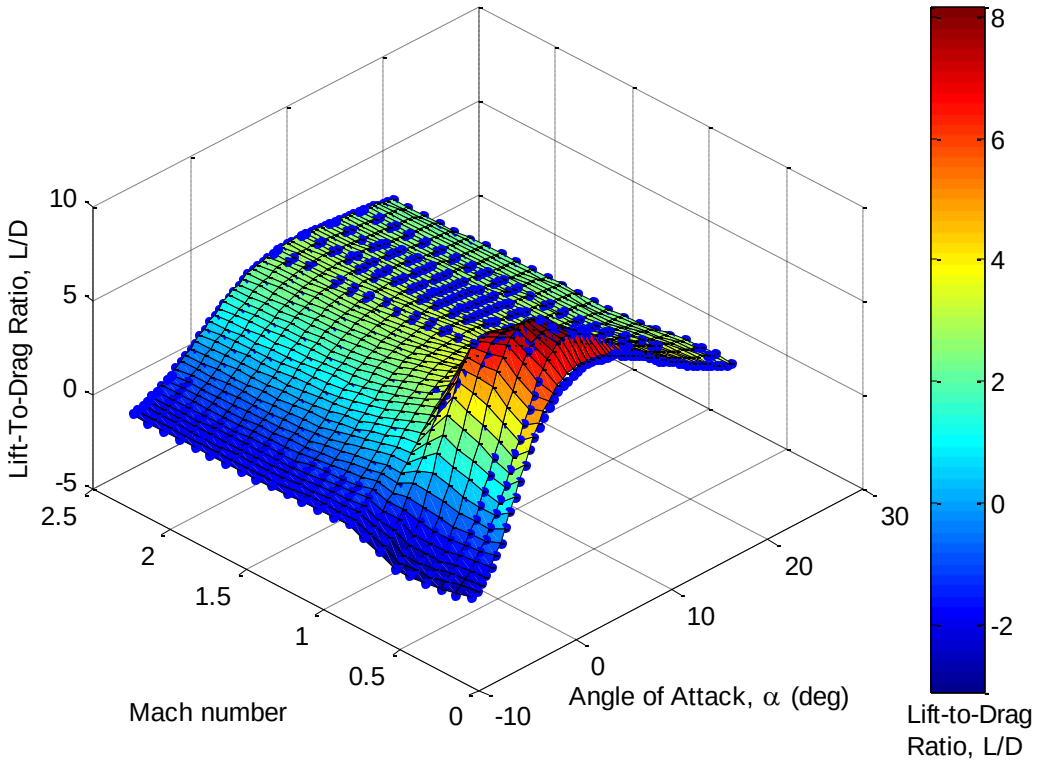

Figure 2.1. Lift-to-drag ratio (*L/D*) of X-34 modeled as function of Mach number and angle of attack. Shading indicates value of *L/D* computed by aerodynamic model. Dots are points found using bilinear interpolation of original data in order to illustrate accuracy of model.

Table 2.1 illustrates the difference in computational times between bilinear interpolation using the MATLAB `interp2` function and the piecewise polynomial model. Each trial consists of evaluating C_L and C_D at fifty-five angles of attack from -3 deg to 21 deg at a single Mach number. Some trials occur at a Mach number equal to a

value of M_n (i.e., a Mach number for which data are given), while others occur at an intermediate Mach number (i.e., one between two values of M_n), demonstrating the capacity of the two approaches for interpolation. Computational times are measured using the MATLAB `tic` and `toc` functions. Running the first trial takes about 100 times longer for each approach as MATLAB calls and loads the appropriate functions into memory. It is clear this is due to startup processes because the same trial can be repeated later with much faster performance. The times from the initial trial are, therefore, omitted from Table 2.1. It should also be noted that the structure of the MATLAB code for the piecewise aerodynamic model has been written as a sort of binary tree of if-then statements with the intent of minimizing the amount of logical checks required for a given evaluation of aerodynamic coefficients. The code for the aerodynamic model and its time trial can be found in Appendix B.

Table 2.1. Comparison of times required to compute C_L and C_D at fifty-five angles of attack at a single Mach number using two approaches: 1) bilinear interpolation with MATLAB `interp2` function, and 2) a piecewise set of polynomial fits, one for each known Mach number. Mach numbers are either equal to one value of M_n or intermediate to two values.

		Time Required to Compute C_L and C_D (s)	
Mach	**Relationship of M to M_n**	*Bilinear Interpolation*	*Piecewise Polynomial Model*
0.3	Equal	0.001682	0.000163
0.4	Equal	0.001540	0.000161
0.5	Intermediate	0.001478	0.000141
0.7	Intermediate	0.001589	0.000144
0.95	Equal	0.001472	0.000143
1.05	Equal	0.001439	0.000140
1.3	Intermediate	0.001473	0.000136
1.5	Intermediate	0.001452	0.000138
1.7	Intermediate	0.001759	0.000147
2.0	Equal	0.001506	0.000139
2.25	Intermediate	0.001509	0.000139

A limiter was written into the aerodynamic model such that any Mach number input outside the acceptable range of Mach number inputs (0.3 to 2.5) is automatically treated as the nearest acceptable Mach number (e.g., Mach 3.0 is treated as Mach 2.5). While this does not guarantee realistic aerodynamics, Mach numbers outside the acceptable range should not be necessary for optimal trajectories in the TAEM phase, and this measure does allow the objective function to remain evaluable if such a Mach number is given to the model, as sometimes occurs in suboptimal iterations of the optimization algorithm.

No such limiter was placed on angle of attack inside the aerodynamic model because boundary constraints are used to avoid unacceptable angles of attack in the optimal trajectory. Because the model consists of polynomial functions of angle of attack, the polynomials are continuous and differentiable even outside the acceptable range of inputs. For gradient-based optimization it can be more helpful for these polynomials to produce differentiable (though unrealistic) aerodynamic coefficients for inputs outside the acceptable range, in order to aid the gradient-based algorithm in returning to a more favorable region, than for the model to simply plateau (as with a limiter) and give no gradient information.

Because Eq. (2.21) is only valid for the constant drag-polar and not the piecewise polynomial model, a new means of finding α^* (the angle of attack that maximizes *L/D*) is desired. α^* is a function of Mach number, so a brute-force testing of the aerodynamic model output was conducted over the range of acceptable values of α and M. C_L and C_D were evaluated at Mach numbers from 0.3 to 2.5 at intervals of 0.01, and angles of attack from -3 deg to 21 deg at intervals of 0.001 deg. The angles of attack corresponding to the

maximum value of *L/D* at a given Mach number were fitted with a piecewise set of linear functions:

$$\alpha^* = f(M) = \begin{cases} \alpha_{10} + \alpha_{11}M, & M_{\alpha_1} \le M < M_{\alpha_2} \\ \alpha_{20} + \alpha_{21}M, & M_{\alpha_2} \le M < M_{\alpha_3} \\ \vdots & \\ \alpha_{p0} + \alpha_{p1}M, & M_{\alpha_p} \le M \le M_{\alpha_{p+1}} \end{cases} \tag{2.25}$$

where α_{pq} is the *q*th-order regression coefficient of the *p*th piece of the α^* function, and M_{α_p} is the lower bound on Mach number for which the *p*th piece is valid, and $M_{\alpha_{p+1}}$ is the upper bound. The resulting relationship between α^* and Mach number is displayed in Fig. 2.2. This relationship is used to simulate flying at maximum *L/D* in order to provide a reference trajectory for comparison with the optimal trajectory. Optimal trajectories are also expressed in terms of deviations from the max-*L/D* trajectory.

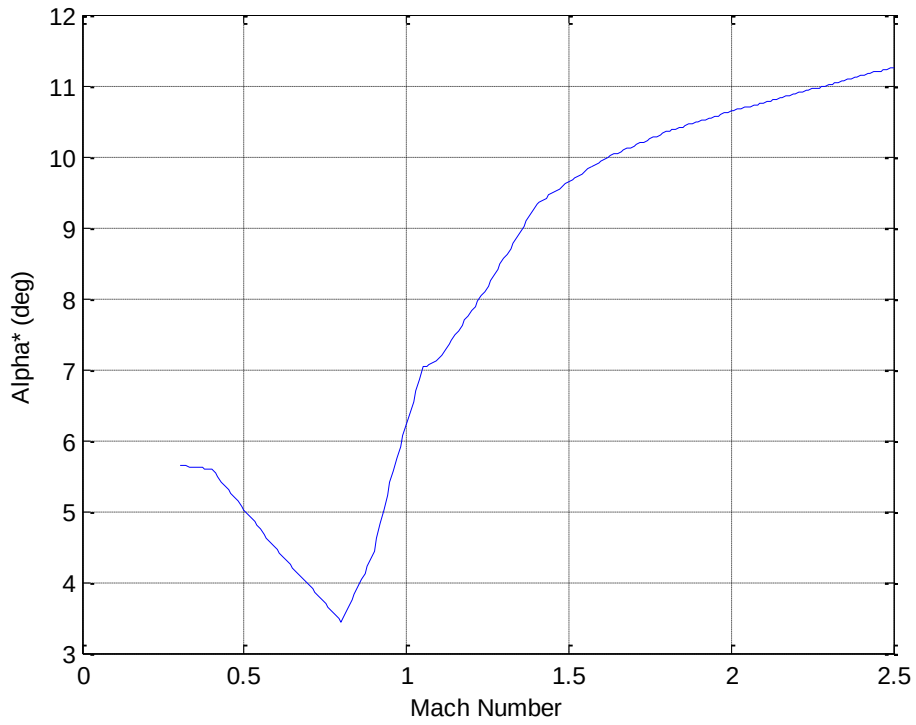

Figure 2.2. Angle of attack for maximum *L/D* as a function of Mach number, according to piecewise polynomial model of X-34 aerodynamics.

2.3. Atmospheric Model

One basic model of atmospheric density is the exponential model, in which density, ρ, decreases exponentially as altitude increases, according to the following relationship:

$$\rho = \rho_0 \exp\left(-\frac{h}{H_s}\right) \tag{2.26}$$

where h is altitude above sea level, ρ_0 is the atmospheric density at sea level (~0.002377 slug/ft^3) and H_s is the "scale height" (~30,500 ft), a constant that determines how slowly density decreases. The exponential model makes $d\rho/dh$ simple to calculate:

$$\frac{d\rho}{dh} = -\frac{\rho}{H_s} \tag{2.27}$$

The simple differentiability and continuity of the exponential model makes it useful for control-law derivations, and its speed of evaluation is good for reducing computational time.

For greater accuracy, however, other models have been considered, including a piecewise model that divides the atmosphere into linear temperature and isothermal regions (i.e., temperature decreases linearly from sea level up to 11,000 m, remains constant past that to 20,000 m, increases past that up to 32,000 m and even faster up to 47,000 m, and then remains constant above 47,000 m). The piecewise thermal model is more accurate than the simple exponential model, but it also requires more computational time to evaluate. Moreover, the piecewise model also computes speed of sound, V_s, in air by treating air as an ideal gas:

$$V_s = \sqrt{\frac{\gamma_g R_g T}{M_g}} \tag{2.28}$$

where γ_g is the adiabatic index of air (~1.400253219), R_g is the molar gas constant, T is the absolute temperature of the air, and M_g is the molar mass of air (where $R_g/M_g \sim 237$ $m^2/(K\text{-}s^2)$). The speed of sound is used for calculation of Mach number in Eq. (2.22). Figure 2.3 compares the densities computed by the exponential and piecewise thermal

models. Figure 2.4 compares the values of $d\rho/dh$ calculated by each model, and Fig. 2.5 shows the speed of sound calculated by the piecewise thermal model using Eq. (2.28).

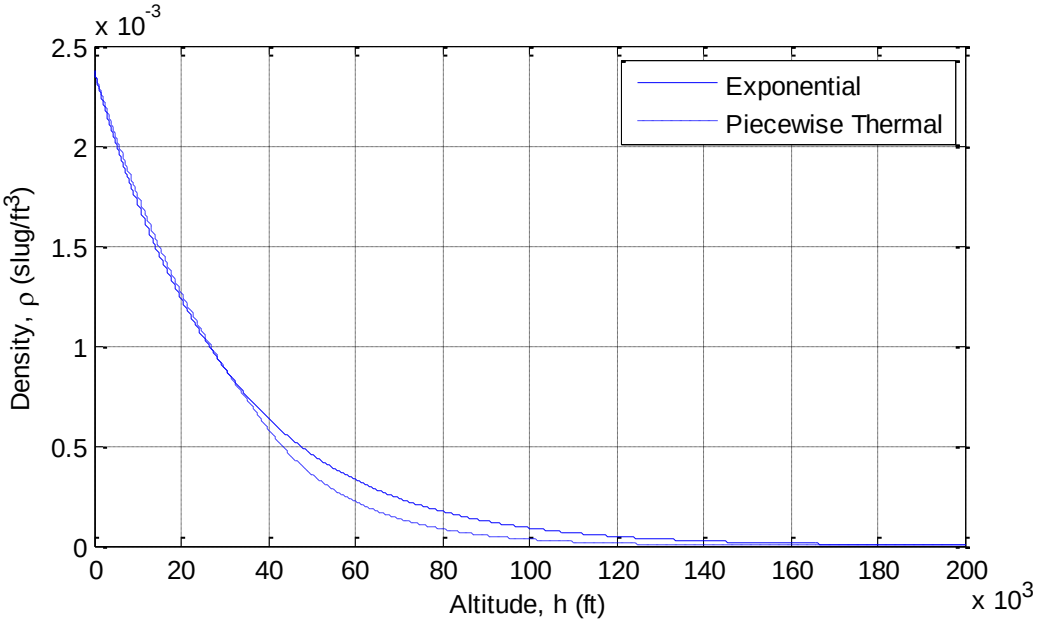

Figure 2.3. Density ρ computed by exponential and piecewise thermal models as functions of altitude h.

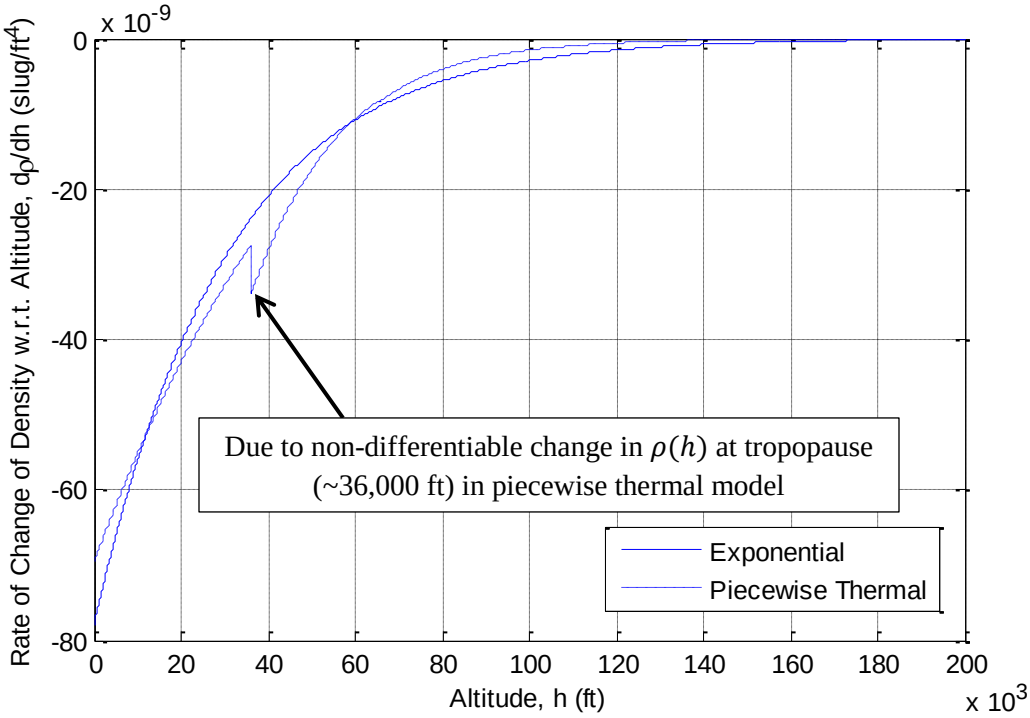

Figure 2.4. Rate of change of density, $d\rho/dh$, computed by exponential and piecewise thermal models as functions of altitude h. Discontinuity due to piecewise nature of model.

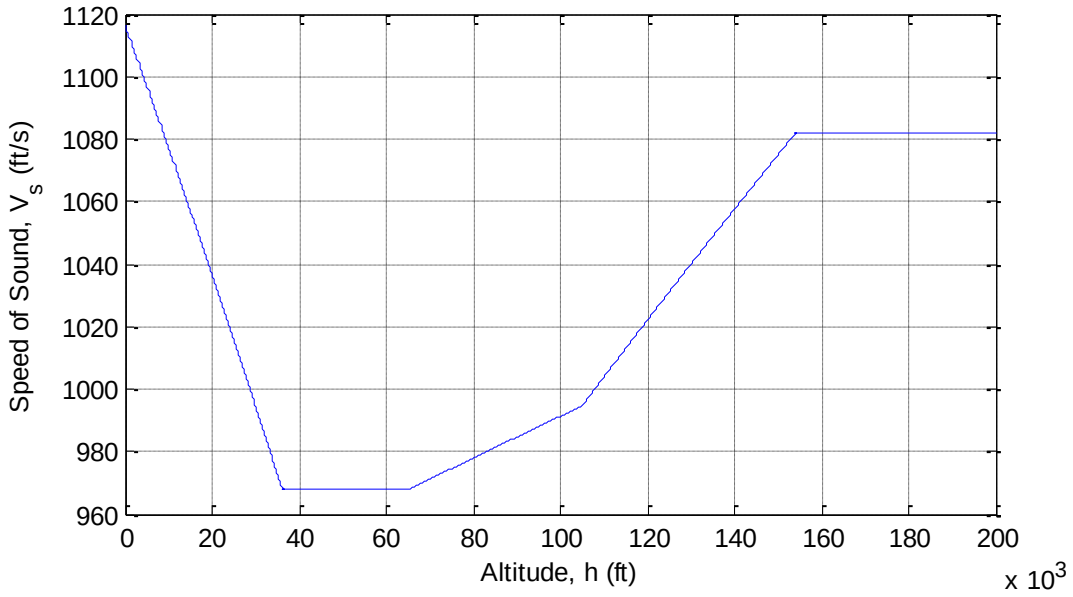

Figure 2.5. Speed of sound, V_s, computed by piecewise thermal model as function of altitude h, using Eq. (2.28).

The 1976 US Standard atmosphere offers even greater accuracy than the exponential or piecewise models, but it requires two-dimensional interpolation among a large number of actual data points, which takes even more time to calculate than either of the preceding approaches. The piecewise thermal model, therefore, is selected for its balance of accuracy and computational efficiency for the trajectory simulations in this study. MATLAB code for this model can be found in Appendix B.

2.4. Controls Model

Given that the purpose of this research is to develop a means of computing a control profile that maximizes the range covered by the vehicle, several possible approaches have been considered for defining the control profile. The only control input considered in this study is angle of attack and, to simplify the model, the dynamics of the

control system are neglected—i.e., adjustments in angle of attack are assumed to take place instantaneously. In reality, angle of attack cannot be adjusted instantaneously because it is controlled by ailerons that require time to move and because the vehicle requires time to respond to the new control input. The assumption of instantaneous control is adequate, however, for observing basic trends in optimal control profiles, even if those profiles are not continuous or differentiable. A non-differentiable control profile cannot be achieved in reality (where velocity, acceleration, and higher-order rates must all be continuous), but a differentiable curve might be fitted to approximate the non-differentiable profile with little effect on performance of the vehicle. It is assumed that a realistic control system could nearly replicate the control profiles found in this study by use of such an approximation. Hence, the dynamics of the control system are neglected for the purposes of this investigation.

Initially the control profile was defined as a set of control nodes equally spaced along the time axis, where each control node specified the angle of attack at a particular instant in time, as in Fig. 2.6. For any instant in time between two control nodes, α at that instant was determined by interpolating between the values of α at the two neighboring nodes. The time of the final control node was set as the estimated final flight time, t_f, which served as one of the independent variables of the optimization problem. If t ever exceeded t_f, α would be maintained at the same value of α that was defined for t_f.

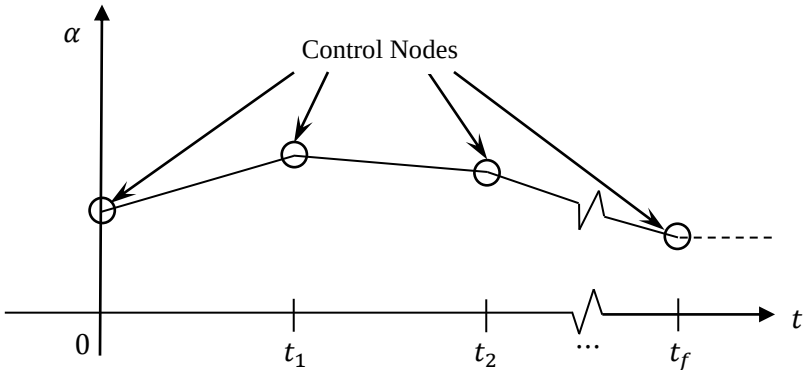

Figure 2.6. Original control profile definition, in which control nodes specify α at evenly-spaced times (t_1, t_2, …) along the trajectory. α at intermediate times is found by interpolation. t_f is the estimated final flight time. For $t > t_f$, α equals α at $t = t_f$.

As discussed in Section 1.3, the traditional approach to maximizing range of an unpowered aerial vehicle has been to fly a trajectory for which *L/D* at each instant is maximized. A max-*L/D* trajectory is accomplished by flying at α^* (angle of attack that corresponds to maximum *L/D*) at each instant, where α^* can be determined by the aerodynamics of the vehicle and the current Mach number (see constant drag-polar and Mach-dependent aerodynamic models in Section 2.2). For a constant drag-polar model, α^* is found by Eq. (2.21), and for the piecewise polynomial model of the X-34 aerodynamics, α^* is a function of Mach number as depicted in Fig. 2.2.

Optimal control profiles should closely resemble the maximum-*L/D* trajectory, which is derived as a simple approximation of the optimal trajectory. In order to highlight the differences between flying at max *L/D* and flying an optimal trajectory, the original control profile definition was modified so that each control node was defined as a deviation $\delta\alpha$ from α^* (the angle of attack corresponding to maximum *L/D*), where $\delta\alpha$ is a function of time:

$$\alpha(t) = \alpha^*(M) + \delta\alpha(t) \tag{2.29}$$

where α^* is either constant (Eq. (2.21)), in the case of a constant drag-polar model, or α^* is a function of Mach number as shown in Fig. 2.2, in the case of a piecewise polynomial model of the X-34 aerodynamics. If every control node throughout the trajectory were initialized to $\delta\alpha = 0$, then the vehicle would fly a max-*L/D* trajectory because at each instant in time, $\delta\alpha$ would be determined by interpolating between the nearest control nodes, which would both be zero, and the current angle of attack would be determined by adding the interpolated value of $\delta\alpha$ (which would be zero) to α^*.

Because the final flight time, t_f, is initially unknown, t_f was originally an independent variable in the optimization problem, as mentioned in Section 2.1 and also regarding the original control profile definition of Fig. 2.6. Because t_f was an independent variable, the positions of the control nodes along the time axis were modified as t_f was modified, adding unnecessary complexity to the optimization problem. Moreover, it was believed that the optimal angle of attack at a given instant was primarily affected by Mach number, so the control nodes were parameterized in terms of Mach number instead of time:

$$\alpha(M) = \alpha^*(M) + \delta\alpha(M) \tag{2.30}$$

Angle of attack deviations $\delta\alpha(M)$ at intermediate Mach numbers were determined by interpolating between the two nearest control nodes. But Mach number was not guaranteed to be monotonic, and because the flight dynamics are integrated with respect

to energy height, it was finally decided that the control nodes should be parameterized in terms of energy height, which is monotonic:

$$\alpha(e) = \alpha^*(M) + \delta\alpha(e) \tag{2.31}$$

Hence, $\delta\alpha(e)$ at a given energy height is determined by interpolating between the nearest control nodes, as in Fig. 2.7. Because the flight dynamics are integrated from one known energy height to another (i.e., e_0 to e_f), and because energy height decreases monotonically, this approach avoids the problems that might occur if Mach number would vary outside the range of Mach numbers over which the control profile is defined (this problem could be dealt with by using the $\delta\alpha$ at the nearest acceptable Mach number for Mach numbers outside the acceptable range, but this sort of control saturation reduces the effectiveness of the control profile in accomplishing the maximum-range objective).

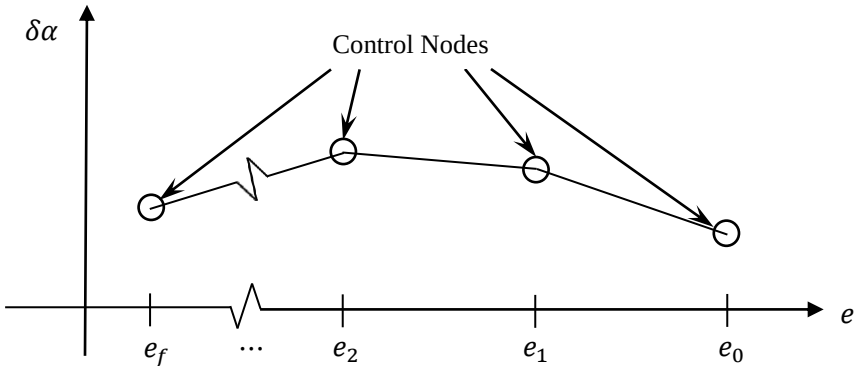

Figure 2.7. Modified control profile definition, in which control nodes specify $\delta\alpha$ at evenly-spaced energy heights (e_1, e_2, …) between initial energy height e_0 and final energy height e_f. $\delta\alpha$ at intermediate energy heights is found by interpolation. Note that $e_0 > e_f$ because energy decreases monotonically during the flight.

In addition to deciding how to parameterize the control nodes (angle of attack for a given energy height), it must also be decided how they should be spaced throughout the acceptable range of parameter values and how many are necessary. For the purposes of this study and for simplicity they are evenly spaced throughout the acceptable range of parameter values, as indicated in Fig. 2.7. It is possible, however, that certain regions of the control profile could benefit by having more nodes (e.g., the control profile may be more complex in the transonic region), while other regions may not need the same resolution because linear interpolation sufficiently resembles the optimal control profile in those regions. The number of control nodes is chosen experimentally by observing the tradeoff between maximum range achieved and computational time required (see Section 3.2).

It is possible that a higher-order interpolation method would increase the range achieved for a given amount of computational time by better approximating the optimal control profile for a given number of nodes. Higher-order interpolation methods have not been tested in this study because linear interpolation, which is a simpler approach, has behaved adequately thus far. Higher-order interpolation would increase the amount of time required to evaluate angle of attack at each integration step, but it could also result in fewer function evaluations in the optimization process by reducing the number of control nodes needed, which, in turn, could reduce the computational time required for the optimization algorithm to converge. It is noted here that a linear interpolation function (`lininterp`, see Appendix B) was developed for the purpose of computing $\alpha(e)$, given the values of the control nodes. This function performs much faster than the MATLAB `interp1` function by omitting the error-checking built into that function.

CHAPTER 3: NUMERICAL OPTIMIZATION

3.1. Definition of Optimization Problem

The optimization problem is defined as follows: Find the $\delta\alpha(e)$ profile that minimizes

$$F = -R(e_f) \tag{3.1}$$

subject to the equations of motion, Eqs. (2.7)-(2.10), where $R(e_f)$ is the horizontal range flown when the vehicle has reached the final energy height, e_f, and the $\delta\alpha(e)$ profile is defined according to Eq. (2.31) and Fig. 2.7 in Section 2.4. The value of $\delta\alpha$ at each control node, then, serves as one independent variable in the optimization problem. Equation (3.1) gives the negative of $R(e_f)$ for use with the MATLAB `fmincon` function because `fmincon` only minimizes objective functions, and the purpose of this optimization is to maximize $R(e_f)$. In order for the terminal states of the trajectory to coincide with the A&L interface, it is possible to set one or more terminal-state equality constraints:

$$C_1(e_f) = V(e_f) - V_f = 0 \tag{3.2}$$

$$C_2(e_f) = \gamma(e_f) - \gamma_f = 0 \tag{3.3}$$

$$C_3(e_f) = h(e_f) - h_f = 0 \tag{3.4}$$

where $V(e_f)$, $\gamma(e_f)$, and $h(e_f)$ are the velocity, flight-path angle, and altitude, respectively, at final energy height e_f, and V_f, γ_f, and h_f are the constrained final velocity, constrained final flight-path angle, and constrained final altitude, respectively (i.e., the values of the states at the A&L interface). Unless otherwise noted, no terminal states are constrained in this study (i.e., Eqs. (3.2)-(3.4) are neglected) because these constraints would increase the computational time to converge on an optimal solution, and the final energy height is already determined, which acts as a constraint on the combination of final velocity and altitude (see Eq. (2.5) for relationship of energy height to velocity and altitude).

Side constraints are placed on the angle-of-attack deviations $\delta\alpha(e)$ at each control node so that $\alpha(e)$ at each node in the optimal control profile may be inside the range of acceptable inputs (-6 deg to 21 deg) to the aerodynamic model:

$$-6 - \alpha^*(M) < \delta\alpha(e) < 21 - \alpha^*(M) \tag{3.5}$$

where $\alpha^*(M)$ is either a constant (Eq. (2.21)) or a piecewise linear function of Mach number (Fig. 2.2), depending on whether the aerodynamic model is a constant drag-polar or a Mach-dependent piecewise polynomial function. Equation (3.5) is intended to apply for all control nodes, which are evenly spaced from e_0 to e_f, and, ideally, for all values of

e from e_0 to e_f. When using the Mach-dependent aerodynamic model, however, values of e between nodes are not constrained, as will be shown.

Recall from Section 2.4 and Fig. 2.7 that $\delta\alpha(e)$ is evaluated at values of e between control nodes by linear interpolation between the two neighboring control nodes. Constraining $\delta\alpha$ at each control node does not necessarily ensure that the values of $\delta\alpha(e)$ between control nodes will also be within the side constraints. Recall from Eq. (2.22) that M is a function of V and V_s, where V_s is a function of altitude h. Hence, when using the piecewise aerodynamic model, $\alpha^*\left(M\left(V_s(h)\right)\right)$ changes throughout the trajectory as h changes. Therefore, as depicted in Fig. 3.1, it is possible for $\delta\alpha(e)$ to violate the side constraints in Eq. (3.5) when e is between two control nodes.

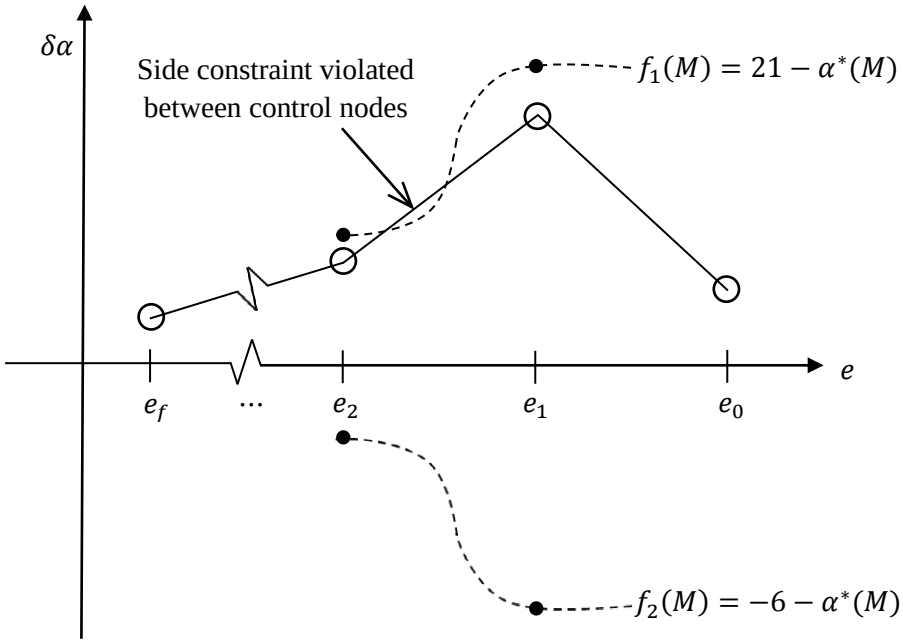

Figure 3.1. Control profile in which control nodes specify $\delta\alpha$ at evenly-spaced energy heights (e_1, e_2, …) between initial energy height e_0 and final energy height e_f. $\delta\alpha$ at intermediate energy heights is found by interpolation. Dashed lines indicate side constraints (Eq. (3.5)) that are enforced only at control nodes, which can result in violations at intermediate values of e. Black dots indicate locations at which constraints are applied.

Violations of the side constraints in Eq. (3.5) should not seriously affect the optimization process. When a side constraint is violated, the value of $\alpha(e)$ falls outside the range of acceptable inputs to the aerodynamic model. The aerodynamic model, which in this case is the Mach-dependent piecewise model (no violations occur when using a constant drag-polar because α^* is constant in Eq. (2.21)), uses polynomial fits of $C_L(\alpha)$ and $C_D(\alpha)$ at a given Mach number. Giving the model an input α that is outside the range of α for which the model was fitted causes the model to extrapolate, which reduces the accuracy of the computed aerodynamic coefficients. Given the relatively low order (highest order in Eqs. (2.23)-(2.24) is fourth) of polynomials used in the model, though, extrapolation should not result in completely unreasonable estimates of the aerodynamic coefficients at these values of α. These extrapolated values could, in fact, give a gradient-based optimization algorithm some guidance as to how to return to near-optimal values of α, as mentioned in Section 2.2 in the discussion of whether to use a limiter on α in the aerodynamic model. Such a limiter could conceal useful information from the optimization algorithm about the derivatives of the polynomial fits in the aerodynamic model. It should be noted that optimal trajectories are not expected to use values of α near the side constraints of Eq. (3.5), so the question of unrealistic aerodynamic coefficients should not affect optimal trajectories. As evidence of where the optimal trajectory will typically lie, consider that the max-*L/D* trajectory is a good approximation of the optimal trajectory, and $\alpha^*(M)$ tends to remain at low positive values of α as seen in Fig. 2.2.

At times, as will be noted later, the optimization problem is expanded by allowing the optimization algorithm to also adjust the initial flight-path angle, $\gamma(e_0)$, to maximize

the range achieved (i.e., to minimize Eq. (3.1)). If $\gamma(e_0)$ is allowed to vary, side constraints are also placed on this independent variable to ensure that

$$-\frac{\pi}{2} \leq \gamma(e_0) \leq +\frac{\pi}{2} \tag{3.6}$$

Outside this range of initial flight-path angles, the vehicle would begin its trajectory flying upside down and backtracking, i.e., with lift vector pointed toward the ground and range going toward negative infinity.

For initial investigations the MATLAB `fmincon` function was used to conduct the optimization. This function numerically approximates the gradient and Hessian and selects an algorithm to use based on the problem. Its algorithms use Newton's Method or other gradient-based approaches, including sequential quadratic programming. The `fmincon` function calls the objective function which, in turn, calls other functions to evaluate the range of the vehicle given the current control profile. Because `fmincon` can only minimize functions, the objective function was simply the negative of the final range achieved in the trajectory (Eq. (3.1)). Side constraints, linear constraints, and nonlinear constraints are all possible with `fmincon`, though the only constraints used for this study were side constraints on $\delta\alpha$ (and $\gamma(e_0)$ when it is a free variable). Constraints could be placed on final velocity, altitude, and flight-path angle (as in Eqs. (3.2)-(3.4)), but given that the trajectory is integrated from one energy height to another, which partially constrains the final altitude and velocity, no additional constraints were placed on the final state values.

3.2. Selection of the Number of Control Nodes

Numerous trials were conducted with different numbers of control nodes to determine the number of nodes that constitutes a good balance between accuracy and computational cost. Each consecutive trial doubled the number of intervals between control nodes from the previous trial, thereby halving the mesh size for the control profile, as seen in Fig. 3.2.

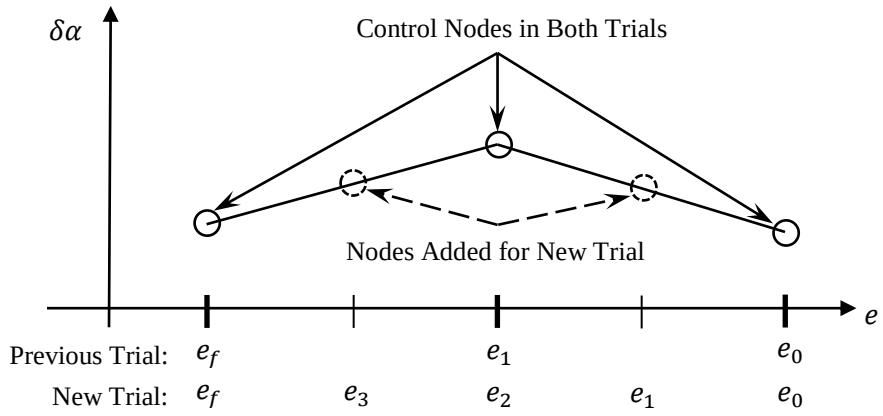

Figure 3.2. Example of doubling the number of intervals (halving the mesh size) between control nodes. Two nodes are added to the previous three, doubling the previous two intervals to make four total intervals.

Hence, if there is only one node in the first trial (i.e., a constant offset from the $\alpha^*(M)$ profile), then adding a node to make two nodes in the second trial, then doubling the number of intervals to make three nodes, the following relationship arises:

$$N_i = 1, \;\; i = 1 \tag{3.7}$$

$$N_i = 2, \;\; i = 2 \tag{3.8}$$

$$N_{i+1} = N_i + (N_i - 1) = 2N_i - 1, \;\; i > 2 \tag{3.9}$$

where N_i is the number of control nodes in the ith trial. This relationship produces the following sequence of numbers of control nodes: 1, 2, 3, 5, 9, 17, 33, 65, 129, etc.

In one series of tests—the "inherited-initials" approach—the optimal control profile $\delta\alpha(e)$ from a given trial is used as the initial guess for the following trial with the intent of improving consistency (i.e., always finding a better solution) and convergence in the new trial. The initial guess for each new trial is generated by maintaining all the same control nodes from the previous trial while adding a new node at the midpoint of each interval, as shown in Fig. 3.2. The initial value of each new node is set to the value of $\delta\alpha(e)$ at that energy height (i.e., linearly interpolated between the two neighboring control nodes), just as in the Fig. 3.2 illustration. In a different series of tests—the "zero-initials" approach—the initial guess for the control profile was set all to zeroes (i.e., $\delta\alpha(e_n) = 0$ for $n = 0,1, \ldots, N_i$ in the ith trial) at the start of each trial. Hence, the initial guess for the optimal trajectory for each trial comprised a max-*L/D* trajectory in which α at any instant was α^* at the current Mach number.

The two series of trials resulted in several useful observations. Firstly, as was expected, the maximum range achieved increased asymptotically as the number of control nodes was increased. The higher the resolution given to the control profile by adding nodes, the greater the maximum range became, but only up to a point. Initial doublings of the number of intervals between nodes resulted in larger improvements in maximum range, but later doublings did not result in such large improvements. These

later doublings required much more computational time and were not computationally cost-effective for the small improvements they yielded. Some compromise should be made between maximizing accuracy and minimizing computational time. Secondly, the inherited-initials approach resulted in comparable maximum ranges to the zero-initials approach but typically with a greater cumulative computational time. Not until large numbers of nodes were involved did the inherited-initials approach truly pay off with lesser computational times and improved convergence ability.

Figure 3.3 shows the maximum ranges achieved for various numbers of control nodes using both the inherited-initials and the all-zero-initials approaches, compared with the range achieved by flying a max-*L/D* trajectory. Figure 3.4 shows the corresponding percentage improvements of those ranges and their respective cumulative computational costs, also compared with the max-*L/D* trajectory. The relationship between number of control nodes and cumulative computational time for each approach is shown in Fig. 3.5.

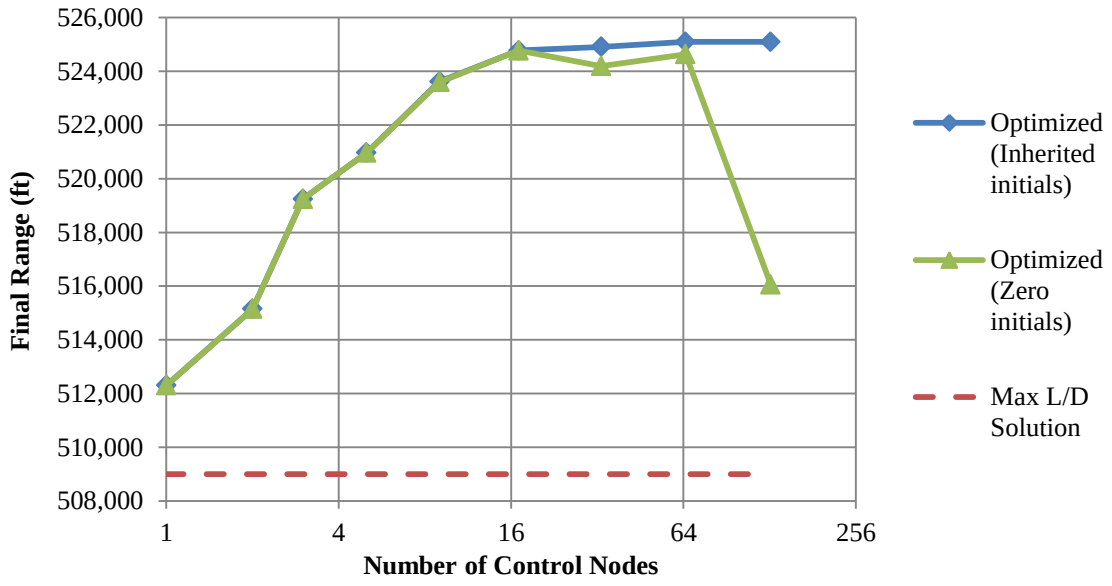

Figure 3.3. Optimal final range achieved for varying numbers of control nodes, using both inherited-initials and zero-initials approaches, compared to max-L/D solution.

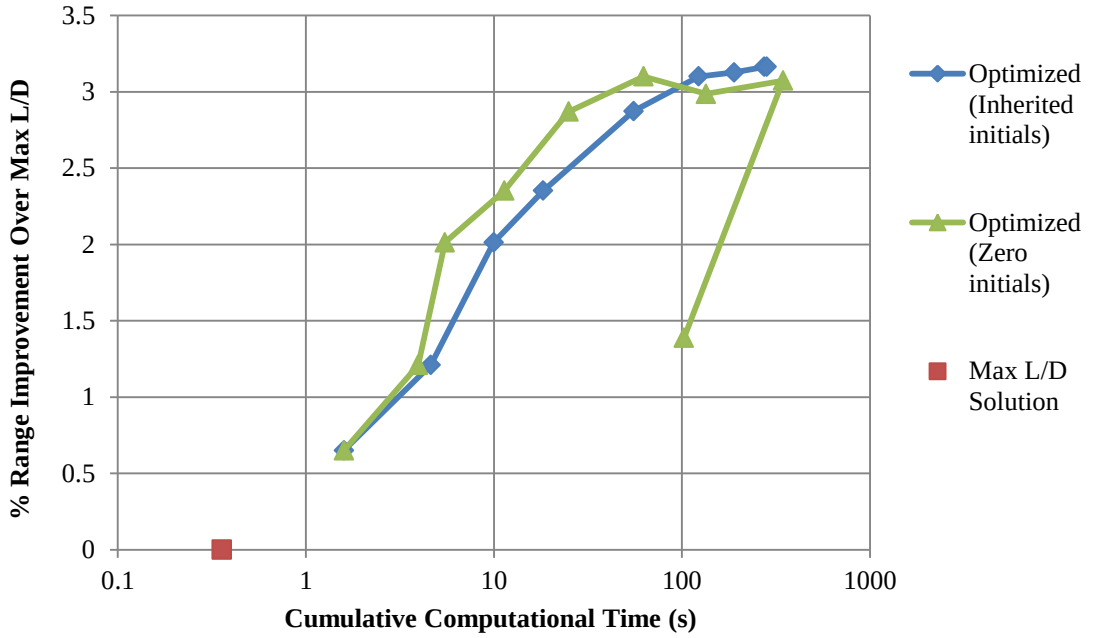

Figure 3.4. Percentage improvement of optimal ranges over max-L/D solution as function of cumulative computational time, using both inherited-initials and zero-initials approaches.

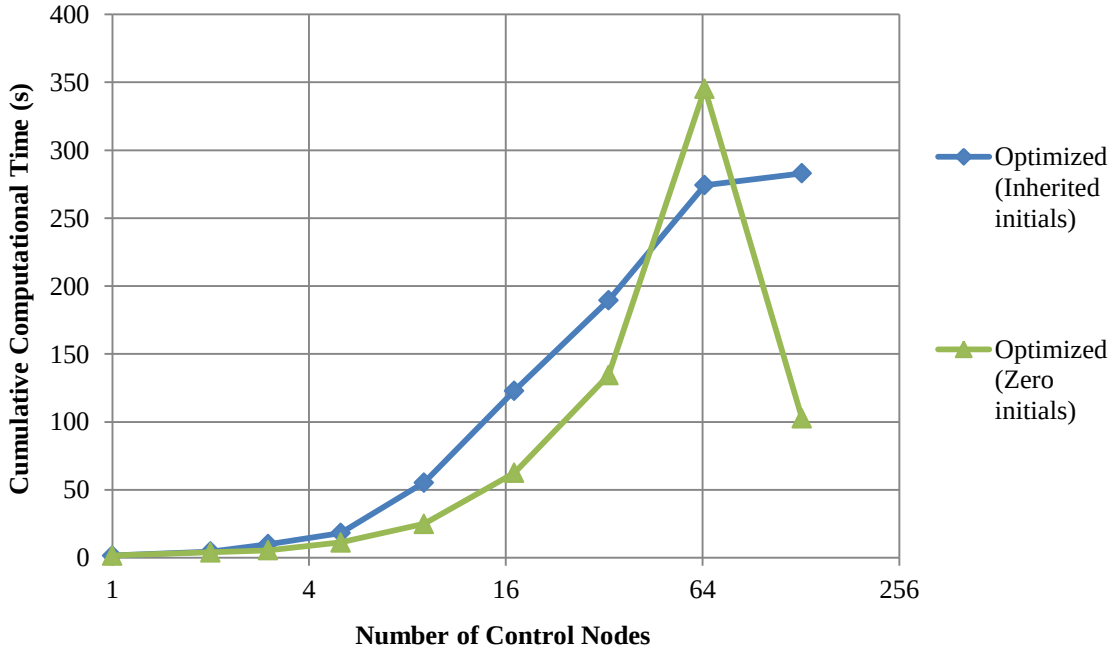

Figure 3.5. Cumulative computational times required to compute optimal trajectories with various numbers of control nodes, using both inherited-initials and zero-initials approaches.

In all three figures (Figs. 3.3-3.5) the last three zero-initials trials (33, 65, and 129 nodes) did not converge to the true optimum, settling instead to suboptimal ranges. This non-convergence illustrates one advantage of the inherited-initials approach: Because each trial begins with the optimum from the last trial, the new trial has a better chance of converging on an optimal control profile, whereas if each trial starts from an all-zero profile, it is harder to find the optimal profile when there are large numbers of independent variables (i.e., control nodes). Despite the superior performance of the inherited-initials approach for most trials, the 129-node inherited-initials trial did not improve upon the 65-node trial, resulting in exactly the same range. This lack of improvement illustrates that convergence on the true optimum worsens as the number of independent variables increases.

It is possible that tightening the convergence tolerances of the MATLAB `fmincon` function could improve the performance of one or both of these approaches for higher numbers of nodes, but to do so would increase computational time. Without this adjustment, the zero-initials approach is faster to compute for most numbers of nodes tested (33 or fewer), as seen in Fig. 3.5. Notice that the zero-initials approach was also faster for the 129-node trial, but if the approach converged on the true optimum for this trial, the computational time would most likely be much greater than that of the inherited-initials approach. All trials are conducted using an AMD Athlon II X4 630 quad-core processor running MATLAB R2010a in Windows 7.

Notice that the greatest improvement over a max-*L/D* trajectory, about 3.16 %, occurred in the 65- and 129-node inherited-initials trials. These trials, however, took about 274.3 s (4.572 min) and 283.0 s (4.717 min) to compute. A much faster trial that

resulted in about 3.10 % improvement over max-*L/D* was the 17-node zero-initials trial, requiring only 62.5 s (1.04 min). The faster trial undershoots the one with greater range by 317.9 ft in a trajectory of 525,093.4 ft (65-node inherited initials). For the purposes of this investigation, the 17-node zero-initials trajectory optimization should be sufficient. Though 62.5 s may not be sufficiently low to update the control profile in real time with an onboard control system, this trajectory required 816.9 s to fly, which is enough to conduct the optimization thirteen times. As each update is computed, the control system could follow the most recent control profile.

Table 3.1 lists the initial conditions and final energy height used in computing all trials in Figs. 3.3-3.5. Note that the initial energy height was determined by the initial velocity and altitude, rather than being a separate constraint. The initial velocity and altitude were selected to be representative of initial values for TAEM [6,7]. The initial flight-path angle was not allowed to vary in these trials but was constrained to the value given in Table 3.1 because optimized initial flight-path angles are typically positive (near +15 deg, for example) and highly unrealistic for the TAEM phase. The initial flight-path angle was chosen to be consistent with a Constant-Dynamic-Pressure Quasi-Equilibrium Glide (CDPQEG) trajectory, a concept to be discussed in Section 4.3. Initial flight-path angle is given in deg in Table 3.1 for convenience, but γ must be in rad for integration of $d\gamma/de$ (Eq. (2.8)). The initial velocity at 70,000 ft results in a Mach number of 1.5, which is less than the maximum Mach number available in the piecewise aerodynamic model (Mach 2.5). Though these initial conditions do not make full use of the aerodynamic model, they do provide a fairly realistic starting point for a TAEM trajectory. Both sets of trials used the piecewise aerodynamic and atmospheric models.

Table 3.1. Initial and final states of numerical optimization trials shown in Figs. 3.3-3.5.

Trajectory Boundary	Velocity V (ft/s)	Altitude h (ft)	Flight-path Angle γ (deg)	Energy Height e (ft)
Initial States	1500	70,000	-7.56	104,966.1
Final States	--	--	--	14,514.8

All trajectories in Figs. 3.3-3.5 were integrated to the final energy height given, which was chosen to approximate the states of the vehicle at the A&L interface. Typically A&L begins at about Mach 0.5 at an altitude of 10,000 ft [6-7], which corresponds to a velocity of 539 ft/s. The final energy height e_f is then computed as $e_f = V^2/(2g) + h = 14{,}514.8$ ft. The final energy height does not guarantee that the final altitude or velocity will be those of the A&L interface, but it does require that the final energy of the vehicle be equal to that of the vehicle at the altitude and velocity of the A&L interface. If it were critical that specific values of velocity, flight-path angle, or altitude were met for the interface between the TAEM and A&L phases, then terminal state constraints (as in Eqs. (3.2)-(3.4)) could be enforced by the MATLAB `fmincon` function by referring it to a nonlinear constraints function.

The trials in Figs. 3.3-3.5 were conducted with a maximum integration step size of 1,000 ft (i.e., energy height cannot decrease by more than 1,000 ft in a given integration step). Maximum step sizes above this resulted in lower maximum ranges achieved and optimal control profiles with less detail (i.e., smoother). Maximum step sizes below this increased computational time and did not seem to increase maximum range significantly. If the maximum step size is 1,000 ft, the integration must consist of at least 91 steps (i.e., $(e_0 - e_f)$ / (1,000 ft), rounded up to nearest integer), which should be sufficient to give a fairly high-resolution plot of the trajectory. A thorough

investigation of the relationship between integration step size and maximum range achieved might help explain the non-convergence associated with high numbers of nodes.

Figures 3.6-3.12 depict the states of the vehicle, as well as Mach number and dynamic pressure, throughout the inherited-initials trajectories of Figs. 3.3-3.5, except for the 129-node trajectory. The 129-node solution is omitted because it is identical to the 65-node solution. The optimization algorithm was unable to improve upon the 65-node trajectory using the inherited-initials approach and the same parameters as the other trials (the initial and final states in Table 3.1, integration step size, etc.).

The velocity profiles of each trajectory are fairly similar (seen in Fig. 3.6), terminating at velocities near 400 ft/s in each trial, which is significantly lower than the target velocity of 539 ft/s used to calculate the final energy height e_f, but it is not unreasonably low for beginning the A&L phase, corresponding to about Mach 0.4. Altitude profiles for the various trials are especially close together, as seen in Fig. 3.7. As might be expected, where the final velocity of the vehicle is below the target, the final altitude of most trials is above the target altitude—12,000 ft instead of 10,000 ft—which is consistent with integrating to a specified energy height, where $e = V^2/(2g) + h$. The lower final velocity requires a higher final altitude for the specified final energy height.

Dynamic pressure throughout the trajectories varies much more severely than velocity (see Fig. 3.8), due in part to dynamic pressure varying with the square of velocity, so that any variation in velocity is amplified substantially in dynamic pressure. Dynamic pressure also varies with altitude as atmospheric density changes, further separating the trajectories by their differences in altitudes (though the altitude profiles are very similar for each trial).

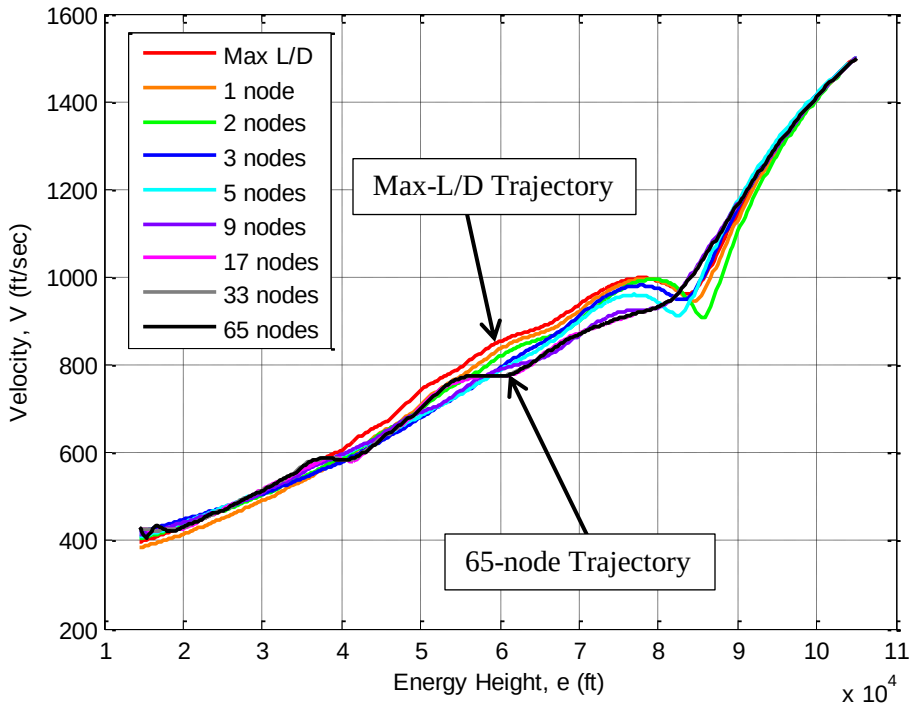

Figure 3.6. Velocity vs. energy height of trajectories optimized with `fmincon` (inherited initials, various numbers of nodes), compared to max-L/D trajectory.

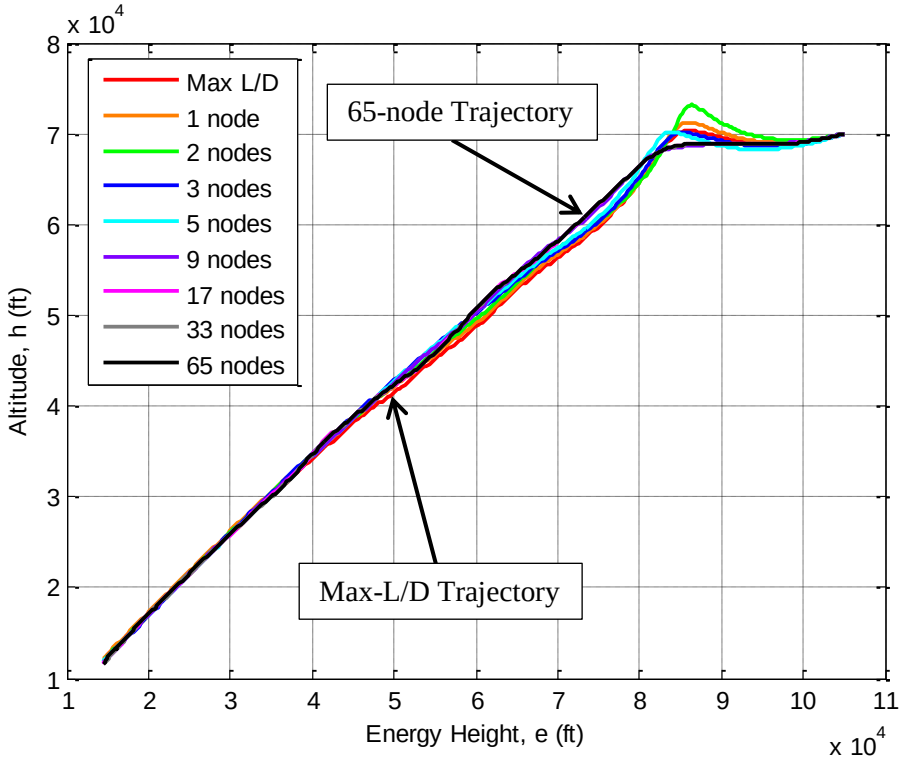

Figure 3.7. Altitude vs. energy height of trajectories optimized with `fmincon` (inherited initials, various numbers of nodes), compared to max-L/D trajectory.

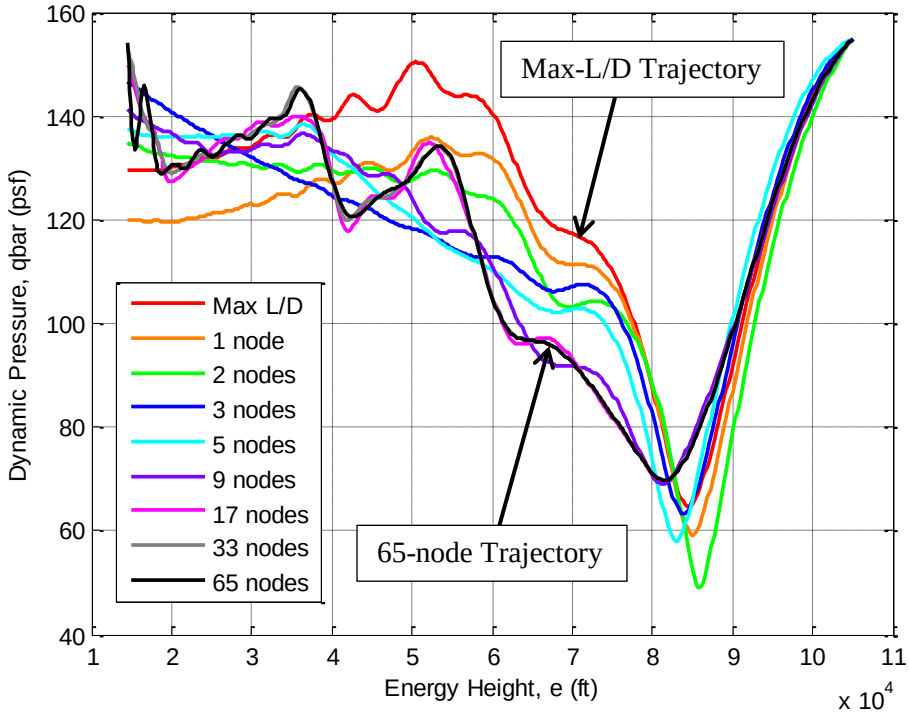

Figure 3.8. Dynamic pressure vs. energy height of trajectories optimized with `fmincon` (inherited initials, various numbers of nodes), compared to max-L/D trajectory.

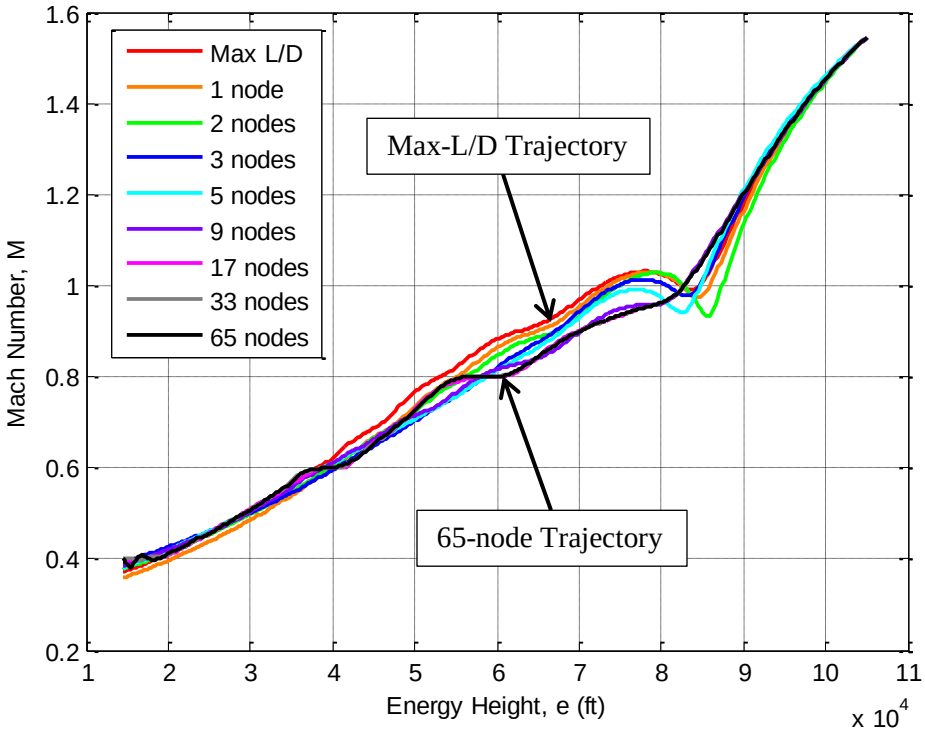

Figure 3.9. Mach number vs. energy height of trajectories optimized with `fmincon` (inherited initials, various numbers of nodes), compared to max-L/D trajectory.

As number of control nodes is increased, dynamic pressure profiles seem to gravitate toward the profile of the 65-node trial, having begun with a much different shape in the max-L/D and 1-node profiles. Given that Mach number is a function of velocity and altitude, both of which are fairly consistent among the trials, it is not surprising that Mach number should also be consistent among the trials (see Fig. 3.9).

Among the flight-path-angle profiles (see Fig. 3.10) there is a significant amount of variation. Most profiles tend to converge on a linear trend toward the end of the flight (i.e., as energy height decreases, so on the left side of the plot). This tendency is of great interest, as it could be indicative of a tendency for maximum-range trajectories to converge on the bottom of a Quasi-Equilibrium-Glide (QEG) drag valley. This drag valley will be addressed in Chapter 4. The 65-node profile appears to oscillate outside of this linear trend at the very end of the flight. It is not known why this behavior is optimal, but such behavior could help to understand how the drag valley functions.

It is also interesting to note that most of the optimal trajectories use positive flight-path angles of significant magnitude early in the trajectory. Such behavior amounts to lobbing the vehicle at a high angle above the horizon to give it additional altitude, keeping it aloft longer, provided that the maneuver does not cost the vehicle more energy in the future in recovering from too severe a motion. Hence, the first oscillation of this sort has the greatest magnitude. Subsequent oscillations do occur but with progressively smaller magnitudes. Subsequent large oscillations would impair the final range by requiring the vehicle to make large control efforts in the future to recover (from an unstable maneuver, for instance), which would likely cost the vehicle more energy than it saved by increasing induced drag due to the high angle of attack.

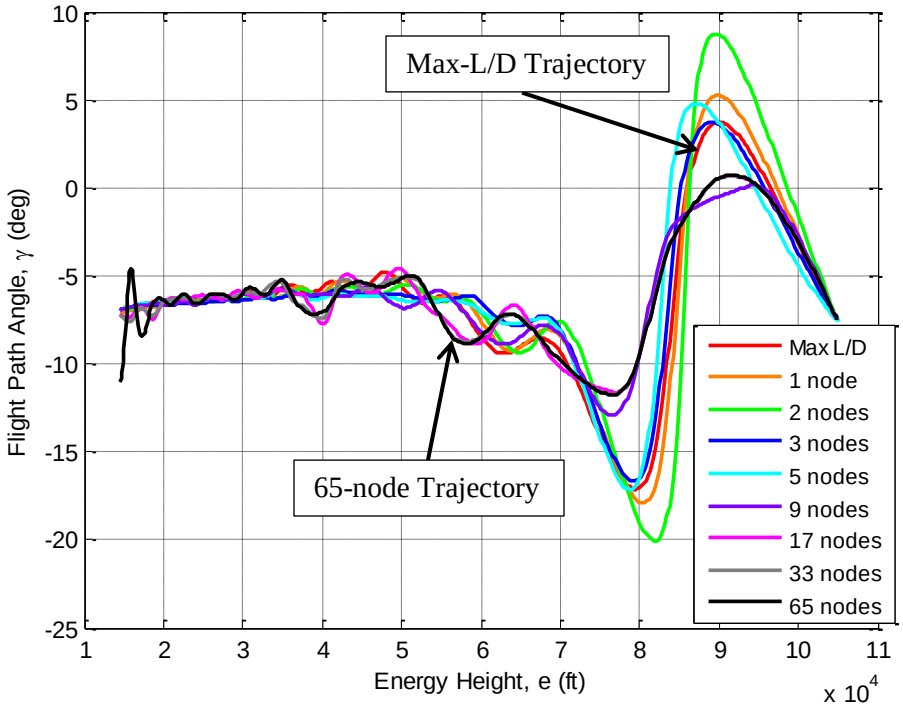

Figure 3.10. Flight-path angle vs. energy height of trajectories optimized with `fmincon` (inherited initials, various numbers of nodes), compared to max-L/D trajectory.

This oscillatory motion is characteristic of most maximum-range trajectories, and even in the case of the drag valley, it is hypothesized that maximum-range trajectories converge on the bottom of the drag valley by oscillating about it with smaller and smaller amplitude.

Figure 3.11 shows the ranges achieved by the inherited-initials trials, from which it can be seen that the range profiles are quite consistent with each other. As the number of nodes increases, the final range also increases, as was illustrated in the plot of range as a function of number of nodes (Fig. 3.3). Nevertheless, the variation between a max-L/D trajectory, which has zero nodes, and a 65-node trajectory is not a very large percentage—about 3.16 % at most, as seen in Fig. 3.4.

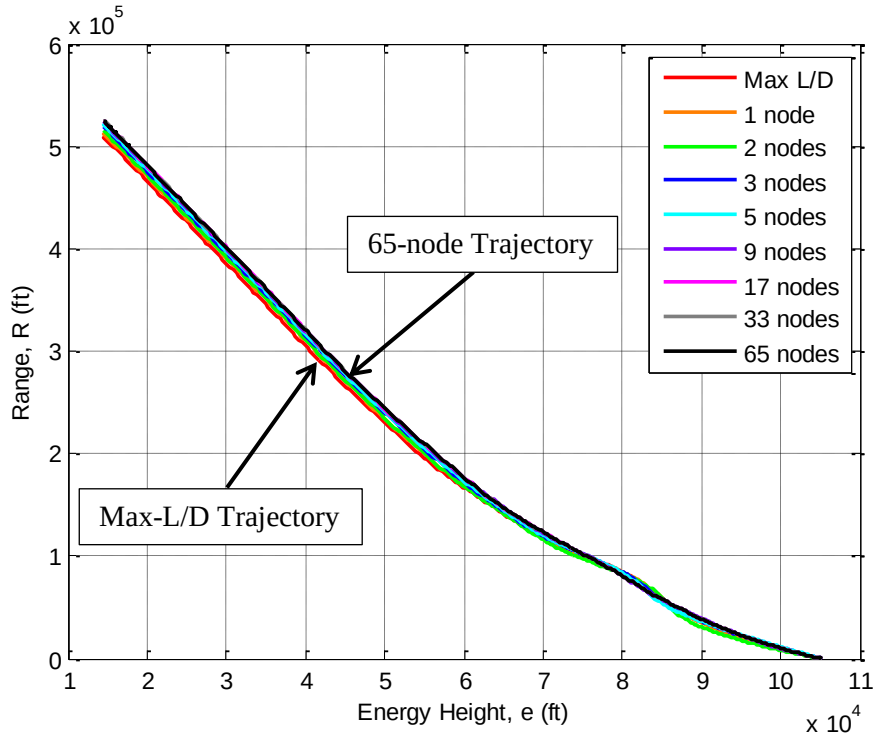

Figure 3.11. Range flown vs. energy height of trajectories optimized with `fmincon` (inherited initials, various numbers of nodes), compared to max-L/D trajectory.

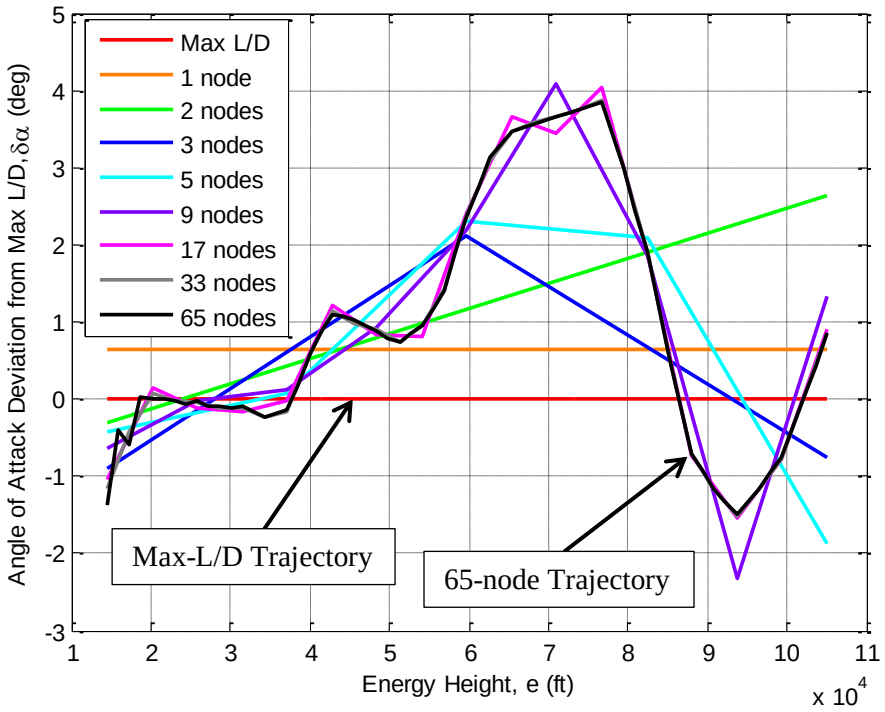

Figure 3.12. Control profiles (angle-of-attack deviation ($\delta\alpha(e)$) from max-L/D vs. energy height) of trajectories optimized with `fmincon` (inherited initials, various numbers of nodes), compared to max-L/D trajectory.

Finally, the control profiles of each inherited-initials trial can be found in Fig. 3.12. Here the deviations in angle of attack ($\delta\alpha(e)$) with respect to the angle of attack for maximum *L/D* (α^*) are shown as functions of energy height. The number of nodes in each trajectory can be seen in the figure. The max-L/D trajectory has no control nodes and follows α^* exactly, so $\delta\alpha(e) = 0$ for all e. The 1-node profile has only one degree of freedom to optimize, resulting in a constant offset where $\delta\alpha(e) = constant$ for all e. The 2-node profile has one linear piece, the 3-node profile has two linear pieces, the 5-node profile has four linear pieces, and so forth. In particular, it can be seen from Fig. 3.12 that increasing the number of control nodes causes the control profile to resolve more and more clearly into the optimal shape. Where the 9-node profile approximates the optimal profile with a sharp peak near 70,000 ft, the 17-node profile approximates the optimal profile with several shorter linear pieces through the same region. Both of these trials appear to be converging on the 65-node profile, which simply has a slanting line through the same region.

This convergence behavior is reminiscent of the convergence of a Fourier series on a particular function by adding more and more terms to the series. The convergence in both situations occurs by adding new functions of higher frequency to the previous summation. For this reason, a Fourier series might be a good method of defining the control profile. As mentioned in Section 2.4, other methods of defining the control profile, such as using a higher-order interpolation method (e.g., spline fit or Fourier series), might improve the computational efficiency of the optimization process by more closely approximating the optimal control profile for a given number of control nodes. Though such an approach would require more computational time for each function

evaluation (due to the higher-order interpolation used), it could reduce the overall computational time by reducing the number of control nodes (and, hence, the number of function evaluations) required to achieve a given range.

3.3. Comparison of Optimization Algorithms

Alternative optimization algorithms have been considered and tested, though not to the same extent as the MATLAB `fmincon` function. De Ridder [20] uses a genetic algorithm to find maximum-range trajectories. Stochastic methods, including genetic algorithms, are better than gradient-based methods for finding maxima in multimodal functions, but computationally they are much slower than gradient methods for continuous, unimodal functions. Though no thorough mathematical proof has been undertaken in this research, it was believed that the objective function for this problem was continuous and unimodal. A preliminary test was conducted to investigate this belief by plotting the objective function for a two-node control profile, such as the one in Fig. 3.13, over a range of values of each node. Figure 3.14 shows the result of this investigation.

From Fig. 3.14 it appears that the objective function is continuous and unimodal over much of the design space with the same initial and final conditions as in Table 3.1. Recall that α can vary between -6 deg and 21 deg when using the piecewise aerodynamic model, so any given control node $\delta\alpha(e_i)$ should not cause $\alpha(e_i)$ to exceed this range (see Eq. (3.5) and Fig. 3.1). α^* varies with Mach number, though, making it impossible to tell where the edge of the design space is in this figure. Nevertheless, this figure includes the maximum achievable range and much of the usable design space.

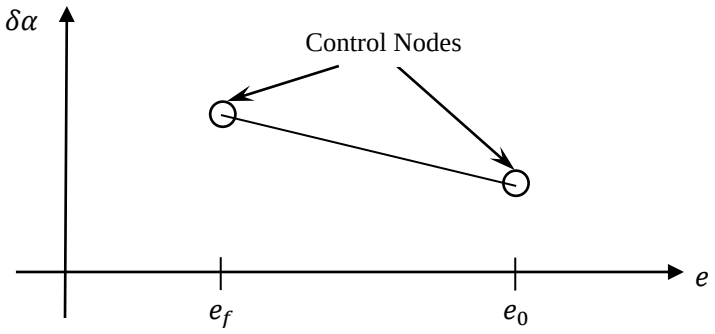

Figure 3.13. Two-node control profile definition, in which control nodes specify $\delta\alpha$ at initial energy height e_0 and final energy height e_f. $\delta\alpha$ at intermediate energy heights is found by interpolation.

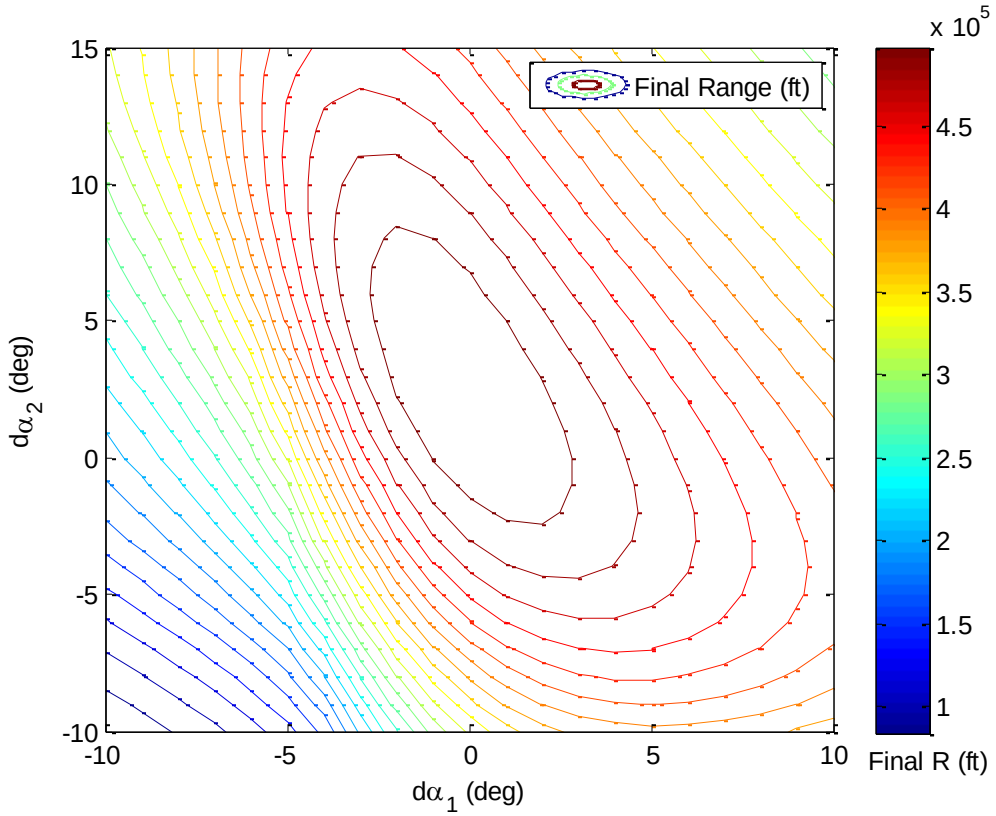

Figure 3.14. Range achieved for various values of each control node in a two-node control profile.

The fact that this two-variable objective function was continuous and unimodal provides no guarantee that the objective function is continuous and unimodal for higher numbers of nodes. To be thorough, then, an analytical investigation of the gradient and Hessian of the objective function should be conducted to determine its behavior for higher numbers of nodes. Consistent with the belief that the objective function is unimodal, however, and consistent with the inferior performance of alternative (especially stochastic) algorithms on the objective function during preliminary tests, this study focuses primarily on gradient-based methods when conducting numerical optimization.

Particle-swarm optimization (PSO) was tested on the objective function as an example of a non-gradient-based approach. PSO is initialized with a randomly-selected set of candidate solutions, where each candidate solution is called a particle. Each particle is initialized with a randomly-selected velocity, where the velocity is a vector that is added to the particle's current solution vector to determine what the particle's candidate solution vector will be in the next iteration. Following the first iteration, the particle velocities are calculated by a weighted average of three components, where the weights are parameters selected by the user. Each component is also randomly weighted. The first component is the particle's previous velocity—i.e., the particle will tend to continue in the direction it was previously moving in the design space. The second component is the difference between the particle's current location and the best solution that particle has found thus far—i.e., the particle will tend toward its personal best solution. The third component is the difference between the particle's current location and the best solution found by any particle—i.e., the particle will tend toward the global

best solution yet found. Ideally, then, the particles will have some inertia and will not forever be changing direction, but they will (especially later in the process) tend to converge on regions that are most promising, while remaining somewhat free to peruse other promising regions found along the way.

Though the maximum ranges achieved with PSO were comparable to those of the gradient-based `fmincon`, the computational time was far greater for PSO than for the gradient-based approach, due to the stochastic nature of PSO. Furthermore, PSO would only provide comparable maximum ranges if it had a sufficient number of particles and iterations. For higher numbers of control nodes, the number of particles or iterations must be increased to continue to compete with the gradient-based approach. Figure 3.15 illustrates the performance of PSO trials for various numbers of control nodes given the same initial and final conditions as described in Table 3.1. The PSO trials are compared to the gradient-based trials with corresponding numbers of nodes and the max-*L/D* trajectory that are displayed in Fig. 3.3. The code written for the PSO algorithm can be seen in Appendix B.

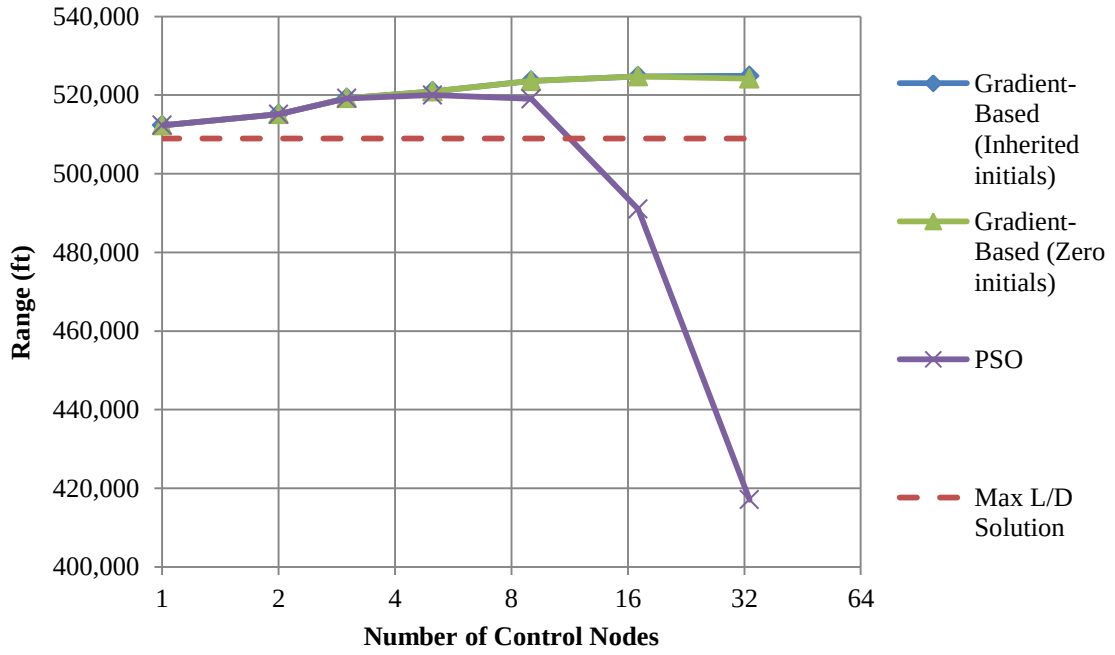

Figure 3.15. Comparison of range achieved with PSO to range achieved with gradient-based optimization for various numbers of nodes, as well as max-*L/D* solution.

All the PSO trials were conducted with 200 particles and 10 iterations. No convergence criterion was set on the PSO algorithm because the trials that were attempted with a relative error convergence criterion frequently ended before reaching a near-optimal solution. Note from Fig. 3.15 that suboptimal solutions were still frequent using the fixed number of iterations. Increasing the number of iterations did not necessarily result in a better solution, however, as a separate 17-node trial was conducted with 20 iterations and resulted in a significantly worse solution than the one shown. This behavior is due to the stochastic nature of PSO, which begins with a random sampling of particles, so there is no guarantee that one trial will perform better or worse than another.

Given the fixed number of particles and iterations, all PSO trials in Fig. 3.15 required about the same amount of computational time. The fastest trial (2-node) required 98.0 s (1.63 min), which is substantially longer than the cumulative

computational times of the corresponding gradient-based trials: 4.62 s (0.0770 min) for the inherited-initials trial and 3.95 s (0.0658 min) for the zero-initials trial. The longest PSO trial (33-node) required 122.0 s (2.033 min), which is somewhat shorter than the corresponding gradient-based trials: 189.5 s (3.159 min) for the inherited-initials trial and 134.4 s (2.240 min) for the zero-initials trial (which also ended suboptimally, but with a 2.99 % improvement over the max-*L/D* solution, compared to the 18.0 % decline found in the 33-node PSO trial).

All trials in Fig. 3.15 were also conducted with a particle-velocity weight of 0.1, a personal-best weight of 0.4, and a global-best weight of 0.5. The choice of these parameters plays a critical role in the performance of the PSO algorithm, so a more thorough investigation of PSO should involve finding how these affect performance.

CHAPTER 4: QUASI-EQUILIBRIUM GLIDE

4.1. Overview

As discussed in Section 1.3, the traditional approach to maximizing the range of a glider is to fly a max-*L/D* trajectory. In this investigation the angle of attack corresponding to max *L/D* can be found with Eqs. (2.20)-(2.21) when using a constant drag-polar as the aerodynamic model, and with Eq. (2.25) and the relationship shown in Fig. 2.2 when using a piecewise-polynomial aerodynamic model. To consider whether there may be a better approximation of the maximum-range trajectory, it is good to consider the reasoning behind flying a max-*L/D* trajectory.

Recall the equation for the time rate of change for range R (Eq. (2.4)):

$$\dot{R} = V\cos\gamma \tag{4.1}$$

where V is velocity and γ is flight-path angle. One approach to maximizing range, then, would be to attempt to keep $\dot{R}$ as large as possible for as much time as possible. The traditional approach is to seek an equilibrium flight, in which V and γ remain constant, thereby causing $\dot{R}$ to remain constant as well. To enforce these constraints, the equations of motion for $\dot{V}$ and $\dot{\gamma}$ (Eqs. (2.1)-(2.2)) are set equal to zero:

$$\dot{V} = -\frac{D(V,h,\alpha)}{m} - g\sin\gamma = 0 \tag{4.2}$$

$$\dot{\gamma} = \frac{L(V,h,\alpha)}{mV} - \frac{g}{V}\cos\gamma = 0 \tag{4.3}$$

where $L(V,h,\alpha) = \frac{1}{2}\rho(h)V^2SC_L$ and $D(V,h,\alpha) = \frac{1}{2}\rho(h)V^2SC_D$, as given in Eqs. (2.12)-(2.13). Note that C_L and C_D are functions of α when using a constant drag-polar aerodynamic model (see Eqs. (2.15)-(2.16)) or functions of α and Mach number when using a piecewise polynomial model (see Eqs. (2.23)-(2.24) and Fig. 2.1). Note also that Mach number is a function of V and speed of sound V_s (see Eq. (2.22)), which changes with altitude (see Eq. (2.28) and Fig. 2.5). Equations (4.2)-(4.3) can be simplified to

$$D(V,h,\alpha) = -mg\sin\gamma \tag{4.4}$$

$$L(V,h,\alpha) = mg\cos\gamma \tag{4.5}$$

Dividing Eq. (4.5) by Eq. (4.4) results in

$$\frac{L(V,h,\alpha)}{D(V,h,\alpha)} = -\frac{1}{\tan\gamma} \tag{4.6}$$

Hence, equilibrium flight requires that Eq. (4.6) be true. L and D are both positive, so γ must be negative (i.e., below the horizon). To maximize the range of an equilibrium flight, γ must be as close to zero as possible so that $\dot{R}$ is as large as possible (Eq. (4.1)). Minimizing $|\gamma|$ means that $|\tan(\gamma)|$ is also minimized which, given that $\tan(\gamma) < 0$,

maximizes *L/D* according to Eq. (4.6). Therefore, flying an equilibrium glide flight for maximum range requires that *L/D* be maximized.

As noted, L and D are functions of V, h, and α. As h decreases (due to the negative flight-path angle), atmospheric density $\rho(h)$ changes, which threatens to change L and D. According to Eqs. (4.4)-(4.5), L and D must remain constant in order for equilibrium flight to continue. Therefore, C_L and C_D must change to counteract the effect of $\rho(h)$ on L and D, while at the same time, *L/D* must remain constant according to Eq. (4.6). For a constant drag-polar aerodynamic model, C_L and C_D cannot change without affecting *L/D*. If it were possible to change C_L without changing *L/D*, then the derivative of *L/D* with respect to C_L must be zero. This constraint, however, is described in Eq. (2.18) and results in Eqs. (2.19)-(2.20), which require that C_L equal $\sqrt{C_{D_0}/K}$, a constant. Therefore, with a constant drag-polar model, equilibrium flight could only occur if ρ did not vary with h. Note that max *L/D* is still possible by maintaining $\alpha = \alpha^*$, which is a very good approximation of the maximum-range trajectory, but equilibrium flight is not achieved.

If a true equilibrium glide is possible, then, it must be accomplished using the piecewise polynomial aerodynamic model. M changes as altitude changes, causing C_L and C_D to change automatically. To maintain equilibrium flight would require that *L/D* remain constant while C_L and C_D change. From Fig. 2.1 it can be seen that there may be some means of maintaining constant *L/D* for various Mach numbers. It is unclear, however, what effect this would have on V and γ, given that such a control profile would likely require very low or very high values of α as M decreased. These extreme values of α would almost certainly have undesirable impacts on the trajectory, given that following

this control profile would likely violate Eqs. (4.4)-(4.5). It is possible to develop a control algorithm that would attempt to maintain equilibrium conditions. The resulting approach to maximizing range is here termed a Quasi-Equilibrium Glide (QEG), given that a true equilibrium glide (i.e., states remain constant) is not possible.

De Ridder [20] proposed finding the angle of attack and flight-path angle that result in a constant velocity ($\dot{V} = 0$) and constant flight-path angle ($\dot{\gamma} = 0$) for a given velocity and altitude. If the drag corresponding to this quasi-equilibrium flight condition is plotted as a three-dimensional surface over a range of velocities and altitudes, the drag surface forms a valley, called a drag valley, which will be discussed in Section 4.2. De Ridder claims that maximum-range trajectories given initial conditions in the vicinity of the drag valley will tend to follow the bottom of this valley. Therefore, this valley could be a means of approximating the optimal trajectory without conducting the numerical optimization necessary to calculate the trajectory. This approach is here referred to as Constant-Velocity Quasi-Equilibrium Glide (CVQEG).

Another possible set of flight conditions that might approximate maximum-range control profiles is the Constant-Dynamic-Pressure Quasi-Equilibrium Glide (CDPQEG). Like CVQEG, this approach finds an angle of attack and flight-path angle that result in a constant flight-path angle ($\dot{\gamma} = 0$). But now, instead of maintaining a constant velocity, the angle of attack and flight-path angle must result in a constant dynamic pressure ($\dot{\bar{q}} = 0$). A drag valley can also be produced for this approach by plotting a three-dimensional surface comprising the drag values that correspond to the CDPQEG flight conditions for a given velocity and altitude.

These two approaches and their corresponding drag valleys will be discussed in greater depth in Sections 4.2 and 4.3, along with the algorithms used to compute the drag valley for each approach. A control system is considered in Section 4.4 that could be implemented in an onboard guidance system to attempt to enforce the flight conditions for each approach. Brief discussion is made in Section 4.5 of a different type of control system that would follow the bottom of the CDPQEG valley, given the tendency of numerically optimized trajectories to converge on the states along the bottom of this valley.

4.2. Constant-Velocity Quasi-Equilibrium Glide (CVQEG)

The conditions necessary for Constant-Velocity Quasi-Equilibrium Glide (CVQEG) can be determined by setting Eqs. (2.1)-(2.2) equal to zero as in Eqs. (4.2)-(4.3), which are repeated here:

$$\dot{V} = -\frac{D(V,h,\alpha)}{m} - g\sin\gamma = 0 \tag{4.7}$$

$$\dot{\gamma} = \frac{L(V,h,\alpha)}{mV} - \frac{g}{V}\cos\gamma = 0 \tag{4.8}$$

Hence, Eqs. (4.7)-(4.8) can be fully described as functions of three states and the control variable: velocity V, altitude h, flight-path angle γ, and angle of attack α (control). There are, therefore, two equations and four unknowns, so two unknowns must be specified in order for a solution to exist.

De Ridder [20] computes the drag valley, which is plotted as a function of V and h (i.e., plotted over the energy space), by specifying V and h (i.e., energy) for one point in the plot. A guess is made for α, and γ is estimated using Eq. (4.6), providing a candidate set of values for the four unknowns. De Ridder then uses these states to calculate V with the following equation, which is found by solving Eq. (4.8) for V, given that $L(V, h, \alpha) = \frac{1}{2}\rho(h)V^2 C_L$ (from Eqs. (2.12) and (2.14)):

$$V = \sqrt{\frac{2mg\cos\gamma}{\rho S C_L}} \tag{4.9}$$

The value of V calculated with Eq. (4.9) is compared to the value of V specified for the point in the drag valley being computed. If the error between the calculated and specified values is smaller than a predefined error tolerance, then the current candidate set of state values is considered the solution to Eqs. (4.7)-(4.8) for the specified V and h. If the error is not smaller than the error tolerance, then a new guess of α is made and the procedure is reiterated.

In this study it was found that a more efficient procedure for solving Eqs. (4.7)-(4.8) is to use the Newton-Raphson Method or a finite-difference version of the same, similar to the Secant Method. If a constant drag-polar aerodynamic model (Eqs. (2.15)-(2.16)) and an exponential atmospheric model (Eqs. (2.26)-(2.27)) are used, then the partial derivatives of the left-hand sides of Eqs. (4.7)-(4.8) can be found analytically, allowing the Newton-Raphson Method to be used to solve both equations simultaneously. These partial derivatives are

$$\frac{\partial \dot{V}}{\partial \alpha} = -\frac{\rho V^2 S K C_{L\alpha}(C_{L0}+C_{L\alpha}\alpha)}{m} \tag{4.10}$$

$$\frac{\partial \dot{V}}{\partial \gamma} = -g\cos\gamma \tag{4.11}$$

$$\frac{\partial \dot{\gamma}}{\partial \alpha} = \frac{\rho V S C_{L\alpha}}{2m} \tag{4.12}$$

$$\frac{\partial \dot{\gamma}}{\partial \gamma} = \frac{g}{V}\sin\gamma \tag{4.13}$$

Now the Jacobian of the system of equations (Eqs. (4.7)-(4.8)) can be found as

$$[\mathbf{J}] = \begin{bmatrix} \frac{\partial \dot{V}}{\partial \alpha} & \frac{\partial \dot{V}}{\partial \gamma} \\ \frac{\partial \dot{\gamma}}{\partial \alpha} & \frac{\partial \dot{\gamma}}{\partial \gamma} \end{bmatrix} \tag{4.14}$$

Supposing V and h are selected for a particular point, then let the current guess of α and γ be $\{\mathbf{x}_i\} = \begin{Bmatrix} \alpha_i \\ \gamma_i \end{Bmatrix}$ where i is the current iteration of the Newton-Raphson Method. Hence, the next guess of α and γ can be computed as follows [24]:

$$\{\mathbf{x}_{i+1}\} = \{\mathbf{x}_i\} - [\mathbf{J}]^{-1}\{f\} \tag{4.15}$$

where $\{ f \} = \begin{Bmatrix} \dot{V} \\ \dot{\gamma} \end{Bmatrix}$, evaluated with Eqs. (4.7)-(4.8). Note that both [**J**] and $\{ f \}$ must be recomputed on each iteration using the current estimates of α and γ.

If a more complex aerodynamic model is used (instead of the constant drag polar), it becomes more difficult to derive analytical partial derivatives in order to evaluate the Jacobian in Eq. (4.14). Finite-difference approximations of these partial derivatives can be found by evaluating the left-hand sides of Eqs. (4.7)-(4.8) (i.e., $\dot{V}$ and $\dot{\gamma}$) at perturbed angles of attack and flight-path angles. The partial derivative of $\dot{V}$ with respect to α is approximated as

$$\frac{\partial \dot{V}}{\partial \alpha} \cong \frac{\dot{V}(\alpha_i + \Delta\alpha, \gamma_i) - \dot{V}(\alpha_i, \gamma_i)}{\Delta\alpha} \tag{4.16}$$

where $\Delta\alpha$ is some small perturbation in angle of attack, such as 10^{-5} deg, and $\dot{V}(\alpha_i, \gamma_i)$ is $\dot{V}$ evaluated at the guesses for angle of attack and flight-path angle for the *i*th iteration, α_i and γ_i. The other partial derivatives can be approximated similarly as

$$\frac{\partial \dot{V}}{\partial \gamma} \cong \frac{\dot{V}(\alpha_i, \gamma_i + \Delta\gamma) - \dot{V}(\alpha_i, \gamma_i)}{\Delta\gamma} \tag{4.17}$$

$$\frac{\partial \dot{\gamma}}{\partial \alpha} \cong \frac{\dot{\gamma}(\alpha_i + \Delta\alpha, \gamma_i) - \dot{\gamma}(\alpha_i, \gamma_i)}{\Delta\alpha} \tag{4.18}$$

$$\frac{\partial \dot{\gamma}}{\partial \gamma} \cong \frac{\dot{\gamma}(\alpha_i, \gamma_i + \Delta\gamma) - \dot{\gamma}(\alpha_i, \gamma_i)}{\Delta\gamma} \tag{4.19}$$

where $\dot{\gamma}(\alpha_i,\ \gamma_i)$ is $\dot{\gamma}$ evaluated at the guesses for angle of attack and flight-path angle for the *i*th iteration, and $\Delta\gamma$ is some small perturbation in flight-path angle, such as 10^{-5} deg (1.745×10^{-7} rad; since $\dot{\gamma}$ is computed in rad/sec, the solved value of γ should be in rad).

Using either the Newton-Raphson Method or the finite-difference version of the same, a contour or surface plot can be produced to display the drag corresponding to the CVQEG values of the four flight variables (V, h, γ, and α) at a wide range of energy states—i.e., coordinate pairs of V and h, which together define the energy height of the vehicle. De Ridder claims that numerically optimized trajectories tend to follow the bottom of this valley-shaped surface plot, referred to as a drag valley.

To find the bottom of this drag valley, drag values are calculated for a wide range of energy states, and then bilinear interpolation is used (faster than recalculating drag every time via the Newton-Raphson Method) to approximate the drag at numerous V-h pairs along contours of constant energy. The V-h pair with minimum drag along a constant-energy contour is flagged as a point along the bottom of the drag valley. The process is repeated for various energy contours that intersect the drag valley, and the flagged points comprise an approximation of the bottom of the drag valley. This bottom consists of the minimum drag achievable with CVQEG conditions at each energy height. Separate contour or surface plots can be produced to display the corresponding angle of attack, flight-path angle, and dynamic pressure for each energy state. Because these plots

consist of independently-solved sets of flight variables that satisfy CVQEG conditions at each energy state and do not involve integrating the equations of motion throughout a trajectory, there is no guarantee that the bottom of the drag valley is itself a physically flyable trajectory. Rather, this profile serves as a means of predicting the flight conditions (such as flight-path angle) toward which numerically optimized trajectories will tend. It may not be straightforward, then, to implement this profile into an onboard guidance control system for an RLV. However, a control system could be designed to direct the glider to maintain CVQEG conditions as best as possible. Such an implementation will be discussed in Section 4.4.

Figure 4.1 illustrates the CVQEG drag valley as a colored contour plot, and selected contours of constant energy height are plotted over the drag valley. The approximate bottom of the drag valley, as found by minimizing drag along these contours of constant energy height, is also displayed in the figure. Figures 4.2-4.4 illustrate the flight-path angles, angles of attack, and dynamic pressures, respectively, that satisfy CVQEG conditions (and, hence, that correspond to the CVQEG drag) at each point along the drag valley.

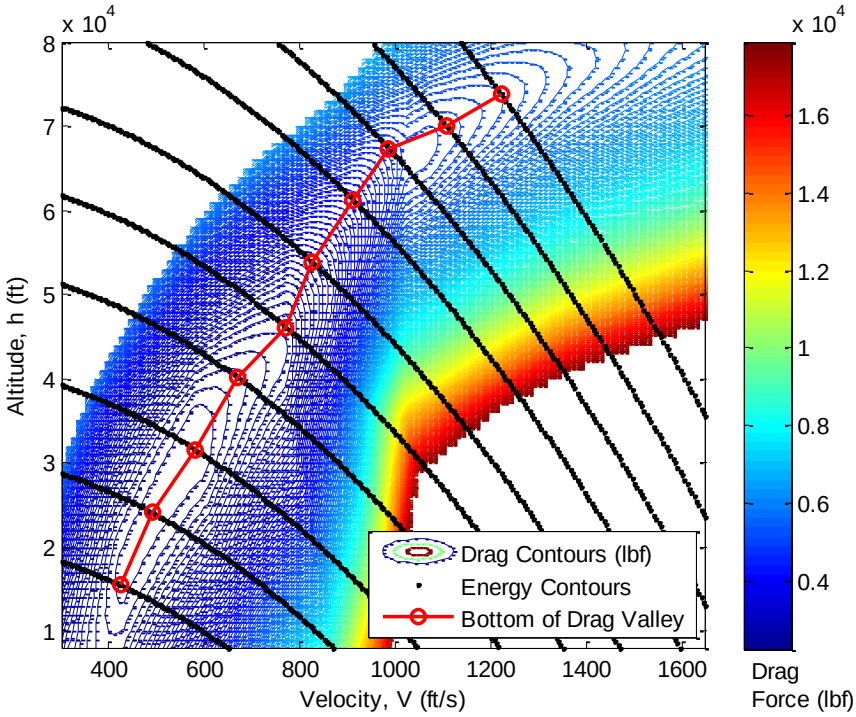

Figure 4.1. Drag corresponding to CVQEG (i.e., drag valley) with selected contours of constant energy height (black) and minimum-drag points along those contours (red circles), approximating bottom of the drag valley.

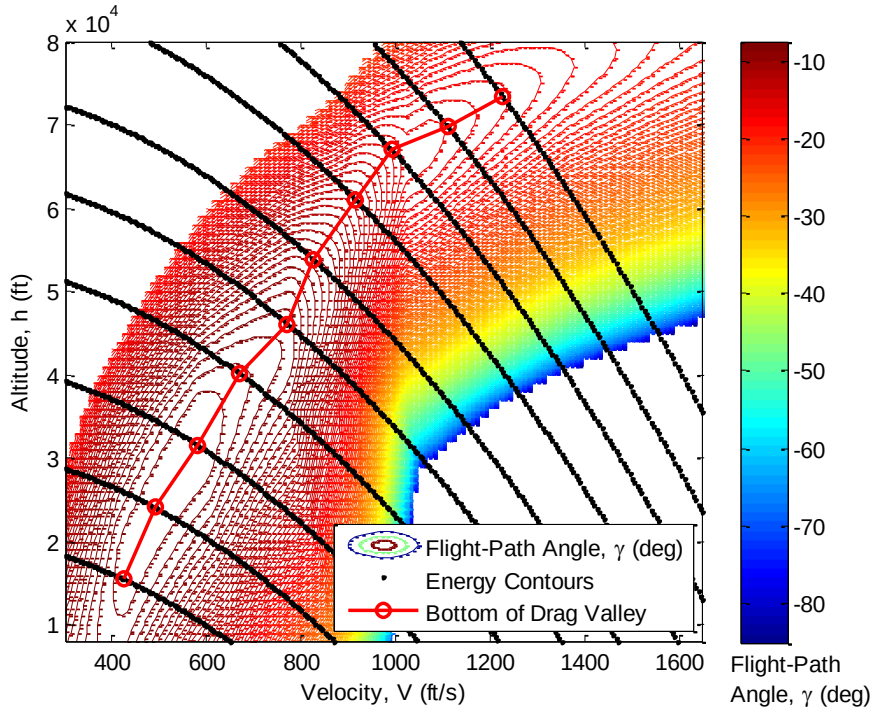

Figure 4.2. Flight-path angle corresponding to CVQEG drag valley with selected contours of constant energy height (black) and minimum-drag points along those contours (red circles), approximating bottom of the drag valley.

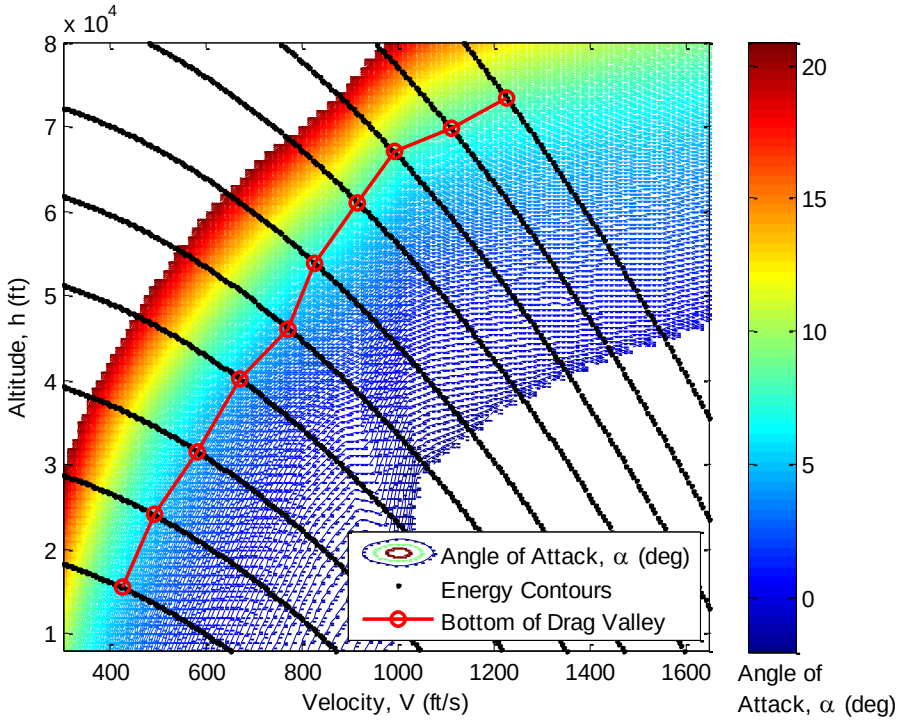

Figure 4.3. Angle of attack corresponding to CVQEG drag valley with selected contours of constant energy height (black) and minimum-drag points along those contours (red circles), approximating bottom of the drag valley.

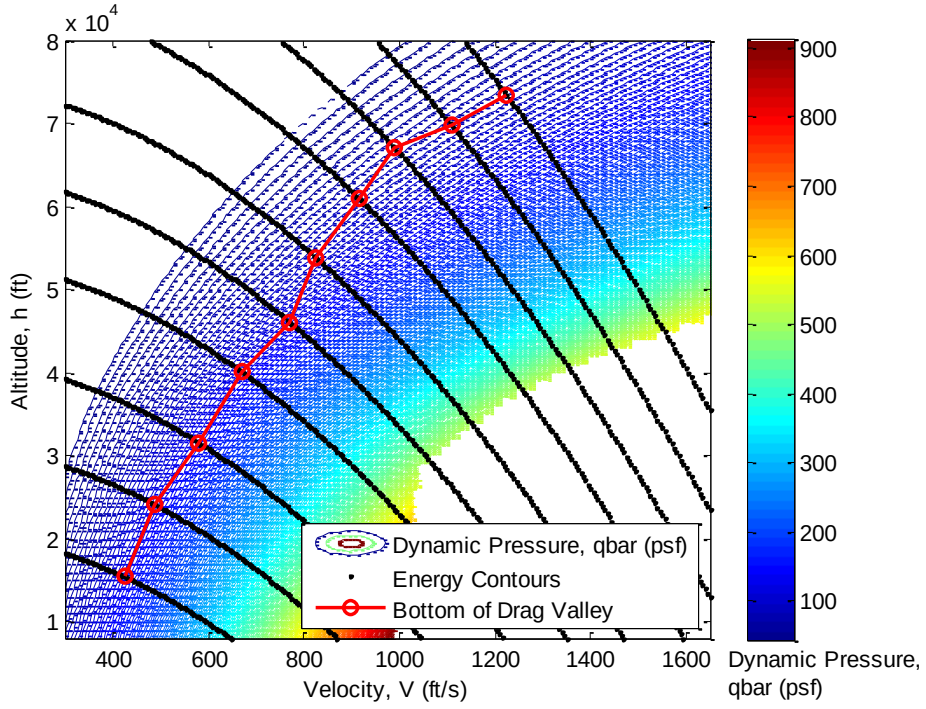

Figure 4.4. Dynamic pressure corresponding to CVQEG drag valley with selected contours of constant energy height (black) and minimum-drag points along those contours (red circles), approximating bottom of the drag valley.

4.3. Constant-Dynamic-Pressure Quasi-Equilibrium Glide (CDPQEG)

The flight conditions for Constant-Dynamic-Pressure Quasi-Equilibrium Glide (CDPQEG) are defined to maintain constant $\bar{q}$ (instead of constant V) and γ. Recall from Eq. (2.14) that

$$\bar{q} = \frac{1}{2}\rho V^2 \tag{4.20}$$

Therefore, the time rate of change of $\bar{q}$ is

$$\dot{\bar{q}} = \rho V \frac{dV}{dt} + \frac{1}{2}\frac{d\rho}{dh}\frac{dh}{dt}V^2 \tag{4.21}$$

Substituting $\dot{V} = dV/dt$ and $\dot{h} = dh/dt$ into Eq. (4.21), and using the equations of motion for $\dot{V}$ and $\dot{h}$ (Eqs. (2.1) and (2.3)), results in

$$\dot{\bar{q}} = \rho V\left(-\frac{D(V,h,\alpha)}{m} - g\sin\gamma\right) + \frac{1}{2}\frac{d\rho}{dh}V^3\sin\gamma \tag{4.22}$$

where $D(V,h,\alpha) = \frac{1}{2}\rho(h)V^2SC_D$, as given in Eqs. (2.13)-(2.14).

To maintain constant $\bar{q}$ and γ requires that $\dot{\bar{q}} = 0$ and $\dot{\gamma} = 0$, providing the conditions for CDPQEG flight (Eq. (4.24) is repeated from CVQEG conditions—see Eq. (4.8)):

$$\dot{\bar{q}} = \rho V\left(-\frac{D(V,h,\alpha)}{m} - g\sin\gamma\right) + \frac{1}{2}\frac{d\rho}{dh}V^3\sin\gamma = 0 \tag{4.23}$$

$$\dot{\gamma} = \frac{L(V,h,\alpha)}{mV} - \frac{g}{V}\cos\gamma = 0 \tag{4.24}$$

where $L(V,h,\alpha) = \frac{1}{2}\rho(h)V^2SC_L$, as given in Eqs. (2.12) and (2.14). Note that C_L and C_D are functions of α when using a constant drag-polar aerodynamic model (see Eqs. (2.15)-(2.16)) or functions of α and Mach number when using a piecewise polynomial model (see Eqs. (2.23)-(2.24) and Fig. 2.1). Note also that Mach number is a function of V and speed of sound V_s (see Eq. (2.22)), which changes with altitude (see Eq. (2.28) and Fig. 2.5). Furthermore, the rate of change of density with altitude, $d\rho/dh$, is also a function of altitude. Hence, as with the CVQEG flight conditions, Eqs. (4.23)-(4.24) can be fully described as functions of three states and one control variable: velocity V, altitude h, flight-path angle γ, and angle of attack α (control). There are, therefore, two equations and four unknowns, so two unknowns must be specified in order for a solution to exist.

The Newton-Raphson Method (see Eq. (4.15)) can also be employed here to find values of α and γ to satisfy Eqs. (4.23)-(4.24) for a given energy state (pair of V and h values), using either an analytical or finite-difference Jacobian. The Jacobian is identical to that of Eq. (4.14) except every $\dot{V}$ is replaced with $\dot{\bar{q}}$:

$$[\mathbf{J}] = \begin{bmatrix} \frac{\partial\dot{\bar{q}}}{\partial\alpha} & \frac{\partial\dot{\bar{q}}}{\partial\gamma} \\ \frac{\partial\dot{\gamma}}{\partial\alpha} & \frac{\partial\dot{\gamma}}{\partial\gamma} \end{bmatrix} \tag{4.25}$$

Therefore, if a constant drag-polar aerodynamic model is used, $\partial\dot{V}/\partial\alpha$ and $\partial\dot{V}/\partial\gamma$ (Eqs. (4.10)-(4.11)) are replaced with

$$\frac{\partial\dot{\bar{q}}}{\partial\alpha} = -\frac{\rho^2 V^3 SKC_{L\alpha}(C_{L0}+C_{L\alpha}\alpha)}{m} \tag{4.26}$$

$$\frac{\partial\dot{\bar{q}}}{\partial\gamma} = \left(\frac{1}{2}\frac{d\rho}{dh}V^3 - \rho V g\right)\cos\gamma \tag{4.27}$$

The equations for $\partial\dot{\gamma}/\partial\alpha$ and $\partial\dot{\gamma}/\partial\gamma$ (Eqs. (4.12)-(4.13)) are still valid, given that the flight condition of $\dot{\gamma} = 0$ is common to both QEG approaches:

$$\frac{\partial\dot{\gamma}}{\partial\alpha} = \frac{\rho V S C_{L\alpha}}{2m} \tag{4.28}$$

$$\frac{\partial\dot{\gamma}}{\partial\gamma} = \frac{g}{V}\sin\gamma \tag{4.29}$$

Note also that the Newton-Raphson Method (Eq. (4.15)) now requires that $\{f\} = \begin{Bmatrix}\dot{\bar{q}}\\ \dot{\gamma}\end{Bmatrix}$ (see left-hand sides of Eqs. (4.23)-(4.24)) instead of $\{f\} = \begin{Bmatrix}\dot{V}\\ \dot{\gamma}\end{Bmatrix}$, as for CVQEG. Both $[\mathbf{J}]$ and $\{f\}$ still need to be recomputed on each iteration using the current estimates of α and γ.

As in the CVQEG algorithm, finite-difference approximations of these partial derivatives can be calculated (helpful when using a piecewise-polynomial aerodynamic model) by evaluating $\dot{\bar{q}}$ and $\dot{\gamma}$ (see left-hand sides of Eqs. (4.23)-(4.24)) at perturbed angles of attack and flight-path angles. The finite-difference approximations used for CVQEG (Eqs. (4.16)-(4.19)) are kept valid for this purpose by replacing every $\dot{V}$ with $\dot{\bar{q}}$:

$$\frac{\partial \dot{\bar{q}}}{\partial \alpha} \cong \frac{\dot{\bar{q}}(\alpha_i+\Delta\alpha, \gamma_i)-\dot{\bar{q}}(\alpha_i, \gamma_i)}{\Delta\alpha} \tag{4.30}$$

$$\frac{\partial \dot{\bar{q}}}{\partial \gamma} \cong \frac{\dot{\bar{q}}(\alpha_i, \gamma_i+\Delta\gamma)-\dot{\bar{q}}(\alpha_i, \gamma_i)}{\Delta\gamma} \tag{4.31}$$

$$\frac{\partial \dot{\gamma}}{\partial \alpha} \cong \frac{\dot{\gamma}(\alpha_i+\Delta\alpha, \gamma_i)-\dot{\gamma}(\alpha_i, \gamma_i)}{\Delta\alpha} \tag{4.32}$$

$$\frac{\partial \dot{\gamma}}{\partial \gamma} \cong \frac{\dot{\gamma}(\alpha_i, \gamma_i+\Delta\gamma)-\dot{\gamma}(\alpha_i, \gamma_i)}{\Delta\gamma} \tag{4.33}$$

where $\Delta\alpha$ is some small perturbation in angle of attack, such as 10^{-5} deg, $\Delta\gamma$ is some small perturbation in flight-path angle, such as 10^{-5} deg (1.745×10^{-7} rad; since $\dot{\gamma}$ is computed in rad/sec, the solved value of γ should be in rad), $\dot{\bar{q}}(\alpha_i, \gamma_i)$ is $\dot{\bar{q}}$ evaluated at the guesses for angle of attack and flight-path angle for the *i*th iteration (α_i and γ_i), and $\dot{\gamma}(\alpha_i, \gamma_i)$ is $\dot{\gamma}$ evaluated at the guesses for angle of attack and flight-path angle for the *i*th iteration.

A comparable drag valley to the one described in Section 4.2 can be generated that satisfies CDPQEG flight conditions over a wide range of energy states (i.e., coordinate pairs of V and h, which together define the current energy height of the vehicle). Each point in the contour or surface plot is found using the Newton-Raphson Method (with an analytical or finite-difference Jacobian) to calculate the values of the four flight variables (V, h, γ, and α) that satisfy CDPQEG flight conditions at that particular velocity and altitude. Then the drag corresponding to those states is plotted as a point in the contour or surface plot.

To test whether maximum-range trajectories tend to follow the CVQEG drag valley or that of the CDPQEG, a path that approximates the bottom of the drag valley can be found. This path consists of points along the drag surface corresponding to the minimum drag achievable with CDPQEG conditions at a given energy state. Drag is approximated at numerous V - h pairs along an energy contour by using bilinear interpolation (faster than recalculating drag every time via the Newton-Raphson Method). The V-h pair with minimum drag along the contour is flagged as a point along the bottom of the drag valley. After this process has been completed for several energy contours intersecting the drag valley, the flagged points are plotted as approximations of the bottom of the drag valley. Separate contour or surface plots can be produced to display the angle of attack, flight-path angle, and dynamic pressure corresponding to the CDPQEG conditions at each energy state.

As with the CVQEG drag valley, these plots consist of independently-solved sets of flight variables that satisfy CDPQEG conditions at each energy state and do not involve integrating the equations of motion throughout a trajectory, so there is no

guarantee that the drag valley is itself a physically flyable trajectory. It instead serves as a means of predicting the flight conditions (such as flight-path angle) toward which numerically optimized trajectories will tend. A control system could be designed, however, to direct the glider to maintain CDPQEG conditions as best as possible, as will be discussed in Section 4.4.

Figure 4.5 illustrates the CDPQEG drag valley as a colored contour plot, and selected contours of constant energy height are plotted over the drag valley. The approximate bottom of the drag valley, as found by minimizing drag along these contours of constant energy height, is also displayed in the figure. Figures 4.6-4.8 illustrate the flight-path angles, angles of attack, and dynamic pressures, respectively, that satisfy CDPQEG conditions (and, hence, that correspond to the CDPQEG drag) at each point along the drag valley.

Comparing the drag valleys for the two different approaches (CVQEG and CDPQEG) as seen in Figs. 4.1 and 4.5 reveals that they are largely identical over much of the V-h space, differing primarily in the area over which the drag valley is calculable. The CDPQEG drag valley covers more of the V-h space because the algorithm for finding CDPQEG states (i.e., the Newton-Raphson Method) converges for a larger variety of V-h pairs. This discrepancy in convergence between the two algorithms seems to account for the primary differences between their drag valleys. Apart from the convergence issue, the drag values for CVQEG and CDPQEG at a given V-h pair seem to be quite close.

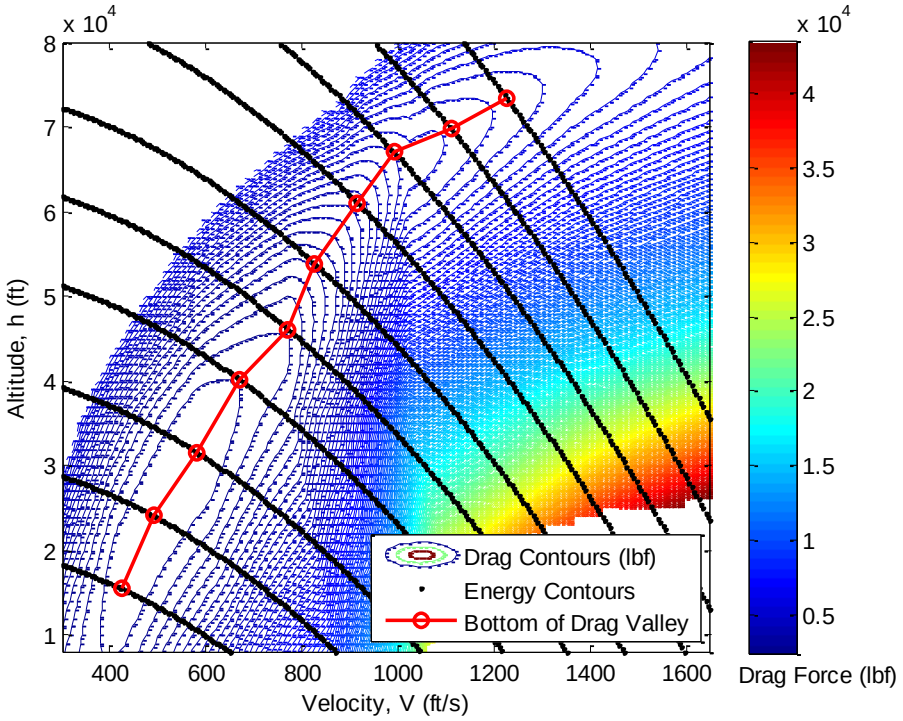

Figure 4.5. Drag corresponding to CDPQEG (i.e., drag valley) with selected contours of constant energy height (black) and minimum-drag points along those contours (red circles), approximating bottom of the drag valley.

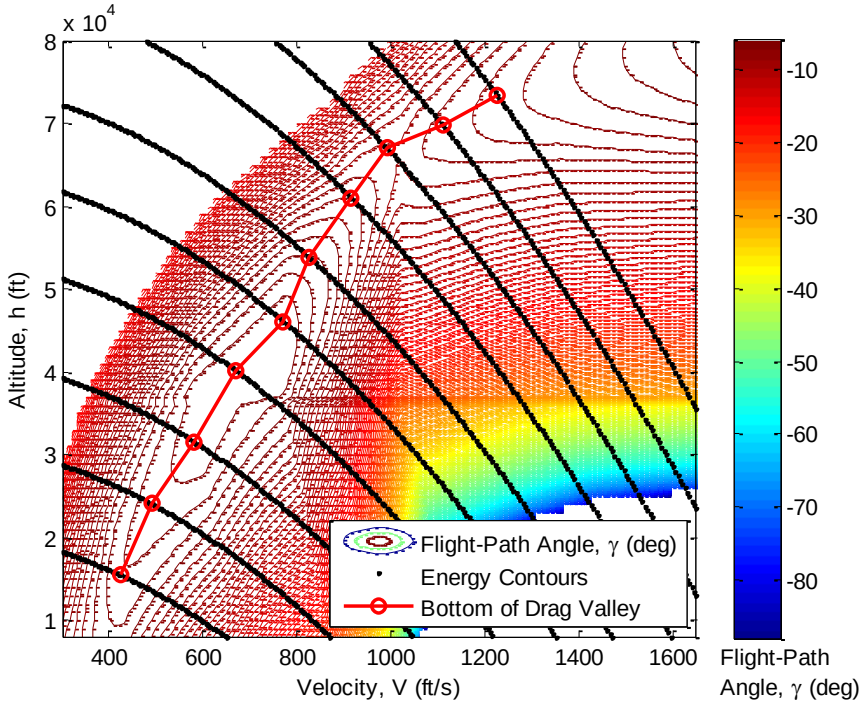

Figure 4.6. Flight-path angle corresponding to CDPQEG drag valley with selected contours of constant energy height (black) and minimum-drag points along those contours (red circles), approximating bottom of the drag valley.

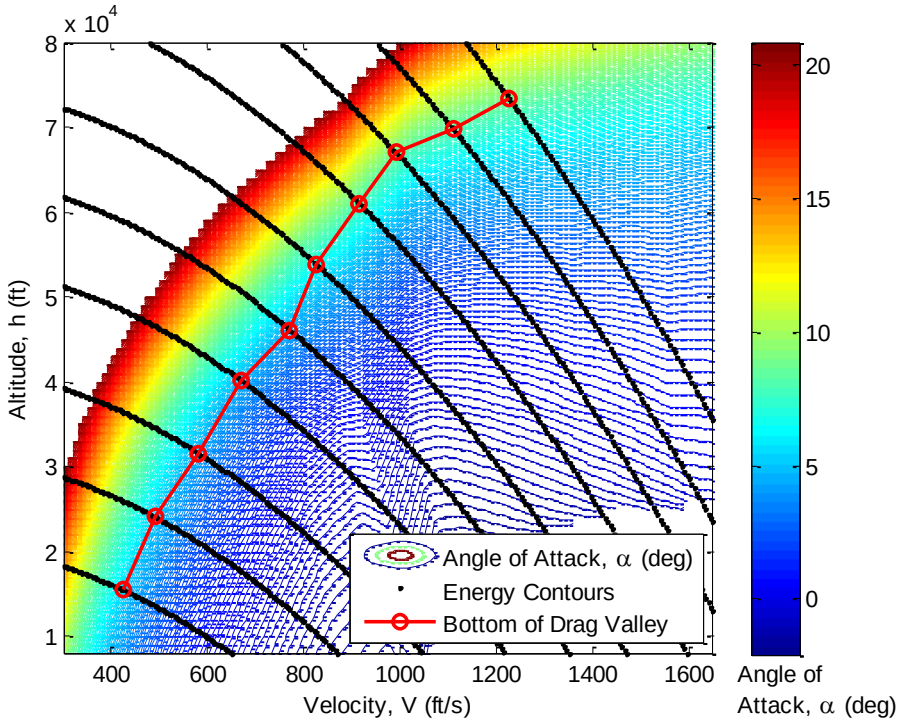

Figure 4.7. Angle of attack corresponding to CDPQEG drag valley with selected contours of constant energy height (black) and minimum-drag points along those contours (red circles), approximating bottom of the drag valley.

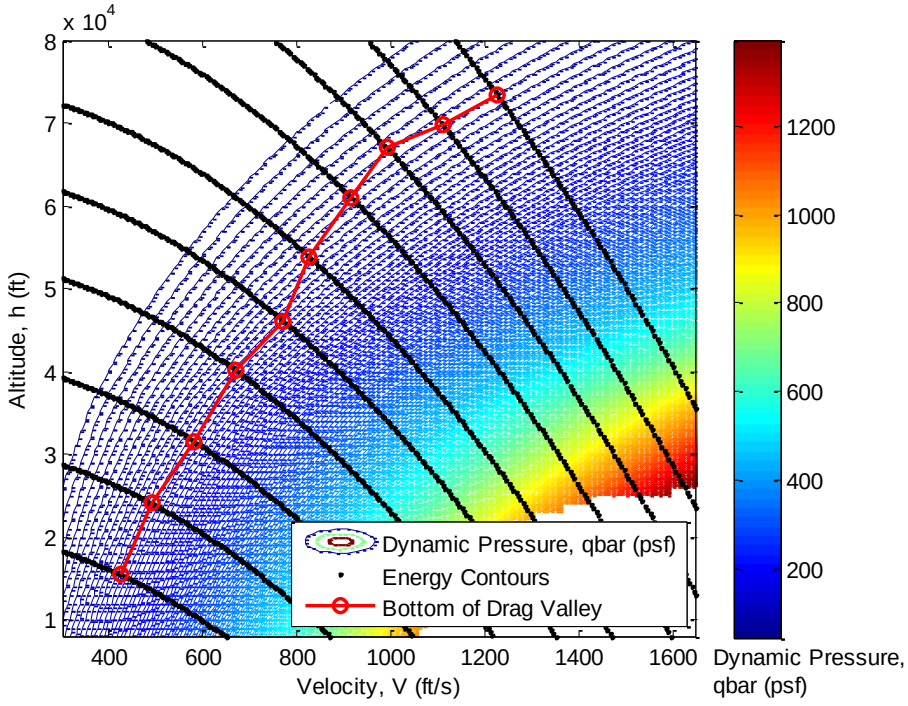

Figure 4.8. Dynamic pressure corresponding to CDPQEG drag valley with selected contours of constant energy height (black) and minimum-drag points along those contours (red circles), approximating bottom of the drag valley.

4.4. Control System for Maintaining Quasi-Equilibrium Glide Conditions

As mentioned in the previous sections, there is no guarantee that the path along the bottom of a drag valley will be a physically flyable trajectory, given that it arises not by integrating the equations of motion but by calculating independent sets of flight variables to satisfy the Quasi-Equilibrium Glide (QEG) conditions. Hence, if QEG conditions are to be maintained throughout a flyable trajectory, an onboard control system must be designed to enforce the conditions in real-time.

A possible system that would enforce these conditions would consist of a simple feedback controller that uses the current velocity and altitude to find a reference angle of attack and flight-path angle that would satisfy the conditions of the current control mode (CVQEG or CDPQEG), using the algorithms discussed in Sections 4.2 and 4.3. A perturbation signal could be produced by comparing the current flight-path angle γ with the reference flight-path angle γ_{QEG} (i.e., flight-path angle necessary for QEG), then passing the perturbation through a gain and adding it to the reference angle of attack α_{QEG} (i.e., angle of attack necessary for QEG) to produce a control input. Hence, the control input is

$$\alpha = \alpha_{QEG} - K_{\gamma}\left(\gamma - \gamma_{QEG}\right) \tag{4.34}$$

where K_{γ} is a gain on the error in the flight-path angle. Note that both α_{QEG} and γ_{QEG} are found by solving Eqs. (4.7)-(4.8) for CVQEG or Eqs. (4.23)-(4.24) for CDPQEG. Also note that if K_{γ} is positive and $\gamma > \gamma_{QEG}$, then the control input α is less than the QEG

reference angle of attack α_{QEG}, which will tend to turn the velocity vector toward the ground (and hence, the flight-path angle becomes more negative). If K_γ is positive and $\gamma < \gamma_{QEG}$, then the control input α will tend to turn the velocity vector up toward the sky (and, hence, flight-path angle becomes less negative or even becomes positive). This sign convention is consistent with the desire that the control system track the QEG reference flight-path angle γ_{QEG}.

The controller gain K_γ plays a critical role in determining what the final range of the vehicle is, so the selection of this value is especially important. A simple approach to selecting this gain is pole placement, in which the equation that governs flight-path angle (Eq. (2.2)) is linearized about the QEG reference conditions, and the gain is selected to place the pole of this equation in a location that results in the desired closed-loop damping ratio and natural frequency. Recall that the rate of change of flight-path angle with respect to time, according to Eq. (2.2), is

$$\dot{\gamma} = \frac{L}{mV} - \frac{g}{V}\cos\gamma \tag{4.35}$$

In order to linearize Eq. (4.35) about the QEG reference conditions, the following perturbation variables are defined for convenience:

$$\delta\hat{\gamma} = \gamma - \gamma_{QEG} \tag{4.36}$$

$$\delta\hat{\alpha} = \alpha - \alpha_{QEG} \tag{4.37}$$

Now the linearized form of Eq. (4.35) is

$$\delta\dot{\hat{\gamma}} = \frac{\partial\dot{\gamma}}{\partial\gamma}\delta\hat{\gamma} + \frac{\partial\dot{\gamma}}{\partial\alpha}\delta\hat{\alpha} \tag{4.38}$$

where the partial derivatives of $\dot{\gamma}$ with respect to γ and α are derived from Eq. (4.35) and the aerodynamic models. Therefore, the linearized open-loop system matrix (which has only one element for this system) is $\partial\dot{\gamma}/\partial\gamma$. Typically pole placement is conducted by adjusting the roots of the characteristic equation of this matrix to achieve asymptotic stability in the closed-loop system and the desired rate of convergence (or a desired damping ratio and natural frequency if there are two states). The characteristic equation of this system matrix, given that the matrix has only one element, is simply

$$\lambda_{OL} - \frac{\partial\dot{\gamma}}{\partial\gamma} = 0 \tag{4.39}$$

where λ_{OL} is the eigenvalue of the open-loop system. The root of this equation is clearly $\partial\dot{\gamma}/\partial\gamma$. It is desired that the system exhibit damping such that the system states tend to converge on the QEG reference conditions. Such asymptotic stability requires that the root of the characteristic equation have a negative real part. Given that $\partial\dot{\gamma}/\partial\gamma$ is a real value, it must be less than zero in order for the open-loop system to be stable. Both $\partial\dot{\gamma}/\partial\gamma$ and $\partial\dot{\gamma}/\partial\alpha$ for the linearized system (Eq. (4.38)) are discussed next.

While the control system is meant to apply to the piecewise polynomial aerodynamic model, it is difficult to evaluate $\partial\dot{\gamma}/\partial\alpha$ using the piecewise model, given that the piece of the model being used will vary throughout the trajectory as Mach number varies. If an inverse model were developed to find the necessary α for a given C_L, then the control could be defined to be C_L instead of α, and the inverse model could translate the desired value of C_L into a desired value for α. However, in the absence of such an inverse model (which would require some means of accommodating the multiple angles of attack that correspond to a single lift coefficient), the partial derivatives are found by assuming the constant drag-polar aerodynamic relationship $C_L = C_{L_0} + C_{L_\alpha}\alpha$ (Eq. (2.15)). Recalling the relationships of $L = \bar{q}SC_L$ and $\bar{q} = \frac{1}{2}\rho V^2$ (Eqs. (2.12) and (2.14)), the partial derivatives are found as

$$\frac{\partial\dot{\gamma}}{\partial\gamma} = \frac{g}{V}\sin\gamma \tag{4.40}$$

$$\frac{\partial\dot{\gamma}}{\partial\alpha} = \frac{\partial}{\partial\alpha}\left(\frac{\rho V^2 SC_L}{2mV} - \frac{g}{V}\cos\gamma\right) = \frac{\rho VS}{2m}\frac{\partial C_L}{\partial\alpha} = \frac{\rho VSC_{L_\alpha}}{2m} \tag{4.41}$$

From Eqs. (4.39)-(4.40) it can be seen that the sign of the open-loop eigenvalue changes as the sign of γ changes. Because γ is usually negative, λ_{OL} is usually negative, indicating that the open-loop system is stable, converging on QEG reference states when no feedback control is used. Therefore, if the control input α is simply equal to α_{QEG} throughout the trajectory, the open-loop system exhibits a natural tendency for γ to converge on γ_{QEG} for the current velocity and altitude. Whether the nonlinear open-loop

system actually converges to QEG states will be discussed later (see Figs. 4.9-4.16). Next the closed-loop characteristic equation is derived to find the closed-loop eigenvalue.

The control law, Eq. (4.34), can be rewritten in terms of $\delta\hat{\gamma}$ and $\delta\hat{\alpha}$ as

$$\delta\hat{\alpha} = -K_\gamma \delta\hat{\gamma} \tag{4.42}$$

Substituting this rewritten control law (Eq. (4.42)) into the linearized approximation of flight-path angle (Eq. (4.38)) provides the following relationship:

$$\delta\dot{\hat{\gamma}} = \frac{\partial\dot{\gamma}}{\partial\gamma}\delta\hat{\gamma} - \frac{\partial\dot{\gamma}}{\partial\alpha}K_\gamma\delta\hat{\gamma} = \left(\frac{\partial\dot{\gamma}}{\partial\gamma} - \frac{\partial\dot{\gamma}}{\partial\alpha}K_\gamma\right)\delta\hat{\gamma} \tag{4.43}$$

Now the linearized closed-loop system matrix (which contains only one element for this system) is $\left(\frac{\partial\dot{\gamma}}{\partial\gamma} - \frac{\partial\dot{\gamma}}{\partial\alpha}K_\gamma\right)$. Therefore, the characteristic equation for the linearized closed-loop control system is

$$\lambda_{CL} - \left(\frac{\partial\dot{\gamma}}{\partial\gamma} - \frac{\partial\dot{\gamma}}{\partial\alpha}K_\gamma\right) = 0 \tag{4.44}$$

where λ_{CL} is the eigenvalue of the closed-loop system. The root of the characteristic equation is $\left(\frac{\partial\dot{\gamma}}{\partial\gamma} - \frac{\partial\dot{\gamma}}{\partial\alpha}K_\gamma\right)$. From Eq. (4.44) the value of the gain can be found to be

$$K_\gamma = \frac{\frac{\partial\dot{\gamma}}{\partial\gamma} - \lambda_{CL}}{\frac{\partial\dot{\gamma}}{\partial\alpha}} \tag{4.45}$$

where λ_{CL} can be selected to provide the desired system response—i.e., asymptotic stability and the desired rate of convergence on QEG conditions. Note from Eqs. (4.40)-(4.41) that $\partial\dot{\gamma}/\partial\gamma$ and $\partial\dot{\gamma}/\partial\alpha$ will vary with the states V, γ, and h (atmospheric density ρ varies with h according to the exponential model (Eq. (2.26)) or the piecewise thermal model displayed in Fig. 2.3). Therefore, for a desired value of λ_{CL}, the value of K_γ changes as V, γ, and h change. Traditionally pole placement is used to find a constant gain value to employ in the closed-loop control system. In this case, however, a control system could be implemented that modifies the value of K_γ in real time to satisfy Eq. (4.45) and accommodate the changes in the states, enabling the closed-loop system to avoid becoming unstable as the states change.

The partial derivatives must be evaluated at the QEG reference flight-path angle γ_{QEG} and angle of attack α_{QEG}, since the linearization is made about these QEG reference states. Furthermore, α is only computed (according to Eq. (4.34)) once the value of gain K_γ is known. As already discussed, the constant drag-polar model is used to evaluate the partial derivatives, so there must be some nominal drag-polar parameters selected for the model. The nominal parameters are given in Table 4.1 and are selected to provide a constant-drag-polar fit (Eqs. (2.15)-(2.16)) of the X-34 aerodynamics at Mach 0.6, giving a good approximation of typical subsonic flight. Recall that the piecewise-polynomial aerodynamic model is still used to model the flight of the vehicle, but the constant drag-polar is used to find the value of the controller gain (Eq. (4.45)).

Table 4.1. Nominal parameters for constant drag-polar aerodynamic model, selected to model X-34 aerodynamics at Mach 0.6.

Parameter	Value
Zero-Angle Lift Coefficient, C_{L_0}	0.12502
Lift-Slope Coefficient, C_{L_α}	0.051718
Zero-Lift Drag Coefficient, C_{D_0}	0.021348
Induced Drag Coefficient, K	0.26647

The two QEG approaches are compared to the max-*L/D* trajectory, all of which use the same initial velocity and altitude and the same final energy height as the numerical optimization trials in Section 3.2 (see Table 3.1 for initial and final conditions of the numerical optimization trials). These trials also share maximum integration step size (1,000 ft of energy height) with the numerical optimization trials, and they use the piecewise aerodynamic model (Eqs. (2.23)-(2.24) and Fig. 2.1) and the piecewise thermal atmospheric model (Figs. 2.3-2.5). The primary differences from the numerical optimization trials in the initial conditions of these QEG trials are the initial flight-path angles.

The CVQEG and CDPQEG trials select initial flight-path angles γ_0 by solving their respective QEG conditions for γ_{QEG} and α_{QEG} at the initial velocity and altitude. The two different QEG approaches result in substantially different values of γ_0, as seen in Table 4.2, which displays the initial and final conditions of the QEG trials. These values of γ_0 are displayed in Table 4.2 as approximate values because the QEG algorithms use much greater numerical precision than is displayed in the table. Furthermore, it should be noted that the values in the table are given in deg for

convenience, but the actual values of γ should be in rad in order for $d\gamma/de$ to be integrated (Eq. (2.8)).

The rationale for letting each QEG algorithm select the values of γ_0 by solving the QEG conditions for γ_{QEG} at the initial velocity and altitude is to give each approach the best initial condition for it to succeed. That is, by starting each QEG control system at QEG conditions, the controller should have as little trouble as possible tracking the QEG reference flight-path angle. The max-*L/D* trajectory uses the same γ_0 as the CDPQEG trial which, it may be noted, is also the γ_0 selected for the numerical optimization trials in Section 3.2 (see Table 3.1). Hence, the question arises of how γ_0 affects the max-*L/D* trajectory. This question will be addressed in the discussion of flight-path angle along the trajectories that use the QEG controller.

Table 4.2. Initial and final conditions for trials using QEG control systems and the max-*L/D* trajectory used for comparison.

Trajectory Boundary	Velocity V (ft/s)	Altitude h (ft)	Flight-path Angle γ (deg)	Energy Height e (ft)
Max-*L/D* Initial States	1500	70,000	-7.56	104,966.1
CVQEG Initial States	1500	70,000	~ -20.3	104,966.1
CDPQEG Initial States	1500	70,000	~ -7.56	104,966.1
Final States for All Trials	--	--	--	14,514.8

Simulations are conducted using the control system for each QEG approach and with various gain structures. To begin, trajectories are simulated by setting the controller gain K_γ equal to zero throughout the trajectories. These trials, illustrated in Figs. 4.9-

4.16, serve to simulate the open-loop system and provide a baseline from which to measure the benefit of the feedback control system.

The CVQEG velocity profile of Fig. 4.9 shows that simply flying at the CVQEG reference angle of attack at the beginning of the trajectory (right side of the plot, where energy height is greatest) is close enough to satisfying the CVQEG conditions that velocity remains largely constant at first. As energy is lost and no feedback control is enabled (i.e., $K_\gamma = 0$), velocity begins to decrease. Similarly, in Fig. 4.10, flying at the CDPQEG reference angle of attack results in a nearly constant dynamic pressure at first. The CDPQEG trial in Figs. 4.9-4.10 seems to follow the max-*L/D* trajectory much more closely than the CVQEG trial. This tendency strengthens with certain values of K_γ. Note also how neither the max-*L/D* nor the CDPQEG trajectory tends to vary as severely in dynamic pressure as the CVQEG trajectory.

Figures 4.11-4.13 all corroborate a similarity between the max-*L/D* and CDPQEG approaches. In particular, the CDPQEG approach appears to oscillate less than the max-*L/D* approach, cutting off the initial lob of positive flight-path angle that occurs with max *L/D* (and in most of the numerical trials of Section 3.2 to a greater or lesser degree) and avoiding its resulting oscillations. Instead, the CDPQEG approach nearly follows the center of the max-*L/D* oscillations in flight-path angle (Fig. 4.11), while also following nearly the same altitude profile (Fig. 4.12) as max-*L/D* (minus the initial lob as mentioned). This approach, which might intuitively seem more energy efficient, actually costs the CDPQEG in final range, as shown in Fig. 4.13. The CVQEG approach, in an attempt to maintain a constant velocity, turns downward with steeper and steeper flight-path angle (Fig. 4.11), achieving a lower altitude (Fig. 4.12) and a much worse final

range (Fig. 4.13). This behavior is a consequence of CVQEG tending toward more negative angles of attack, as shown in Fig. 4.14.

Here it is noted that γ_0 for the max-*L/D* trajectory was selected to be approximately that of the CDPQEG because of this tendency of the max-*L/D* trajectory to oscillate about the CDPQEG flight-path angle. This phenomenon is not unique to the CDPQEG value of γ_0, but this oscillation also occurs given a γ_0 near that of the CVQEG trajectory. Using the CVQEG γ_0, however, increases the amplitude of oscillation of the max-*L/D* flight-path angle about the CDPQEG flight-path angle, given the greater difference between the two values at the initial condition. Furthermore, using the CVQEG γ_0 significantly reduces the range achieved by max-*L/D* because the large oscillations reduce the velocity of the vehicle (particular the horizontal component of velocity) over much of the trajectory. To maximize the range using max-*L/D* requires a large positive value of γ_0, as mentioned in Section 3.2 with the initial conditions for the numerical optimization trials. Due to the unrealistic nature of a large positive γ_0 as an initial condition for the TAEM phase, the max-*L/D* γ_0 is selected to approximate the γ_0 used for CDPQEG (i.e., γ_0 that satisfies CDPQEG for the initial velocity and altitude).

While CVQEG tends toward more negative angles of attack, as shown in Fig. 4.14, the max-*L/D* trajectory uses large positive angles of attack, especially at first, and CDPQEG uses a shape of control profile similar to that of max-*L/D* but with smaller magnitude. Notice how each control profile has a hump. These humps seem to occur at transonic velocities (near 1,000 ft/s) in each trajectory, likely due to the severe changes in aerodynamics of the vehicle at such velocities. See Fig. 4.9 to compare velocities at the energy heights corresponding to these humps.

Figure 4.15 illustrates the open-loop eigenvalues along the trajectory using each QEG approach. In each case the eigenvalues have very small negative magnitudes, generally increasing in magnitude as energy height decreases. Hence, the open-loop system remains stable throughout these trajectories. For comparison of the scale of these eigenvalues to the times of flight of these trajectories, it is noted that the CVQEG trajectory lasted 118.5 s, the CDPQEG trajectory lasted 619.1 s, and the max-*L/D* trajectory lasted 782.9 s. The magnitudes of these eigenvalues will assist in selecting useful closed-loop eigenvalues for the feedback control system.

Figure 4.16 shows the error in flight-path angle throughout the trajectories for each approach. The CDPQEG error tends to vary about zero, while the CVQEG error grows quite large (over 25 deg) before returning to smaller magnitudes of about 5 deg. Note that the sudden decrease in CVQEG error also occurs at transonic velocities (near 1,000 ft/s), as can be seen by comparing the energy height at the decrease in Fig. 4.16 (about 45,000 ft) to the energy height at a velocity of 1,000 ft/s in Fig. 4.9. The large CVQEG error prior to this point seems to indicate that an open-loop CVQEG control input does not result in fast convergence to CVQEG conditions at supersonic velocities. While the linearized system with each approach (particularly CDPQEG) exhibits some tendency for error to decrease (which is expected with the negative open-loop eigenvalues of Fig. 4.15), the error does not decay exponentially as in a linear system, due to the complexities of the system that were ignored by linearization (Eq. (4.38)).

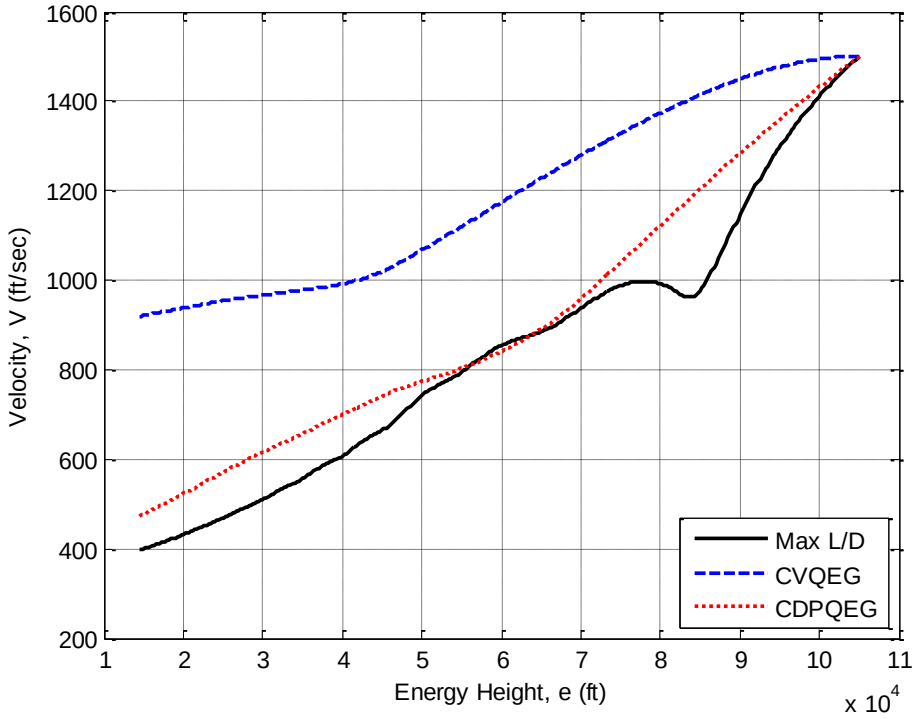

Figure 4.9. Velocity vs. energy height with CVQEG and CDPQEG control systems having $K_\gamma = 0$, compared to max-*L/D* trajectory.

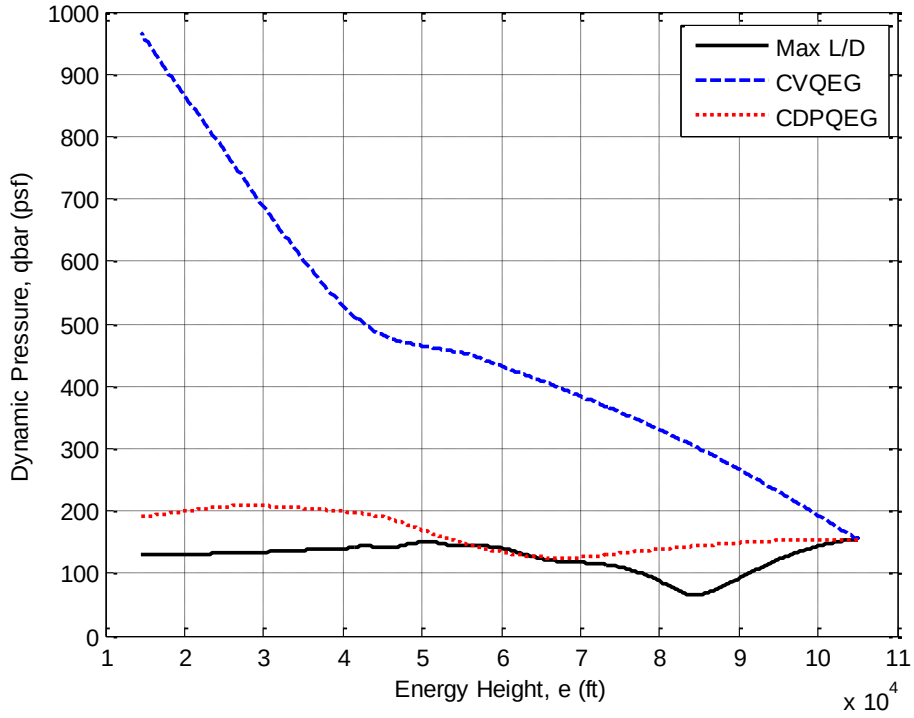

Figure 4.10. Dynamic pressure vs. energy height with CVQEG and CDPQEG control systems having $K_\gamma = 0$, compared to max-*L/D* trajectory.

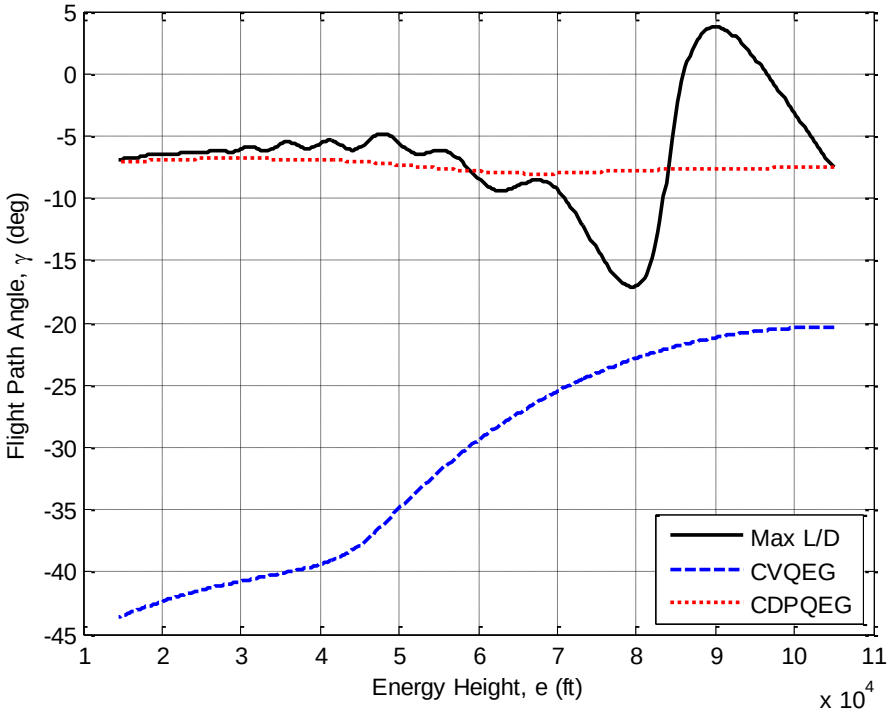

Figure 4.11. Flight-path angle vs. energy height with CVQEG and CDPQEG control systems having $K_\gamma = 0$, compared to max-*L/D* trajectory.

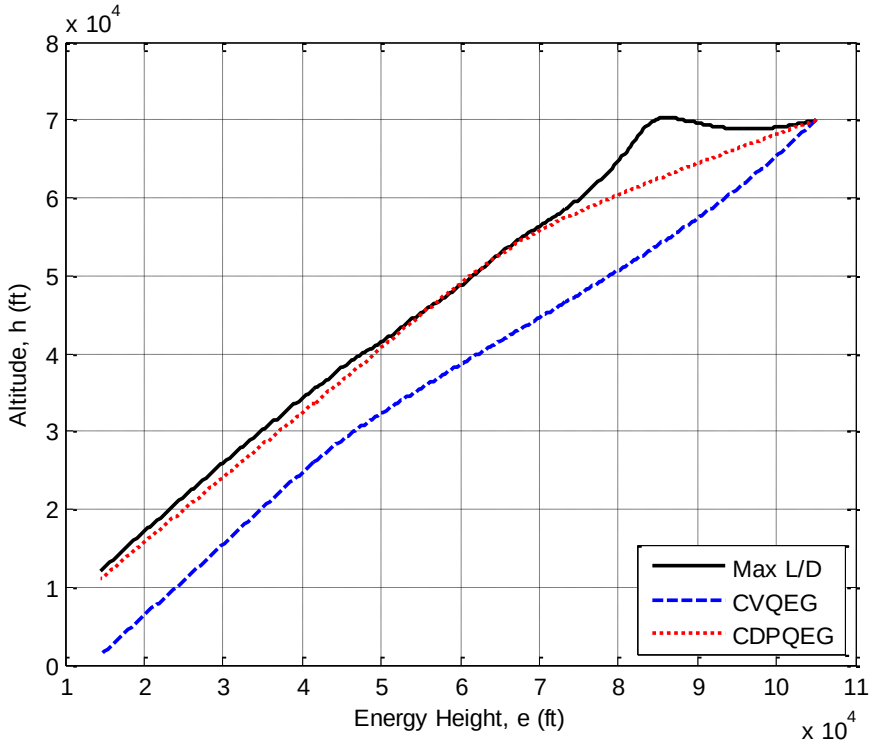

Figure 4.12. Altitude vs. energy height with CVQEG and CDPQEG control systems having $K_\gamma = 0$, compared to max-*L/D* trajectory.

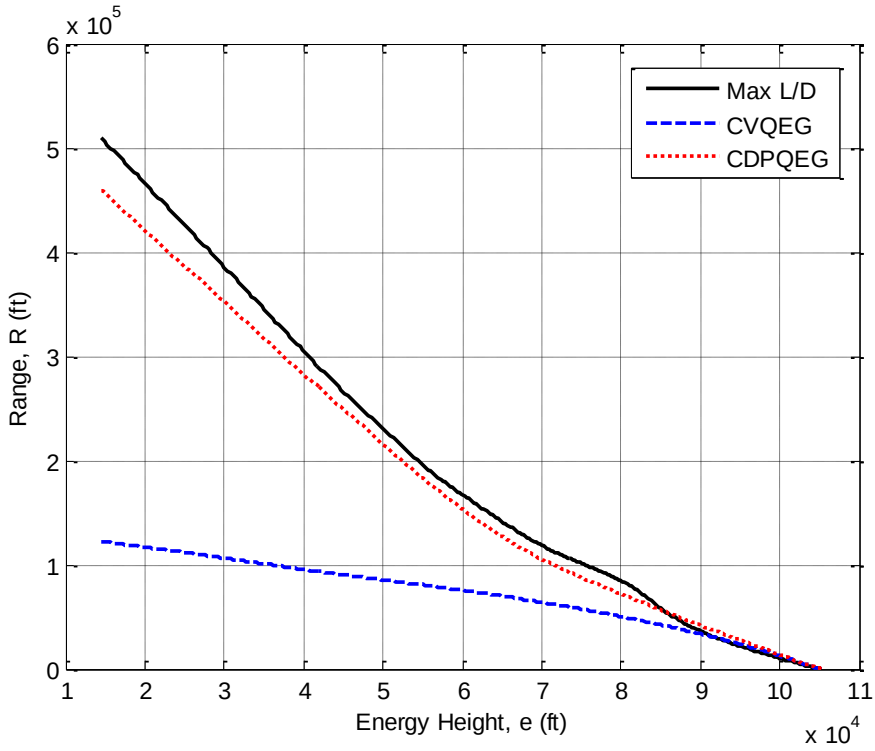

Figure 4.13. Range vs. energy height with CVQEG and CDPQEG control systems having $K_\gamma = 0$, compared to max-*L/D* trajectory.

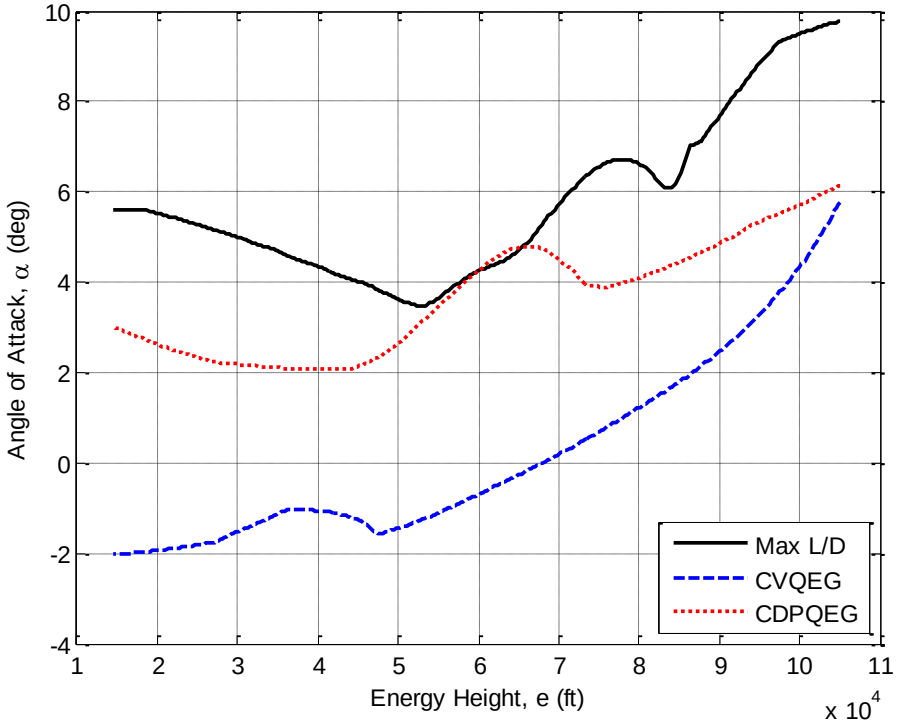

Figure 4.14. Angle of attack vs. energy height with CVQEG and CDPQEG control systems having $K_\gamma = 0$, compared to max-*L/D* trajectory.

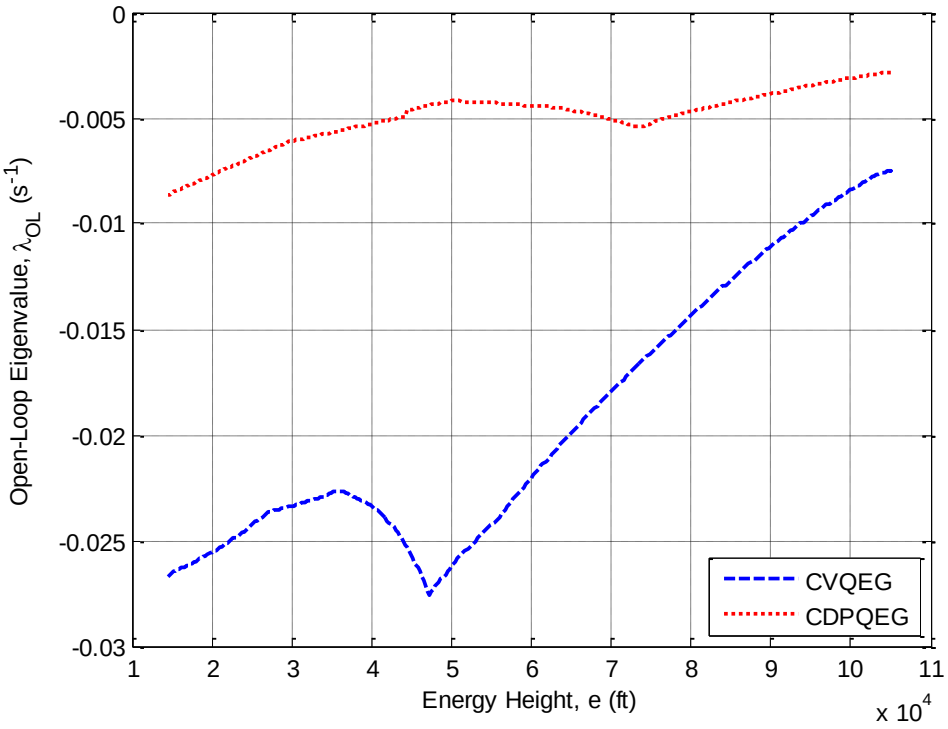

Figure 4.15. Open-loop eigenvalue vs. energy height with CVQEG and CDPQEG control systems having $K_\gamma = 0$.

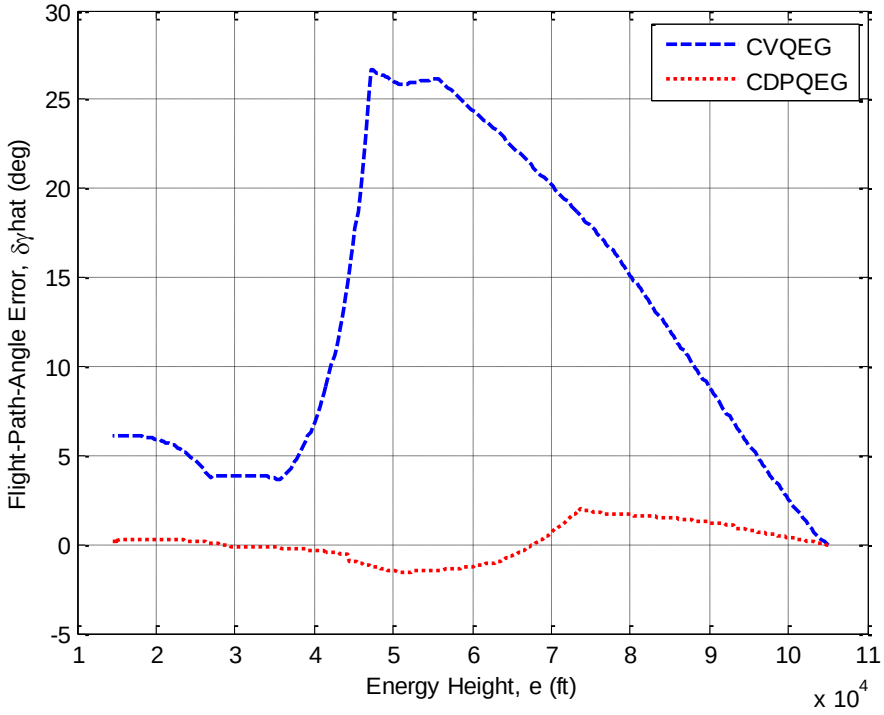

Figure 4.16. Flight-path-angle error vs. energy height with CVQEG and CDPQEG control systems having $K_\gamma = 0$.

At this point the effect of inserting a nonzero controller gain on range is considered. The method of pole placement described in Eq. (4.45), by which K_γ is adjusted in real-time as the partial derivatives in Eq. (4.45) change, requires some selected value of λ_{CL}. For the closed-loop system to be stable, it is desired that λ_{CL} be negative. Considering the magnitude of open-loop eigenvalues of both approaches, as shown in Fig. 4.15, it was decided to set the closed-loop eigenvalue λ_{CL} to -0.05 s^{-1}. This value should multiply the rate of convergence of the linearized CVQEG system about six times at the beginning of the trajectory and about two times at the end, by comparing λ_{CL} to the values of λ_{OL} shown in Fig. 4.15. This value of λ_{CL} should also multiply the CDPQEG rate of convergence by about twenty times at the beginning and about six times at the end of the trajectory. By selecting this closed-loop eigenvalue and updating K_γ in real-time throughout the trajectory according to Eq. (4.45), the CVQEG and CDPQEG approaches resulted in the trajectories displayed in Figs. 4.17-4.24, which are again compared to the max-*L/D* trajectory.

The velocity curves in Fig. 4.17 are very similar to those in Fig. 4.9 (where controller gain K_γ is zero), though the CVQEG approach is somewhat more successful at maintaining a constant velocity. The CDPQEG and max-*L/D* approaches exhibit even greater similarity than in Fig. 4.9. In Fig. 4.18 the CDPQEG dynamic pressure becomes much steadier than in Fig. 4.10 (with zero gain), as desired, while the CVQEG dynamic pressure behaves even more erratically. Of particular interest is how the CDPQEG flight-path angle in Fig. 4.19 follows the center of the max-*L/D* flight-path angle oscillations even more closely than in Fig. 4.11 (zero gain), while again, the CVQEG flight-path angle is much more erratic than before (much more negative and fluctuates more

violently). The CDPQEG altitude follows the max-*L/D* altitude (Fig. 4.20) even more closely than before (Fig. 4.12), but the CDPQEG ranges look fairly similar (compare Fig. 4.21 to Fig. 13). It seems that the max-*L/D* lob buys the max-*L/D* approach extra range, outstripping CDPQEG early on, and the CDPQEG approach misses that extra range by using a steadier flight-path angle. The CVQEG altitude drops faster than it did in the open-loop simulation, and the closed-loop CVQEG range is much lower than the open-loop range.

Figure 4.22 offers what may be the most critical information of any of these figures—the control profiles (angle of attack vs. energy height). The spikes in the QEG control profiles seem to be results of the high value of λ_{CL} and the correspondingly high values of K_γ (see Fig. 4.23). High gains can over-amplify any "noise" in the system which, in this case, rather than being physically-generated noise, is most likely due to roughness of the numerical models (e.g., aerodynamic or atmospheric) or perhaps variation in the convergence point of the Newton-Raphson Method used to solve QEG conditions. Of particular interest is what happens in the CVQEG angle-of-attack profile. A limiter is built into the controller to keep it from giving a control input (angle of attack) less than -6 deg or greater than 21 deg, which are the limits of the piecewise aerodynamic model (because they are the limits of the available aerodynamic data at the Mach numbers included in the model). The control input saturates at -6 deg in Fig. 4.22, but then it suddenly jumps to 0 deg. The controller is designed to default to 0 deg in the event that no solution is found to the QEG conditions. Hence, in Fig. 4.22 the vehicle reaches a velocity-altitude (*V*-*h*) pair for which the CVQEG algorithm does not converge on a solution (indicated by the large white spaces beside the drag valley in Figs. 4.1-4.4).

Note that the default control input could be set to maintain the last control input found, but defaulting to zero has the advantage of revising the last valid control input, when that input was likely unprofitable if it led to a *V*-*h* pair with no known QEG solution. Hence, as in Fig. 4.22, there is a greater chance of returning to a *V*-*h* pair for which there is a solution. After a brief period of saturation, the CVQEG control input in Fig. 4.22 returns to an unsaturated series of values.

A similar period of non-convergence to that of the CVQEG angle-of-attack profile is also visible in the CVQEG gain profile of Fig. 4.23, although it seems to begin and end at lower energy heights than that of the angle-of-attack profile in Fig. 4.22. This blank in the gain profile indicates that the evaluation of K_γ according to Eq. (4.45) failed, most likely because some value required for the calculation was found to be `NaN`, which is MATLAB's notation for "Not A Number." In particular, the function used to solve the QEG conditions with the Newton-Raphson Method returns `NaN` in the event that it does not converge on a solution. Because the partial derivatives in Eq. (4.45) are evaluated at α_{QEG}, if no α_{QEG} is found, then the gain K_γ cannot be calculated. This non-convergence period is also visible in the CVQEG flight-path-angle error in Fig. 4.24, given that $\delta\hat{\gamma}$ cannot be evaluated (Eq. (4.36)) if γ_{QEG} is `NaN`.

Far from eliminating the CVQEG error, the feedback controller seems to have amplified it (compare Figs. 4.16 and 4.24). The CDPQEG error, however, seems to be almost entirely removed by the feedback control system (again compare Figs. 4.16 and 4.24). This is corroborated by the virtually constant dynamic pressure throughout the CDPQEG approach in Fig. 4.18, indicating that CDPQEG conditions are being maintained successfully throughout the flight.

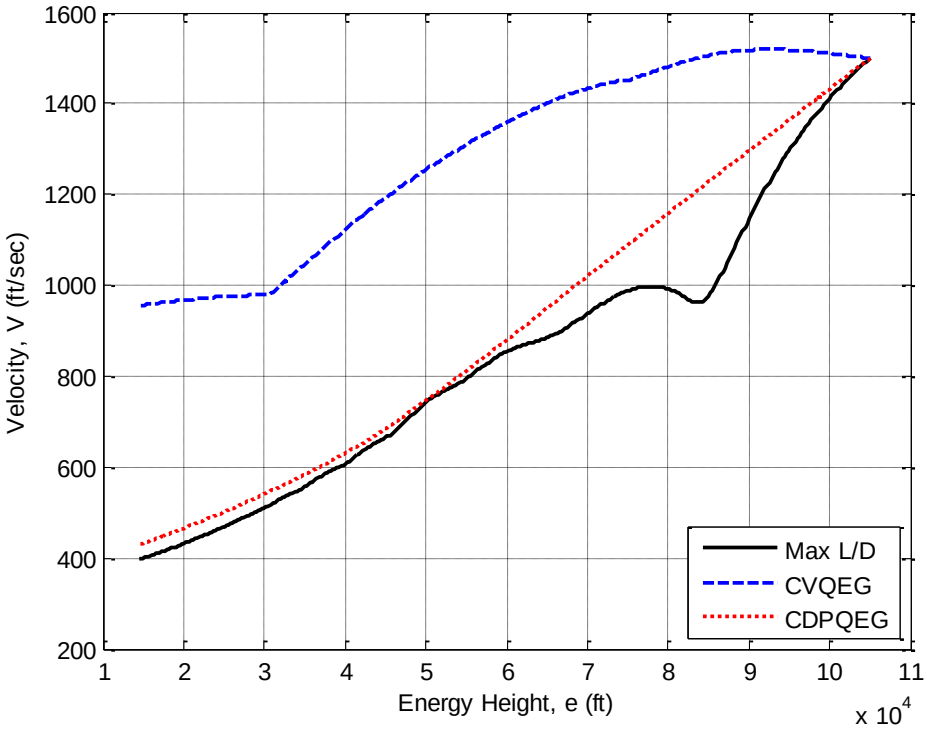

Figure 4.17. Velocity vs. energy height with each QEG control system updating K_γ in real time using Eq. (4.45) and $\lambda_{CL} = -0.05$ s^{-1}, compared to max-*L/D* trajectory.

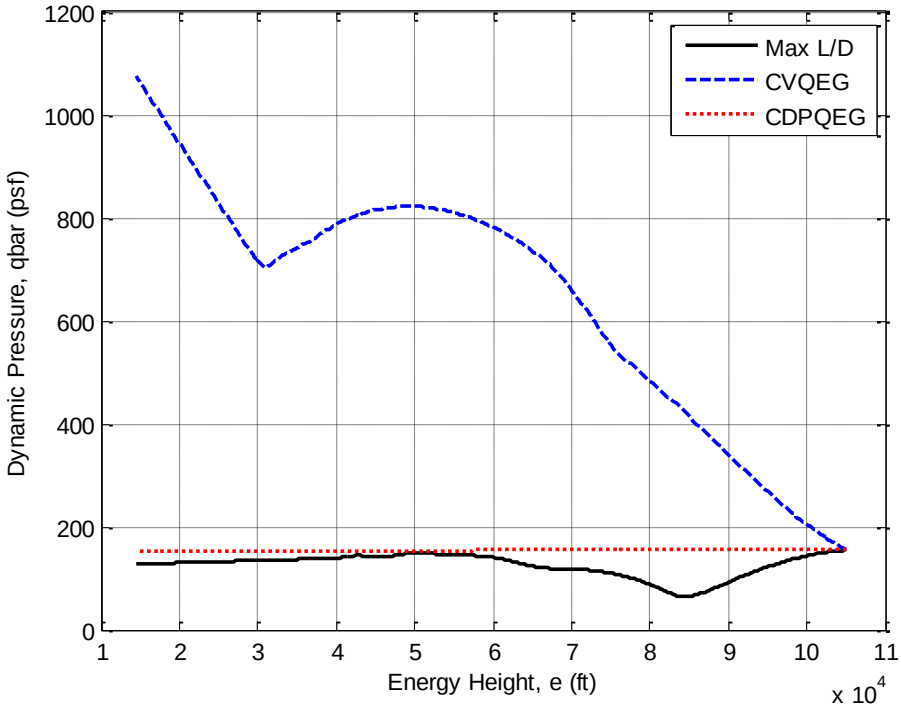

Figure 4.18. Dynamic pressure vs. energy height with each QEG control system updating K_γ in real time using Eq. (4.45) and $\lambda_{CL} = -0.05$ s^{-1}, compared to max-*L/D* trajectory.

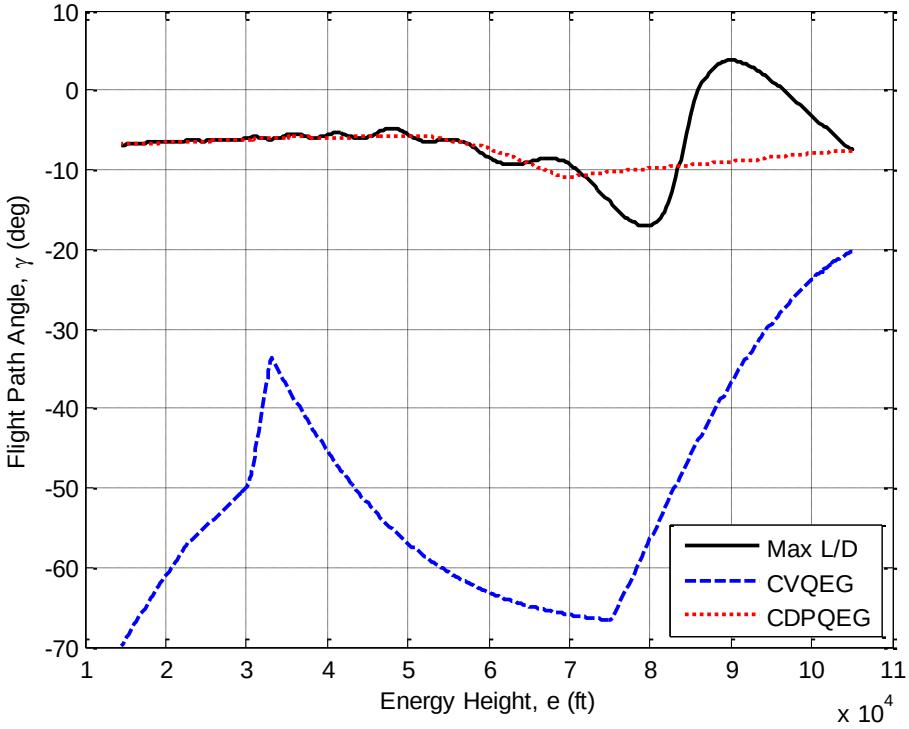

Figure 4.19. Flight-path angle vs. energy height with each QEG control system updating K_γ in real time using Eq. (4.45) and $\lambda_{CL} = -0.05$ s^{-1}, compared to max-*L/D* trajectory.

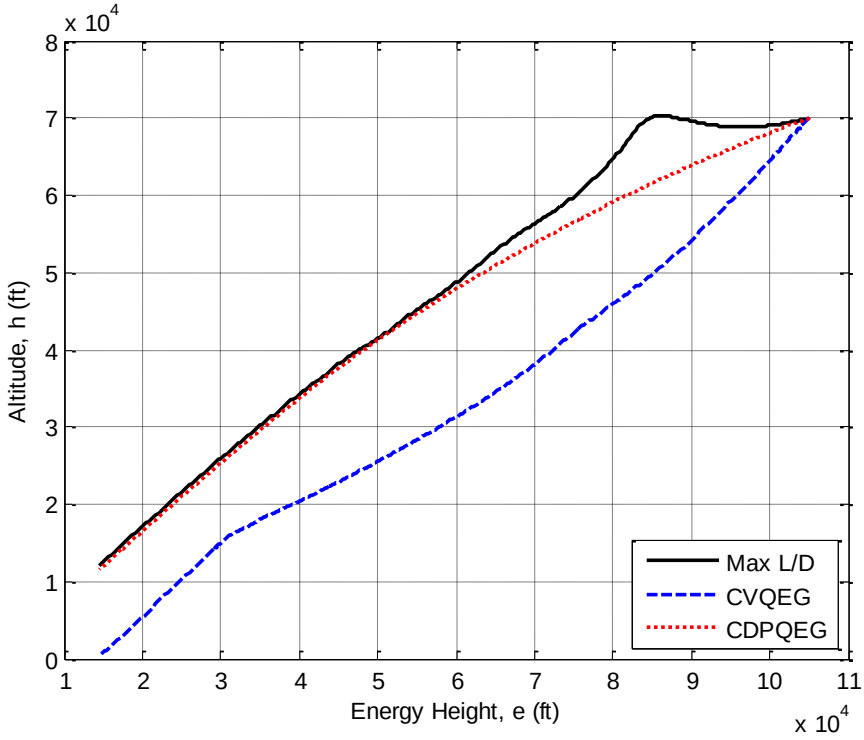

Figure 4.20. Altitude vs. energy height with each QEG control system updating K_γ in real time using Eq. (4.45) and $\lambda_{CL} = -0.05$ s^{-1}, compared to max-*L/D* trajectory.

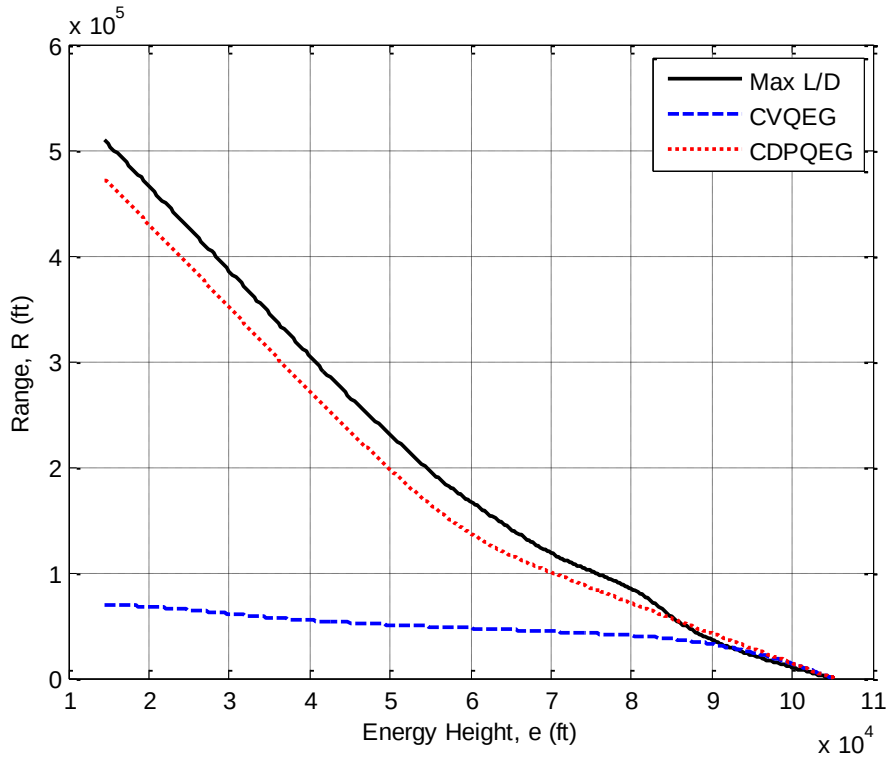

Figure 4.21. Range vs. energy height with each QEG control system updating K_γ in real time using Eq. (4.45) and $\lambda_{CL} = -0.05$ s^{-1}, compared to max-*L/D* trajectory.

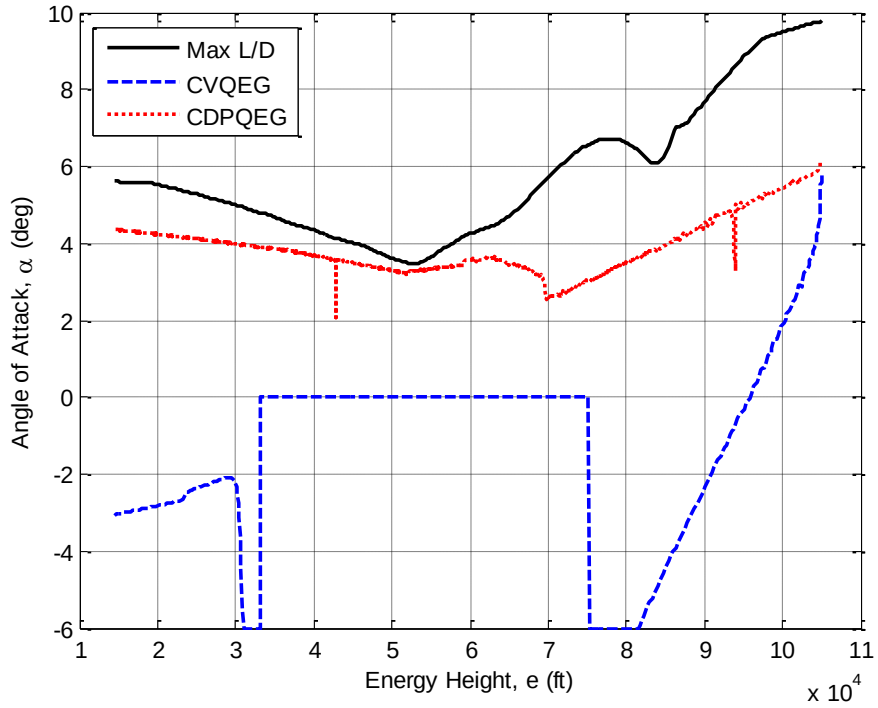

Figure 4.22. Angle of attack vs. energy height with each QEG control system updating K_γ in real time using Eq. (4.45) and $\lambda_{CL} = -0.05$ s^{-1}, compared to max-*L/D* trajectory.

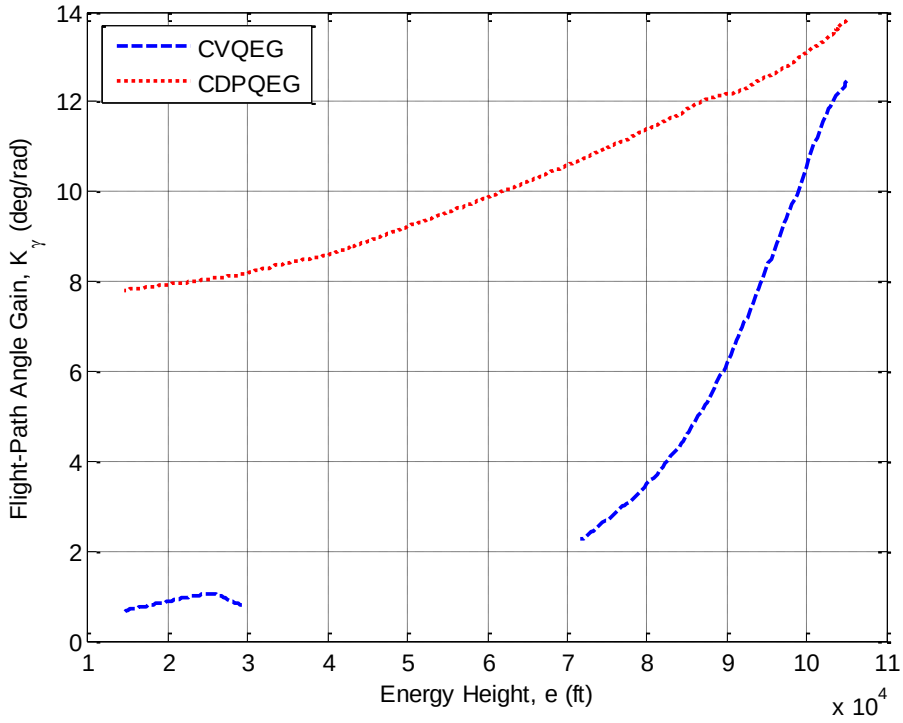

Figure 4.23. Flight-path-angle gain (K_γ) vs. energy height with each QEG control system updating K_γ in real time using Eq. (4.45) and $\lambda_{CL} = -0.05$ s^{-1}.

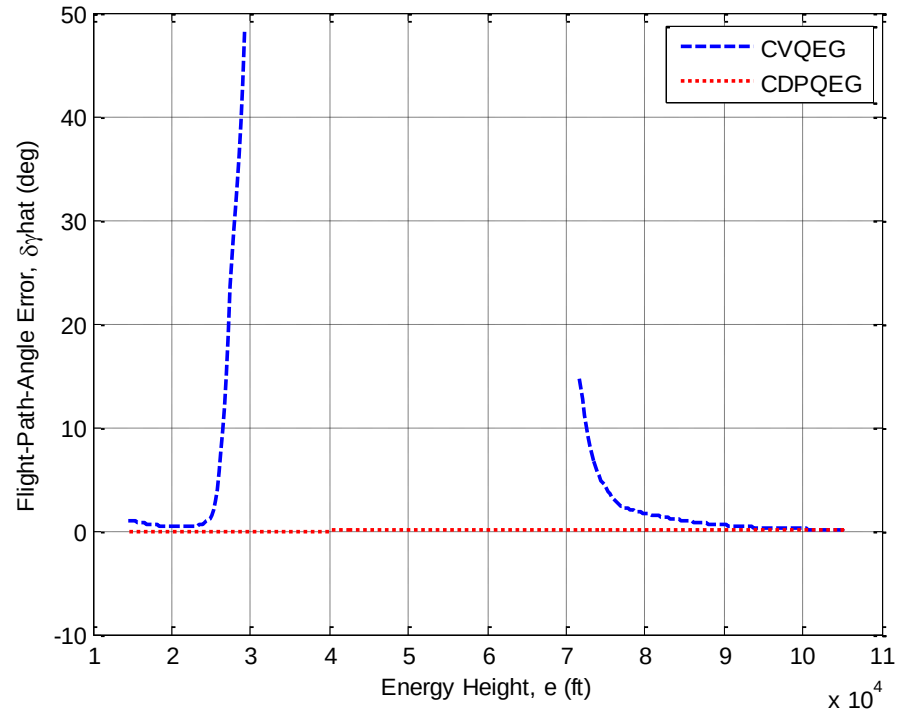

Figure 4.24. Flight-path-angle error vs. energy height with each QEG control system updating K_γ in real time using Eq. (4.45) and $\lambda_{CL} = -0.05$ s^{-1}.

Given that the choice of λ_{CL} plays a critical role in determining the gains throughout the trajectory and the corresponding range achieved, a variety of values of λ_{CL} were tested to observe the effect of λ_{CL} on the final range. Figure 4.25 displays the ranges achieved with both QEG approaches using various values of λ_{CL}, along with the open-loop (zero-gain) QEG approaches (from Figs. 4.9-4.16) and the max-*L/D* approach for comparison.

From Fig. 4.25 it appears that most of these values of λ_{CL} result in lesser final ranges for the closed-loop CVQEG approach than with the open-loop CVQEG approach. The exception is with $\lambda_{CL} = -0.01$ s^{-1}, for which the final CVQEG range is greater, but this seems to be the result of an especially large spike in flight-path angle (up to about 15 deg) that lobs the vehicle farther than it would otherwise go. This approach still falls short of the CDPQEG or max-*L/D* approaches. Reducing the magnitude of λ_{CL} below this point seems to result in widely fluctuating CVQEG trajectories that do not improve on range.

It appears that the ranges achieved with the closed-loop CDPQEG approach are slightly greater than with the open-loop CDPQEG approach, and these ranges remain largely constant over the values of λ_{CL} tested here. The reason why the range seems unaffected by λ_{CL} may be that these values of λ_{CL} are all significantly greater in magnitude than the open-loop eigenvalues of the CDPQEG approach (see Fig. 4.15), of which the greatest in magnitude was about -0.008. Hence, increasing the magnitude of λ_{CL} may not accelerate convergence to CDPQEG conditions by much, if convergence is already occurring quickly in each of these trials.

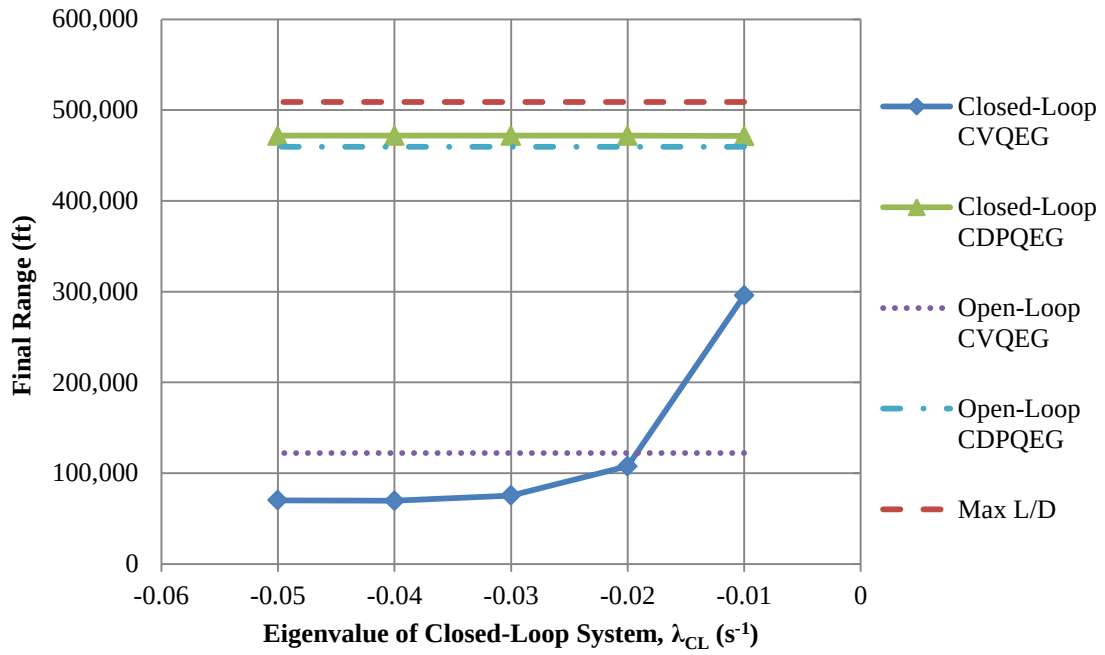

Figure 4.25. Final range achieved vs. eigenvalue of closed-loop system (λ_{CL}) with CVQEG and CDPQEG control systems updating K_γ in real time using Eq. (4.45), compared to final ranges of open-loop CVQEG and CDPQEG and max-*L/D* approaches.

Note that none of these trajectories succeeded in achieving the same range as the max-*L/D* approach. This seems to indicate that QEG conditions alone are not sufficient to maximize range. Perhaps a more sophisticated approach, such as following the bottom of the drag valley, would yield better results. This approach will be briefly discussed in Section 4.5. Even the best QEG trial shown in Fig. 4.25, the $\lambda_{CL} = -0.03$ s^{-1} CDPQEG trial, only resulted in a final range of 471,793.2 ft, which is still 7.31 % below the max-*L/D* trial of 508,991.2 ft.

While the QEG trials may not have achieved the same range as the max-*L/D* approach, occasionally they required less computational time than max-*L/D*. For example, the open-loop trials and the $\lambda_{CL} = -0.01$ s^{-1} trials required about 0.2 s computational time instead of the 0.395 s for the max-*L/D* approach. Nevertheless, the

trials with a greater magnitude of λ_{CL} required more computational time, such as the $\lambda_{CL} = -0.05$ s^{-1} CDPQEG trial, which required about 1.0 s.

4.5. Control System to Follow the Bottom of the Drag Valley

Note that the controller described in Section 4.4 (Eq. (4.34)) does not necessarily follow the bottom of the drag valley discussed in previous sections. Instead, this controller only attempts to maintain QEG conditions. Therefore, another approach to maximizing range might be to develop a controller that follows the bottom of the drag valley. As noted previously, the bottom of the drag valley is not guaranteed to be a flyable trajectory and therefore, would likely require a controller in order for a vehicle to approximate the path it outlines in the energy space. Such a control system would need to drive the vehicle toward a velocity, altitude, and flight-path angle that are consistent with the bottom of the drag valley. An investigation is needed to determine how to drive the vehicle to these states most efficiently. If the bottom of the drag valley is, in fact, a good approximation of the maximum-range trajectory, then what is the optimal path from an arbitrary initial condition to the drag valley bottom? The behavior of numerically optimized trajectories in approaching the drag valley bottom may lend insights to this question.

Figures 4.26-4.27 illustrate the tendency of numerically optimized trajectories to converge upon the bottom of the drag valleys. Figure 4.26 displays numerically optimized 17-node trajectories for three different initial conditions converging on the bottom of the CVQEG valley, and Fig. 4.27 displays the same trajectories converging on the bottom of the CDPQEG valley. The initial conditions for the three trajectories are

displayed in Table 4.3. They are selected to have the same initial energy height, where Case 1 is the nominal initial condition for TAEM used in Section 3.2 (see Table 3.1).

Table 4.3. Three different initial conditions for numerically optimized trajectories that converge on drag valleys.

Case	Velocity V (ft/s)	Altitude h (ft)	Flight-path Angle γ (deg)	Energy Height e (ft)
1	1500	70,000	-7.56	104,966.1
2	1600	65,182.4	-7.56	104,966.1
3	1400	74,506.7	-7.56	104,966.1

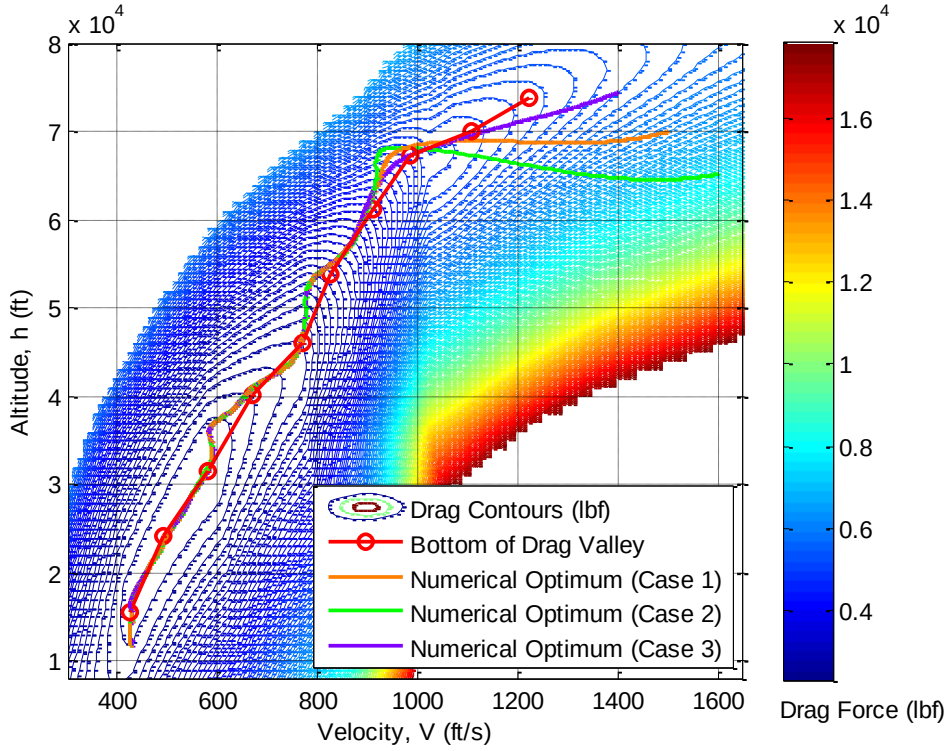

Figure 4.26. Numerically optimized trajectories with three different initial conditions converging on the bottom of the CVQEG drag valley.

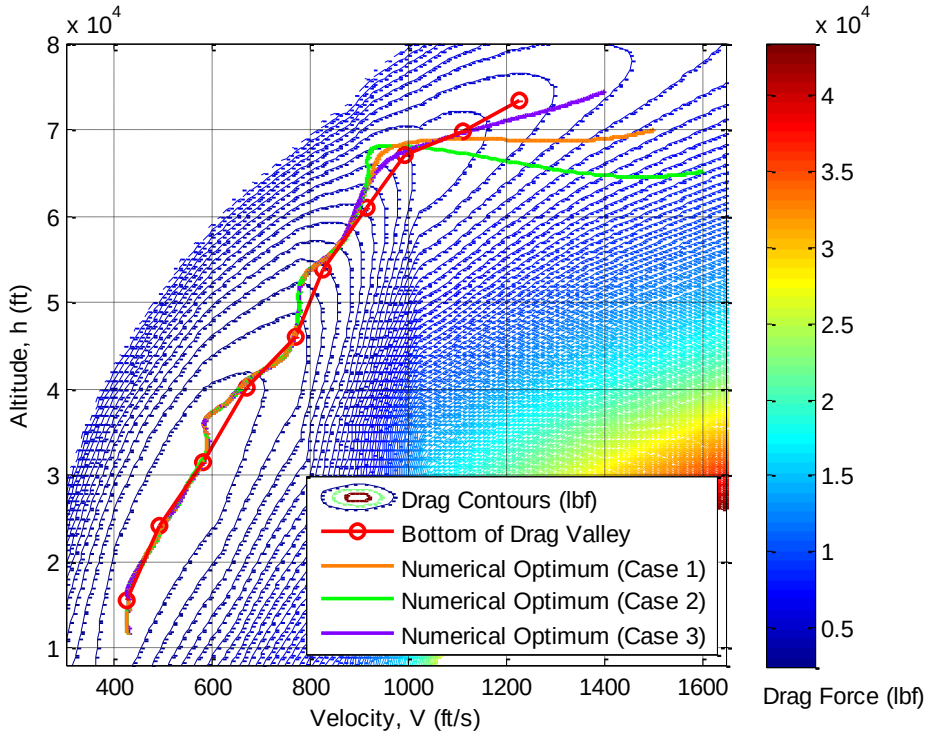

Figure 4.27. Numerically optimized trajectories with three different initial conditions converging on the bottom of the CDPEG drag valley.

Figures 4.26-4.27 indicate that the numerically optimized trajectories converge on the bottoms of the CVQEG and CDPQEG drag valleys in the V and h dimensions. The figures do not indicate, however, if the trajectories converge on the other state values corresponding to the bottoms of the valleys. The dynamic pressures and angles of attack are very similar from the bottom of one valley to the other, and the optimal trajectories tend to converge on these values, as shown in Figs. 4.28-4.29. The flight-path angles along the bottoms of the valleys are significantly different, however, as shown in Fig. 4.30. Here the optimal trajectories tend to favor the CDPQEG flight-path angles over those of CVQEG. This is a very important difference between the two QEG approaches, indicating that CDPQEG may be the more accurate approach for maximizing range. Finally, Fig. 4.31 illustrates that there is little difference between the ranges achieved by

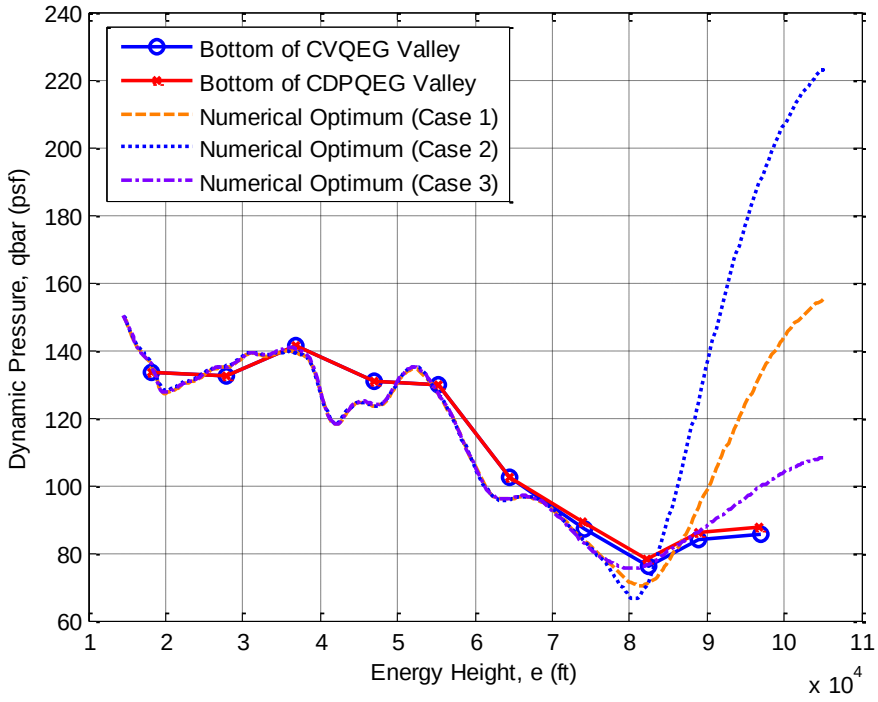

Figure 4.28. Dynamic pressure along numerically optimized trajectories with three different initial conditions, compared to dynamic pressures corresponding to the bottoms of the CVQEG and CDPQEG drag valleys.

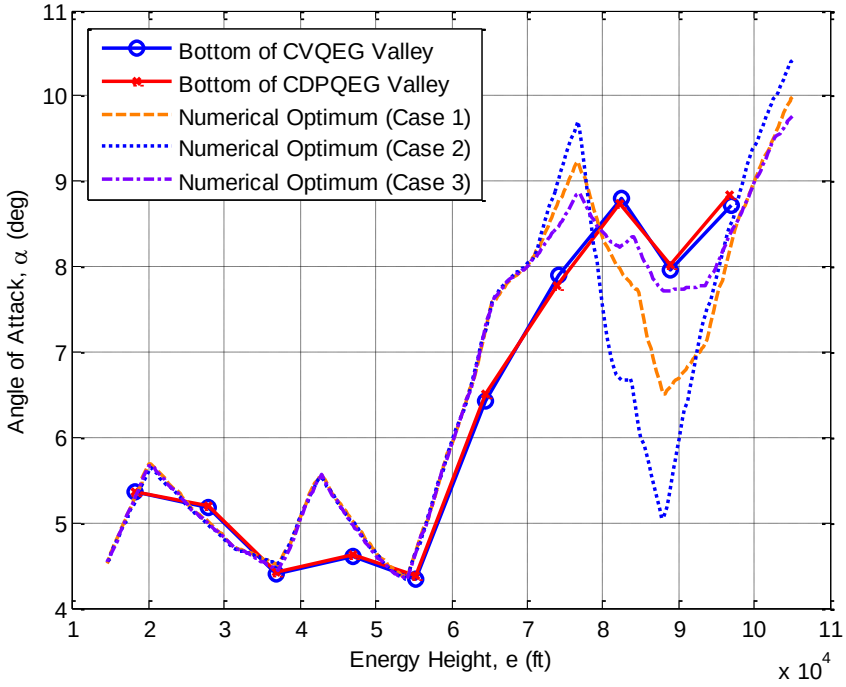

Figure 4.29. Angle of attack along numerically optimized trajectories with three different initial conditions, compared to angles of attack corresponding to the bottoms of the CVQEG and CDPQEG drag valleys.

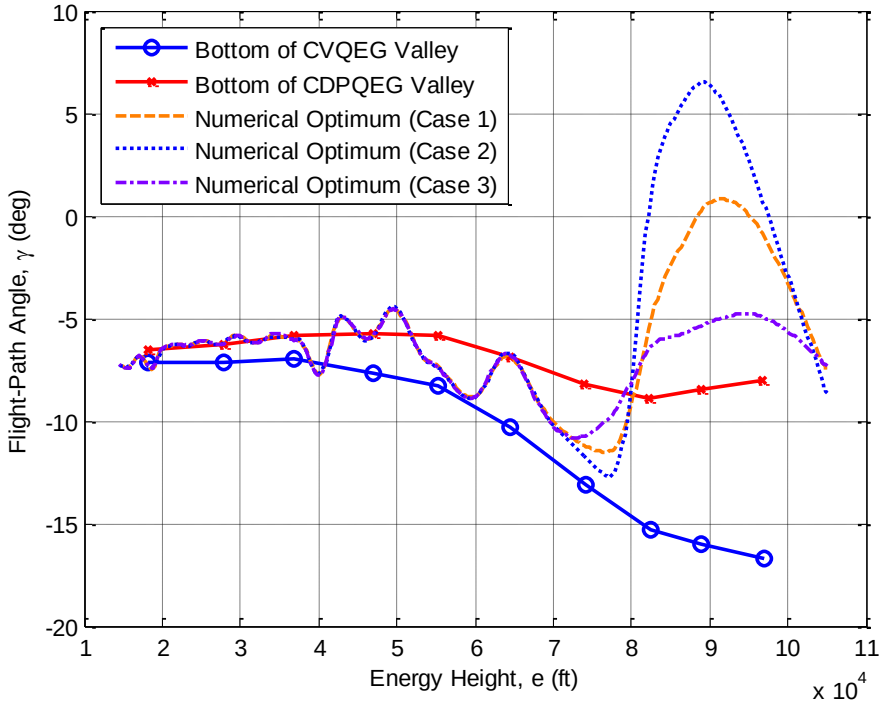

Figure 4.30. Flight-path angle along numerically optimized trajectories with three different initial conditions, compared to flight-path angles corresponding to the bottoms of the CVQEG and CDPQEG drag valleys.

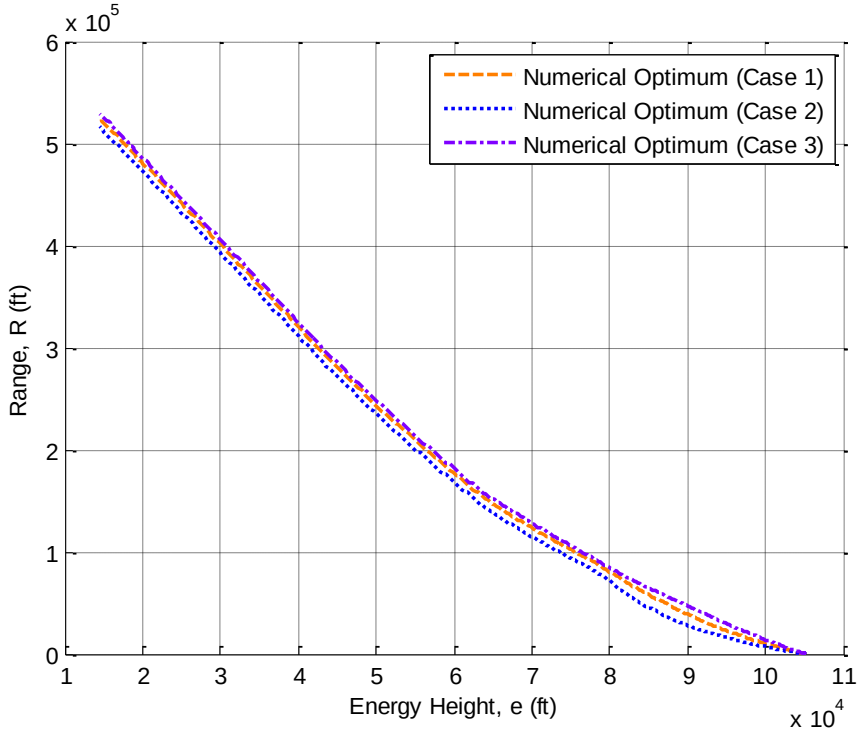

Figure 4.31. Ranges achieved by numerically optimized trajectories with three different initial conditions.

the three optimized trajectories. Given that the bottoms of the drag valleys are not actual trajectories, no ranges are displayed for them.

Considering the behavior of numerically optimized trajectories in tending to converge on the bottom of the drag valleys, with flight-path angles converging on those at the bottom of the CDPQEG valley, an intriguing direction for future research is to develop a control system that tracks not only flight-path angle (as in Section 4.4), but also velocity and altitude, driving the vehicle toward the bottom of the CDPQEG valley.

CHAPTER 5: OPTIMAL CONTROL THEORY

5.1. Two-Point Boundary Value Problem (2PBVP)

Another approach to finding the control law is to derive it using optimal control theory as a Two-Point Boundary Value Problem (2PBVP). Due to the complexities of this derivation—in particular, finding costate equations by taking partial derivatives of the objective function with respect to each state variable—the constant drag-polar aerodynamic model (Eqs. (2.15)-(2.16)) and the exponential atmospheric model (Eqs. (2.26)-(2.27)) are used. These models have much simpler derivatives than the piecewise aerodynamic and atmospheric models employed in the numerical optimization (see Sections 2.2 and 2.3 for descriptions of these models). Consequently, the resulting control law is only an approximation of the true optimal control law, but it may provide insight into the true law and the behavior that is characteristic of maximum-range trajectories.

For ease in derivation, the control is defined as the lift coefficient C_L, instead of angle of attack α, with the understanding that α can be easily calculated from C_L when using a constant drag-polar aerodynamic model, for which C_L is simply a linear function of α (Eq. (2.15)).

The traditional 2PBVP [25] seeks to minimize a performance index J, which is defined as

$$J = \phi[\{x(T)\}] + \int_0^T \hat{L}(x, u, t)dt \tag{5.1}$$

where $\phi[x(T)]$ is a scalar function of the state vector x at final time T, and $\hat{L}(x,u,t)$ is a scalar function of the state vector x, control input u, and time t ($\hat{L}$ is usually notated as L, but it is modified here to avoid confusion with lift force L). As discussed in Section 2.1, the final flight time of the TAEM phase is unknown, so instead of using T, the performance index is redefined in terms of final energy height e_f. Moreover, the objective of this investigation is to maximize the final range of the vehicle, so the performance index for this investigation is solely a function of one terminal state (i.e., range), and $\hat{L}(x,u,t)$ can be set equal to zero. Because the traditional optimal control problem seeks to minimize J, it is here defined as the negative of the range at the final energy height e_f, just as it was defined for the numerical optimization problem (Eq. (3.1)):

$$J = -R(e_f) \tag{5.2}$$

where the final range $R(e_f)$ can be found by integrating the equation dR/de (from Eq. (2.10)) from the initial energy height e_0 to the final energy height e_f. Hence, Eq. (5.2) can be expanded to

$$J = -\int_{e_0}^{e_f} \frac{dR}{de}\,de = \int_{e_0}^{e_f} \frac{mg}{D}\cos\gamma\,de \tag{5.3}$$

The minimization of J is subject to constraints on the dynamics of the vehicle, which are simply the equations of motion of the vehicle, expressed as derivatives with respect to energy height (Eqs. (2.7)-(2.10)):

$$\hat{f}_1 = \frac{dV}{de} = \frac{g}{V} + \frac{mg^2}{DV}\sin\gamma \tag{5.4}$$

$$\hat{f}_2 = \frac{d\gamma}{de} = -\frac{gL}{DV^2} + \frac{mg^2}{DV^2}\cos\gamma \tag{5.5}$$

$$\hat{f}_3 = \frac{dh}{de} = -\frac{mg}{D}\sin\gamma \tag{5.6}$$

$$\hat{f}_4 = \frac{dR}{de} = -\frac{mg}{D}\cos\gamma \tag{5.7}$$

In the general 2PBVP, the system dynamics are enforced by multiplying the constraint equations by a vector of Lagrange multipliers $\{\lambda\}$ and adding the product to the performance index (Eq. (5.1)):

$$\tilde{J} = \phi[x(T)] + \int_0^T \hat{L}(x,u,t)dt + \int_0^T \{\lambda\}^T(\{f\} - \dot{x})dt \tag{5.8}$$

where $\{\lambda\} = \left[\lambda_V\ \lambda_\gamma\ \lambda_h\ \lambda_R\right]^T$ and λ_V, λ_γ, λ_h, and λ_R are Lagrange multipliers (commonly called costates in the optimal control problem) for V, γ, h, and R, respectively, $\{f\}$ is a vector of state derivatives with respect to time, and $\dot{x}$ denotes the time derivative of state

vector x. The superscripted symbol ^{T} following a vector or matrix indicates that the vector or matrix is being transposed. Given that the performance index for this investigation is defined in terms of final energy height e_f instead of final time T, the augmented performance index of Eq. (5.8) becomes

$$\tilde{J} = -R(e_f) + \int_0^{e_f} \{\lambda\}^T \left(\{ \hat{f} \} - \frac{dx}{de} \right) de \tag{5.9}$$

A Hamiltonian H is defined to aid in notation for deriving the costate equations. The typical definition of the Hamiltonian for the 2PBVP is defined in terms of time t:

$$H(x, u, \{\lambda\}, t) = \hat{L}(x, u, t) + \{\lambda\}^T \{ f(x, u, t) \} \tag{5.10}$$

Rewriting Eq. (5.10) in terms of energy height and setting $\hat{L} = 0$ produces the Hamiltonian for this problem:

$$H(x, u, \{\lambda\}, e) = \{\lambda\}^T \{ \hat{f}(x, u, e) \} \tag{5.11}$$

Equation (5.11) can be expanded to

$$H = \lambda_V \hat{f}_1 + \lambda_\gamma \hat{f}_2 + \lambda_h \hat{f}_3 + \lambda_R \hat{f}_4 \tag{5.12}$$

The optimality criterion for the 2PBVP is found by setting $\partial H/\partial u = 0$, where u is the control input. In this case, as noted previously, the control input is lift coefficient, so $u = C_L$. Therefore, the optimality criterion is

$$\frac{\partial H}{\partial u} = \frac{\partial}{\partial C_L}\left(\lambda_V \hat{f}_1 + \lambda_\gamma \hat{f}_2 + \lambda_h \hat{f}_3 + \lambda_R \hat{f}_4\right) = 0 \tag{5.13}$$

which can be calculated as separate partial derivatives and then summed together. Recalling that $L = \frac{1}{2}\rho V^2 S C_L$ and $D = \frac{1}{2}\rho V^2 S C_D$ (Eqs. (2.12)-(2.14)), and recalling that for a constant drag-polar, $C_D = C_{D_0} + K{C_L}^2$ (Eq. (2.16)), these partial derivatives can be computed:

$$\frac{\partial}{\partial C_L}\left(\lambda_V \hat{f}_1\right) = -\lambda_V \left(\frac{4mgK}{\rho V^2 S\left(C_{D_0} + KC_L^2\right)^2}\right)\frac{g\sin\gamma}{V} C_L \tag{5.14}$$

$$\frac{\partial}{\partial C_L}\left(\lambda_\gamma \hat{f}_2\right) = \lambda_\gamma \left(\frac{4mgK}{\rho V^2 S\left(C_{D_0} + KC_L^2\right)^2}\right)\left[\left(\frac{\rho V^2 S C_L}{2m} - g\cos\gamma\right) C_L - \frac{\rho V^2 S}{4mK}\left(C_{D_0} + KC_L^2\right)\right] \tag{5.15}$$

$$\frac{\partial}{\partial C_L}\left(\lambda_h \hat{f}_3\right) = \lambda_h \left(\frac{4mgK}{\rho V^2 S\left(C_{D_0} + KC_L^2\right)^2}\right)(\sin\gamma) C_L \tag{5.16}$$

$$\frac{\partial}{\partial C_L}\left(\lambda_R \hat{f}_4\right) = \lambda_R \left(\frac{4mgK}{\rho V^2 S\left(C_{D_0} + KC_L^2\right)^2}\right)(\cos\gamma) C_L \tag{5.17}$$

where the derivations of Eqs. (5.14)-(5.17) can be found in Appendix A (Eqs. (A.1)-(A.4)). Removing the common factor, $\left(\frac{4mgK}{\rho V^2 S\left(C_{D_0}+KC_L^2\right)^2}\right)$, which is nonzero, from Eqs. (5.14)-(5.17), and substituting the equations into the optimality criterion (Eq. (5.13)) results in

$$\left[-\lambda_V \frac{g\sin\gamma}{V} C_L + \lambda_\gamma \left(\left(\frac{\rho V^2 S C_L}{2m} - g\cos\gamma\right) C_L - \frac{\rho V^2 S}{4mK}\left(C_{D_0} + KC_L^2\right)\right) + \lambda_h(\sin\gamma)C_L + \lambda_R(\cos\gamma)C_L\right] = 0 \quad (5.18)$$

Equation (5.18) can be rearranged to a quadratic function of C_L:

$$\left[-\lambda_\gamma \frac{\rho V^2 S}{4m} C_L^2 + \left[-\lambda_V \frac{g\sin\gamma}{V} + \lambda_\gamma \left(\frac{\rho V^2 S C_L}{2m} - g\cos\gamma\right) + \lambda_h \sin\gamma + \lambda_R \cos\gamma\right] C_L - \frac{\rho V^2 S C_{D_0}}{4mK}\right] = 0 \quad (5.19)$$

Equation (5.19) can be solved for C_L with the quadratic formula to find the optimal control profile:

$$C_L = \frac{1}{\lambda_\gamma \frac{\rho V^2 S}{2m}}\left[-\lambda_V \frac{g\sin\gamma}{V} + \lambda_\gamma \left(\frac{\rho V^2 S C_L}{2m} - g\cos\gamma\right) + \lambda_h \sin\gamma + \lambda_R \cos\gamma \pm \sqrt{\left[-\lambda_V \frac{g\sin\gamma}{V} + \lambda_\gamma \left(\frac{\rho V^2 S C_L}{2m} - g\cos\gamma\right) + \lambda_h \sin\gamma + \lambda_R \cos\gamma\right]^2 - \lambda_\gamma \frac{\rho^2 V^4 S^2 C_{D_0}}{4m^2 K}}\right] \quad (5.20)$$

There could be two real solutions to Eq. (5.20), so an additional criterion may be used to determine which solution is, in fact, the optimal control profile. This additional criterion requires that the concavity of H be downward so that H is a maximum:

$$\frac{\partial^2 H}{\partial u^2} = \frac{\partial^2}{\partial C_L^2}\left(\lambda_V \hat{f}_1 + \lambda_\gamma \hat{f}_2 + \lambda_h \hat{f}_3 + \lambda_R \hat{f}_4\right) < 0 \tag{5.21}$$

Hence, the partial derivatives of Eqs. (5.14)-(5.17) can be taken with respect to C_L, substituted into Eq. (5.21), and evaluated for each solution of Eq. (5.20) to determine which solution is truly optimal. An alternate approach, which is simpler than deriving the required partial derivatives, is to evaluate the range achieved with each solution of Eq. (5.20) and determine which control profile results in the better range.

Note that the optimal control profile in Eq. (5.20) is expressed in terms of the costates: $\lambda_V, \lambda_\gamma, \lambda_h, \lambda_R$. To evaluate these costates throughout the trajectory requires solving the costate equations, which are derived from the Hamiltonian. For the 2PBVP defined in terms of time t, the costate equations are found as

$$\dot{\lambda} = -\left(\frac{\partial H}{\partial x}\right)^T \tag{5.22}$$

recalling that λ is a vector of costates, and x is a vector of states. Rewriting Eq. (5.22) in terms of energy height results in

$$\frac{d\lambda}{de} = -\left(\frac{\partial H}{\partial x}\right)^T \tag{5.23}$$

Substituting the Hamiltonian (Eq. (5.12)) and the functions in it ($\hat{f}_1$, $\hat{f}_2$, $\hat{f}_3$, and $\hat{f}_4$ from Eqs. (5.4)-(5.7)) into Eq. (5.23) results in the costate equations. Using the relationships of $L = \frac{1}{2}\rho V^2 S C_L$ and $D = \frac{1}{2}\rho V^2 S C_D$ (Eqs. (2.12)-(2.14)), the constant drag-polar aerodynamic model $C_D = C_{D_0} + K C_L{}^2$ (Eq. (2.16)), and the exponential atmospheric model $\rho = \rho_0 \exp\left(-\frac{h}{H_s}\right)$ (Eq. (2.26)), the costate equations are derived as

$$\frac{d\lambda_V}{de} = -\frac{\partial H}{\partial V} = -\lambda_V\left(-\frac{g}{V^2} - \frac{3mg^2}{DV^2}\sin\gamma\right) - \lambda_\gamma\left(\frac{2gL}{DV^3} - \frac{4mg^2}{DV^3}\cos\gamma\right) - \lambda_h\left(\frac{2mg}{DV}\sin\gamma\right) - \lambda_R\left(\frac{2mg}{DV}\cos\gamma\right) \quad (5.24)$$

$$\frac{d\lambda_\gamma}{de} = -\frac{\partial H}{\partial \gamma} = -\lambda_V\left(\frac{mg^2}{DV}\cos\gamma\right) - \lambda_\gamma\left(-\frac{mg^2}{DV^2}\sin\gamma\right) - \lambda_h\left(-\frac{mg}{D}\cos\gamma\right) - \lambda_R\left(\frac{mg}{D}\sin\gamma\right) \quad (5.25)$$

$$\frac{d\lambda_h}{de} = -\frac{\partial H}{\partial h} = -\lambda_V\left(\frac{\rho_0 mg^2}{\rho H_s DV}\sin\gamma\right) - \lambda_\gamma\left(\frac{\rho_0 mg^2}{\rho H_s DV^2}\cos\gamma\right) - \lambda_h\left(-\frac{\rho_0 mg}{\rho H_s D}\sin\gamma\right) - \lambda_R\left(-\frac{\rho_0 mg}{\rho H_s D}\cos\gamma\right) \quad (5.26)$$

$$\frac{d\lambda_R}{de} = -\frac{\partial H}{\partial R} = 0 \quad (5.27)$$

The states have known initial conditions (discussed in Section 2.1), as in the numerical optimization problem of Chapter 3, but the costates have no known initial

conditions. Conversely, the states have no specified terminal boundary conditions, but the costates are given terminal boundary conditions. The terminal boundary conditions of the costates, also known as transversality conditions, are traditionally defined in terms of final time T as

$$\{\lambda(T)\} = -\left(\frac{\partial \phi}{\partial x}\right)^T\Big|_{t\,=\,T} \tag{5.28}$$

where ϕ is a scalar function of the state vector x at final time T (noting again that the superscript symbol ^{T} indicates the transpose of a vector or matrix). Rewriting Eq. (5.28) in terms of energy height results in

$$\{\lambda(e_f)\} = -\left(\frac{\partial \phi}{\partial x}\right)^T\Big|_{e\,=\,e_f} \tag{5.29}$$

Recall from Eqs. (5.1)-(5.2) that $\phi = -R(e_f)$, and ϕ is the only term in the performance index, since $\hat{L}$ was set equal to zero. Therefore, the transversality conditions of Eq. (5.29) become

$$\lambda_V(e_f) = \frac{\partial}{\partial V}\Big(-R(e_f)\Big)\Big|_{e\,=\,e_f} = 0 \tag{5.30}$$

$$\lambda_\gamma(e_f) = \frac{\partial}{\partial \gamma}\Big(-R(e_f)\Big)\Big|_{e\,=\,e_f} = 0 \tag{5.31}$$

$$\lambda_h(e_f) = \frac{\partial}{\partial h}\left(-R(e_f)\right)\Big|_{e = e_f} = 0 \tag{5.32}$$

$$\lambda_R(e_f) = \frac{\partial}{\partial R}\left(-R(e_f)\right)\Big|_{e = e_f} = -1 \tag{5.33}$$

As in the general 2PBVP, the performance index (Eq. (5.2)) is minimized, subject to the state equations (Eqs. (5.4)-(5.7)) and costate equations (Eqs. (5.24)-(5.27)), when the optimality criteria (Eqs. (5.20)-(5.21)) are met (the traditional 2PBVP definition omits Eq. (5.21) as a criterion, though it is still valid, by assuming there is only one solution to $\partial H/\partial u = 0$), given fixed initial states and free terminal states, and given free initial costates and fixed terminal costates (Eqs. (5.30)-(5.33)).

5.2. Approach to Solving the Two-Point Boundary Value Problem

Solving the problem defined here requires integrating the costate equations and substituting them into the optimality criterion to evaluate the control input throughout the trajectory. One approach to finding the optimal control profile is to use a multiple-shooting method, which breaks down the problem into a series of boundary value problems on smaller intervals, stringing the solutions together into an overall solution. Such an approach will likely require some experimenting to find parameters (such as interval size) that produce a numerically stable and accurate solution. If a solution is found, it may lend insight into the behavior of true maximum-range trajectories.

The initial states and final costates are known, but the final states and initial costates are unknown. Therefore, to evaluate the optimal control profile, which is a

function of the states and costates, there must be some approximation of the states and costates already. The states can be propagated forward (e_0 to e_f) using a nominal control profile (such as the max-*L/D* profile), providing a nominal trajectory. The costates can then be propagated backward (e_f to e_0) using the nominal state profiles, since the costate equations (Eqs. (5.24)-(5.27)) are functions of the states and costates. The states can be re-propagated forward by evaluating the control input with the new costate profiles (see Eq. (5.20)). Then the costates can be re-propagated backward using the new state profiles. This process can be repeated as many times as necessary until the final range converges to within some specified tolerance (i.e., the relative error between successive iterations of forward state propagation falls below some tolerance).

Because this problem is set up with simplified aerodynamic and atmospheric models, the solution to this problem will not necessarily perform optimally under realistic aerodynamic and atmospheric effects. However, insights from the behavior of optimal trajectories in this problem could lead to a deeper understanding of how actual maximum-range trajectories behave, perhaps aiding in the development of a computationally fast and accurate onboard control system for the vehicle.

CHAPTER 6: CONCLUSION

6.1. Summary of Findings

A software package was developed using MATLAB to numerically maximize the range of an unpowered Reusable Launch Vehicle (RLV) during the Terminal-Area Energy Management (TAEM) phase of reentry. A reasonably accurate and computationally efficient aerodynamic model of the X-34 launch vehicle was developed to test the software.

Numerical optimization of the trajectory was conducted for a variety of numbers of control nodes, illustrating the tradeoff between maximum range achieved and computational time. Two different approaches to initializing the numerical optimization were tested—the inherited-initials and the zero-initials approaches. A reasonable compromise between computational time and maximum range achieved was obtained by using the 17-node zero-initials approach, which provided a 3.10 % improvement in range over the max-*L/D* trajectory and required only 62.5 s (1.04 min) to compute on an AMD Athlon II X4 630 quad-core processor running MATLAB R2010a in Windows 7. This improvement in range is very good, considering that the greatest improvement was achieved with the 65- and 129-node inherited-initials solutions, both of which provided a 3.16 % improvement over max-*L/D* and required over four times as much computational time. Though 62.5 s may not be low enough for real-time updates of the trajectory optimization in an onboard control system, it is sufficiently low to update the control profile thirteen times throughout a trajectory that lasts 816.9 s. Given that the onboard control system could follow the most recent control profile while the profile is being

updated to reflect the actual current states, numerical optimization might be a feasible approach to designing an onboard control system.

It was found that both the max-*L/D* and the numerically optimized trajectories exhibited oscillatory behavior in flight-path angle. It was hypothesized that range maximization requires a certain amount of oscillation in flight-path angle, initiated by a high angle of attack. Given just the right amount of "lobbing" the vehicle along a higher flight-path angle, the vehicle's altitude is increased (or, at least, its loss of altitude is slowed), and the vehicle travels farther than it would have by following a steady flight-path angle. Too high a flight-path angle, however, could reduce the horizontal component of velocity too greatly and reduce the overall range achieved, so the right balance must be struck between maximizing horizontal velocity and minimizing loss of altitude.

Stochastic methods of numerical optimization were briefly discussed, especially Particle-Swarm Optimization (PSO). In general, however, it was concluded that gradient-based methods are likely the most effective and computationally efficient approach to range maximization, as it is believed that the objective function is continuous and unimodal.

The maximum lift-to-drag approach to trajectory optimization was discussed, including why it does not result in an equilibrium glide trajectory when atmospheric density changes with altitude. The concept of a Quasi-Equilibrium Glide (QEG) was considered as a possible approximation of maximum-range trajectories that could be implemented in a real atmosphere. Successful tracking of QEG conditions, however, requires a feedback controller. Two particular QEG approaches were discussed in detail:

Constant-Velocity Quasi-Equilibrium Glide (CVQEG) and Constant-Dynamic-Pressure Quasi-Equilibrium Glide (CDPQEG). It was found that the Newton-Raphson Method could be used to solve for the flight-path angle and angle of attack necessary to satisfy QEG conditions for either approach at a particular velocity and altitude. By plotting the drag corresponding to either QEG approach at a specified velocity and altitude, a so-called drag valley could be found. Numerically optimized trajectories were found to converge on the bottom of the drag valley as energy height decreased, even when those trajectories began with different velocities but the same energy height. In particular, the flight-path angle of numerically optimized trajectories seemed to favor the flight-path angles corresponding to the bottom of the CDPQEG drag valley.

A control system was proposed that tracked QEG flight-path angle. Simulations of this control system, however, were unable to outperform the max-*L/D* approach, even with a variety of different gain profiles. Moreover, the QEG simulations often severely underperformed the max-*L/D* approach. Interestingly, though, the flight-path angle along the max-*L/D* trajectory seemed to oscillate about the flight-path angle of the trajectory that used a closed-loop CDPQEG control system. The CDPQEG approach appeared to follow a steadier flight-path angle than the max-*L/D* and numerically optimized trajectories, avoiding the oscillations included in these trajectories. The omission of these oscillations may have contributed to the lower ranges achieved by the CDPQEG approach. The CVQEG control system was successful at maintaining a fairly constant velocity along the beginning of its course, but to maintain constant velocity indefinitely required increasingly severe maneuvering that ultimately detracted from the range achieved. The CDPQEG control system was very successful at maintaining a constant

dynamic pressure throughout its course. Though far less successful at maximizing range, the QEG control systems occasionally required less computational time than the max-*L/D* approach (~0.2 s instead of 0.395 s, though usually QEG required more time) and much less time than numerical optimization. Therefore, if an appropriate QEG control system is developed (perhaps one that tracks the bottom of the QEG drag valley instead of merely tracking the QEG flight-path angle for a given velocity and altitude), and if it is comparable to or outperforms the max-*L/D* approach, then a QEG control system might be a more desirable means of maximizing range than max-*L/D* or numerical optimization, due to its lower computational time. Especially if such a system could approximate numerically optimized trajectories for less computational time, it would be a good candidate for implementation in an onboard control system.

A Two-Point Boundary Value Problem (2PBVP) was set up to find an optimal control profile for the vehicle using optimal control theory. Simpler aerodynamic and atmospheric models were used for this approach to facilitate the derivation of the optimality criterion and costate equations. It was found that the optimality criterion resulted in a quadratic equation in terms of the control input, but both roots of the equation could be evaluated to determine which root is the realistic solution (or the concavity of the performance index could be evaluated at each solution to determine which yields maximum range).

In summary, a variety of approaches were considered for RLV range maximization in the TAEM phase of reentry. Based on this study, numerical optimization appears to be the most promising solution at this time for implementation in an onboard control system.

6.2. Areas for Future Research

A variety of possible areas for future research exist in this field. Additional investigation of the effect of integration step size on maximum range achieved and accuracy of simulation should be conducted. Integration step size could play a critical role in determining whether numerical optimization converges on an optimal control profile and, if it does converge, on the accuracy of that profile and its corresponding range. Numerical stability and reliability of convergence are of extreme importance if numerical optimization is implemented in an onboard control system. Hence, numerical optimization must also be tested for robustness over a wide range of initial conditions. Further improvements in computational time would be highly advantageous for implementation in a control system.

For the sake of robustness, stochastic numerical optimization algorithms might be applied more rigorously to this problem. As noted in this study, however, it is believed that gradient-based algorithms are more effective and computationally efficient than stochastic methods for this objective function.

Given the promise of numerical optimization techniques for this problem, alternate methods of defining the control profile should be investigated. It may be that a different means of interpolating between control nodes (e.g., spline fit, Fourier series, polynomial interpolation) would expedite numerical optimization by reducing the number of control nodes and, hence, the number of function evaluations, required for a given range achieved. Reducing the number of independent variables in the problem often reduces the number of iterations required to converge on a solution, though the higher-

order interpolation method would also increase the computational time required for each integration step along the trajectory.

A higher-order interpolation method might also improve the realism of the control profile by making it more feasible to employ with real control surfaces, given that real control surfaces cannot respond instantly to control commands. For that matter, it might be helpful to include the pitch control dynamics of the vehicle in the simulation model, rather than assuming the vehicle can adjust its angle of attack instantaneously. These un-modeled details could affect the optimization of the control profile. Along with other methods of interpolating between control nodes, it may be desirable to consider non-uniform distributions of nodes along the trajectory, placing more nodes in areas needing higher resolution (e.g., at energy heights for which velocity is transonic), improving the range of the vehicle without the computational expense of increasing the number of control nodes.

As for the QEG approaches, much research is needed to determine whether the CDPQEG approach is a valid approximation of numerically optimized trajectories and whether the CVQEG approach is truly not a valid approximation. More study of the control system that tracks QEG flight-path angle is warranted, to determine the gain profiles that result in the greatest range, and to find if the system can ever outperform the max-*L/D* approach.

An especially interesting direction for future research is to parameterize velocity and flight-path angle along the bottom of the CDPQEG drag valley as functions of energy height, and then to develop a control system that tracks the valley-bottom velocity and flight-path angle. Given that numerically optimized trajectories tend to converge on the

states along the bottom of the CDPQEG drag valley, this approach seems more promising than merely tracking the CDPQEG flight-path angle at a given velocity and altitude. The control system would be tracking a particular velocity-altitude pair and its corresponding CDPQEG flight-path angle. The fact that the max-*L/D* flight-path angle tended to oscillate about the flight-path angle of the trajectory that used a CDPQEG control system also seems to support the validity of the CDPQEG drag valley bottom for maximizing range. Note, however, that the directness of the CDPQEG control system (i.e., avoiding the oscillations associated with max-*L/D* and numerically optimized trajectories) may mean that such a two-variable control system would also underperform the max-*L/D* approach. It might be possible to mitigate this loss of range by tuning the control system to exhibit an underdamped and, therefore, oscillatory, mode of tracking the bottom of the drag valley.

If the Two-Point Boundary Value Problem (2PBVP) of Chapter 5 could be solved numerically, the solution might lend some insight into how optimal trajectories behave. Perhaps a simple control law could be derived from the problem and used to approximate maximum-range trajectories using more realistic aerodynamic and atmospheric models. If a simple control law could be derived and shown to perform as well as or better than the max-*L/D* approach—especially if it closely approximated numerically optimized trajectories—and if it required less computational time than these approaches, then it might be a better approach for maximizing range in an onboard control system. Multiple-shooting methods might be employed to solve the problem by breaking the trajectories up into multiple boundary-value problems and solving each problem in series, then comparing the resulting trajectory with the boundary conditions and trying again. The

costate equations would be integrated in one direction through multiple boundary-value problems, and then the state equations would be integrated in the opposite direction. This approach to solving the problem might reduce some of the complexity and numerical instability of attacking it as a whole, but nevertheless, it is likely to require some experimenting to find a solution.

Finally, there are many other considerations that could be added to the range maximization problem. Modeling wind effects, for instance, could add another layer of realism to the simulation and allow for more robust trajectory solutions. Additional aspects of the TAEM phase might be incorporated into the model, such as finding optimal trajectories for turning maneuvers (e.g., Heading Alignment Cylinders). Perhaps allowing the vehicle to bank would provide some more sophisticated means of maximizing range. The possibilities for adding realism and degrees of freedom are manifold, but it must also be considered how these will affect the reliability and computational efficiency of the control system. Accuracy, speed, and robustness are paramount if such a system is to be implemented in an onboard control system and, particularly, if the system is to improve upon the ability of RLVs to return safely and cheaply to Earth.

APPENDIX A: OPTIMAL-CONTROL DERIVATIONS

The following equations explain how the partial derivatives of the Hamiltonian with respect to the control input (Eqs. (5.14)-(5.17)) were derived, making use of the relationships of $L = \frac{1}{2}\rho V^2 S C_L$ and $D = \frac{1}{2}\rho V^2 S C_D$ (Eqs. (2.12)-(2.14)), and the constant drag-polar aerodynamic model, for which $C_D = C_{D_0} + K{C_L}^2$ (Eq. (2.16)). The functions $\hat{f}_1$, $\hat{f}_2$, $\hat{f}_3$, and $\hat{f}_4$ denote the rates of change of each state (V, γ, h, and R) with respect to energy height e, as defined in Eqs. (5.4)-(5.7). Costates of each state are denoted λ_V, λ_γ, λ_h, and λ_R, respectively. C_L is the control input u for the optimal control problem.

$$\frac{\partial}{\partial C_L}\left(\lambda_V \hat{f}_1\right) = \frac{\partial}{\partial C_L}\left(\lambda_V\left(\frac{g}{V} + \frac{mg^2}{DV}\sin\gamma\right)\right) = \frac{\partial}{\partial C_L}\left(\lambda_V\left(\frac{g}{V} + \frac{mg^2}{\left(\frac{1}{2}\rho V^2 S\left(C_{D_0}+KC_L^2\right)\right)V}\sin\gamma\right)\right)$$

$$= \lambda_V \frac{2mg^2}{\rho V^2 S}\frac{\sin\gamma}{V}\frac{\partial}{\partial C_L}\left(\frac{1}{C_{D_0}+KC_L^2}\right) = \lambda_V \frac{2mg^2}{\rho V^2 S}\frac{\sin\gamma}{V}\left(-\frac{2KC_L}{\left(C_{D_0}+KC_L^2\right)^2}\right)$$

$$= -\lambda_V\left(\frac{4mgK}{\rho V^2 S\left(C_{D_0}+KC_L^2\right)^2}\right)\frac{g\sin\gamma}{V}C_L \tag{A.1}$$

$$\frac{\partial}{\partial C_L}\left(\lambda_\gamma \hat{f}_2\right) = \frac{\partial}{\partial C_L}\left(\lambda_\gamma\left(-\frac{gL}{DV^2} + \frac{mg^2}{DV^2}\cos\gamma\right)\right)$$

$$= \frac{\partial}{\partial C_L}\left(\lambda_\gamma\left(-\frac{g\left(\frac{1}{2}\rho V^2 S C_L\right)}{V^2} + \frac{mg^2}{V^2}\cos\gamma\right)\left(\frac{1}{\frac{1}{2}\rho V^2 S\left(C_{D_0}+KC_L^2\right)}\right)\right)$$

$$= \lambda_\gamma \frac{2mg}{\rho V^2 S}\frac{\partial}{\partial C_L}\left(\left(-\frac{\rho V^2 S C_L}{2m} + g\cos\gamma\right)\left(\frac{1}{C_{D_0}+KC_L^2}\right)\right)$$

$$= \lambda_\gamma \frac{2mg}{\rho V^2 S}\left[\left(-\frac{\rho V^2 S C_L}{2m} + g\cos\gamma\right)\left(-\frac{2KC_L}{\left(C_{D_0}+KC_L^2\right)^2}\right) + \left(-\frac{\rho V^2 S}{2m}\right)\left(\frac{1}{C_{D_0}+KC_L^2}\right)\right]$$

$$= \lambda_\gamma \left(\frac{2mg}{\rho V^2 S\left(C_{D_0}+KC_L^2\right)^2}\right)\left[\left(\frac{\rho V^2 S C_L}{2m} - g\cos\gamma\right)(2KC_L) + \left(-\frac{\rho V^2 S}{2m}\right)\left(C_{D_0} + KC_L^2\right)\right]$$

$$= \lambda_\gamma \left(\frac{4mgK}{\rho V^2 S\left(C_{D_0}+KC_L^2\right)^2}\right)\left[\left(\frac{\rho V^2 S C_L}{2m} - g\cos\gamma\right)C_L + \left(-\frac{\rho V^2 S}{4mK}\right)\left(C_{D_0} + KC_L^2\right)\right] \quad \text{(A.2)}$$

$$\frac{\partial}{\partial C_L}\left(\lambda_h \hat{f}_3\right) = \frac{\partial}{\partial C_L}\left(\lambda_h\left(-\frac{mg}{D}\sin\gamma\right)\right) = \frac{\partial}{\partial C_L}\left(\lambda_h\left(-\frac{mg}{\left(\frac{1}{2}\rho V^2 S\left(C_{D_0}+KC_L^2\right)\right)}\sin\gamma\right)\right)$$

$$= -\lambda_h \frac{2mg}{\rho V^2 S}(\sin\gamma)\frac{\partial}{\partial C_L}\left(\frac{1}{C_{D_0}+KC_L^2}\right) = -\lambda_h \frac{2mg}{\rho V^2 S}(\sin\gamma)\left(-\frac{2KC_L}{\left(C_{D_0}+KC_L^2\right)^2}\right)$$

$$= \lambda_h \left(\frac{4mgK}{\rho V^2 S\left(C_{D_0}+KC_L^2\right)^2}\right)(\sin\gamma)C_L \quad \text{(A.3)}$$

$$\frac{\partial}{\partial C_L}\left(\lambda_R \hat{f}_4\right) = \frac{\partial}{\partial C_L}\left(\lambda_R\left(-\frac{mg}{D}\cos\gamma\right)\right) = \frac{\partial}{\partial C_L}\left(\lambda_R\left(-\frac{mg}{\left(\frac{1}{2}\rho V^2 S\left(C_{D_0}+KC_L^2\right)\right)}\cos\gamma\right)\right)$$

$$= -\lambda_R \frac{2mg}{\rho V^2 S}(\cos\gamma)\frac{\partial}{\partial C_L}\left(\frac{1}{C_{D_0}+KC_L^2}\right) = -\lambda_R \frac{2mg}{\rho V^2 S}(\cos\gamma)\left(-\frac{2KC_L}{\left(C_{D_0}+KC_L^2\right)^2}\right)$$

$$= \lambda_R \left(\frac{4mgK}{\rho V^2 S\left(C_{D_0}+KC_L^2\right)^2}\right)(\cos\gamma)C_L \quad \text{(A.4)}$$

APPENDIX B: MATLAB CODE

B.1. Optimization Initialization and Execution (`RangeOpt.m`)

```
% Space Vehicle Range Optimization Script File
% Josiah Bryan -- MAE 8990 Masters Thesis Research
% Advised by Dr. Craig Kluever
% October 28, 2010

clc
clear all
close all
format long

global e_pts
global d_alpha_pts
global odestepsize
global V0 gamma0 h0 e0
global Vconstr gammaconstr hconstr econstr
global g m S
global controlmode startingpoint
global aeroflag atmosflag
global isgammafree
global K_gamma lambdaCL

%  conversion
d2r = pi/180;
r2d = 1/d2r;

% Switch among open-loop (0), optimization (1), and both (2)
runopt = 2;

% Switch aerodynamic model (1 = drag polar, 2 = piecewise poly, 3 =
% table lookup)
aeroflag = 2;  % DO NOT USE AEROFLAG = 2 WITH ATMOSFLAG = 1
%               (No Mach # generated with atmosflag = 1)

% Switch atmospheric model (1 = exponential, 2 = default atmos.m, 3 =
% atmos_1976.m)
atmosflag = 2;

% Switch whether gamma is (TRUE) or is not (FALSE) a free variable to
be
%   optimized
isgammafree = false;

% Switch drag valley calculations on (TRUE) or off (FALSE)
dragvalleyflag = true;

% Switch whether certain plots are saved
```

```
saveplots = false;

% Set pole for closed-loop flight-path-angle QEG feedback controller
lambdaCL = -0.05;

% Set step size for integration of system
odestepsize = 1000; % ft

% Set acceleration due to gravity (assume constant)
g = 32.174;    % ft/s^2

% Aircraft variables
m = 18000/g;   % slugs   (18000 lb, X-34)
S = 357.5; % ft^2

% Define number of points in angle-of-attack profile for maximum range
NumberOfPts = 17; % number of time points at which to find angle of
attack

% Cycle through various starting points
for startingpoint = 1:3 % SET TO 3 TO USE ALL THREE STARTING POINTS
    switch startingpoint
        case 1
            V0 = 1500; % ft/s
            gamma0 = -7.56*d2r; % rad
            h0 = 70000; % ft
        case 2
            V0 = V0 + 100; % ft/s
            gamma0 = -7.56*d2r; % rad
            h0 = e0 - V0^2 / (2*g);  % ft
        case 3
            V0 = V0 - 200; % ft/s
            gamma0 = -7.56*d2r; % rad
            h0 = e0 - V0^2 / (2*g); % ft
    end

        % Initial energy height
        e0 = V0^2 / (2*g) + h0; % ft

        % Compute final energy height (econstr)
        Vconstr = 539; % ft/s (target velocity)
        hconstr = 10000; % ft (target altitude)
        econstr = Vconstr^2 / (2*g) + hconstr; % ft

    % Run open-loop simulation only if runopt == 0 or 2
    if (runopt == 0 || runopt == 2)
        RangeSim
    end

    % If runopt switch ~= 0, then continue with optimization
    if runopt ~= 0
        controlmode = 'Max L/D'; % default = Max L/D is ref. trajectory
```

```
        % If dalpha points are to be optimized, not ONLY gamma0
        if NumberOfPts > 0
            e_pts = linspace(econstr, e0, NumberOfPts); % energy hts
(ft)
        end

        % Initialize x0 as zero-degree deviations from alpha_star
profile
        x0 = zeros(1,NumberOfPts);
        if isgammafree
            x0 = [x0, gamma0]; % USE TO MAKE GAMMA0 A FREE VARIABLE
        end

        % % Use the following loop to add a new node to each interval
when
        % %  given a starting guess (x0start) with half the desired
number
        % %  of intervals (i.e., with (((NumberOfPts-1)/2)+1) nodes)
        % for i=1:NumberOfPts
        %     if mod(i,2) % if i is odd
        %         x0(i) = x0start(ceil(i/2));
        %     else % if i is even
        %         x0(i) = 1/2*(x0start(i/2)+x0start(i/2+1));
        %     end
        % end

        % Constrain minimum and maximum deviations from alpha*
(degrees)
        for (i = 1:NumberOfPts)
            % Minimum angle of attack set at -6 deg
            lower(i) = -AeroX34AlphaStar(e_pts(i),aeroflag)-6;

            % Maximum angle of attack set at +21 deg
            upper(i) = 21 - AeroX34AlphaStar(e_pts(i),aeroflag);
        end

        % If gamma0 is free variable, set side constrts to -90 and 90
deg
        if isgammafree
            lower(i+1) = -90*d2r;
            upper(i+1) = 90*d2r;
        end

        %  options
        OPTIONS = optimset('Display','iter','MaxIter',500, ...
            'MaxFunEvals',60000);

        %-----------------------------------------------------------------
----
        % CONDUCT OPTIMIZATION
        tic
        % Gradient-based MATLAB fmincon optimization
        [x_opt, f_opt] = fmincon(@Performance, x0, [], [], [], [], ...
            lower, upper, @nonlcon, OPTIONS)
```

```
        % % Use the following code to run PSO algorithm instead
        %[x_opt, f_opt] = PartSwarm(@Performance, lower, upper, 200,
...
        %   0.5, 0.25, 0.25, 20)
        toc
        %-----------------------------------------------------------------
----


        % Find states along optimal trajectory
        [V,gamma,R,t,e] = Traj(x_opt);

        % Calculate h along trajectory
        h = e - V.^2 / (2*g); % array of h values along trajectory

        % Number of data pts along optimal trajectory
        trajlength = numel(V);

        % Calculate Mach number, drag, and alpha along optimal
trajectory
        rho = zeros(trajlength,1);
        Mach_plot = zeros(trajlength,1);
        d_alpha_plot = zeros(trajlength,1);
        alpha_star_plot2 = zeros(trajlength,1);
        dragtraj = zeros(trajlength,1);
        for i=1:trajlength
            % Calculate density, speed of sound, and Mach number
            [rho(i,1), a_temporary] = atmos(2,h(i),atmosflag);
            Mach_plot(i) = V(i) / a_temporary;

            % Calculate alpha* for Max L/D along trajectory
            alpha_star_plot2(i) =
AeroX34AlphaStar(Mach_plot(i),aeroflag);

            % Calculate d_alpha along trajectory by linear
interpolation
            %   of optimal control profile
            if ~isempty(d_alpha_pts)
                d_alpha_plot(i) = lininterp(e_pts,d_alpha_pts,e(i));
            end

            % Calculate CL, CD
            [CL_temporary,CD_temporary] = AeroX34Piece(d_alpha_plot(i)
...
                + alpha_star_plot2(i),Mach_plot(i),aeroflag);

            % Calculate drag
            dragtraj(i) = 1/2*rho(i,1)*V(i)^2*S*CD_temporary;
        end

        % Compute qbar for trajectory
        qbar = 1/2*rho.*V.^2;
```

```
        % PLOT STATES ALONG OPTIMAL TRAJECTORY
        plothndl=figure('Name', ...
  ['Velocity vs. Energy Height for Numerically Optimized Path for Case
'...
            num2str(startingpoint)]);
        plot(e,V)
        grid
        xlabel('Energy Height, e (ft)')
        ylabel('Velocity, V (ft/sec)')
        % Save plot if saveplots flag == true
        if saveplots
            saveas(plothndl,['V_e_' num2str(NumberOfPts) '.fig'])
        end

        plothndl=figure('Name', [...
'Mach Number vs. Energy Height for Numerically Optimized Path for Case
'...
            num2str(startingpoint)]);
        plot(e,Mach_plot)
        grid
        xlabel('Energy Height, e (ft)')
        ylabel('Mach Number, Ma')
        if saveplots
            saveas(plothndl,['M_e_' num2str(NumberOfPts) '.fig'])
        end

        plothndl=figure('Name', ['Flight-Path Angle vs. Energy Height
for Numerically Optimized Path for Case ' ...
            num2str(startingpoint)]);
        plot(e,gamma)
        grid
        xlabel('Energy Height, e (ft)')
        ylabel('Flight-Path Angle, \gamma (deg)')
        if saveplots
            saveas(plothndl,['g_e_' num2str(NumberOfPts) '.fig'])
        end

        plothndl=figure('Name', ...
  ['Altitude vs. Energy Height for Numerically Optimized Path for Case
'...
            num2str(startingpoint)]);
        plot(e,h)
        grid
        xlabel('Energy Height, e (ft)')
        ylabel('Altitude, h (ft)')
        if saveplots
            saveas(plothndl,['h_e_' num2str(NumberOfPts) '.fig'])
        end

        plothndl=figure('Name', ...
    ['Range vs. Energy Height for Numerically Optimized Path for Case
'...
            num2str(startingpoint)]);
        plot(e,R)
        grid
```

```
        xlabel('Energy Height, e (ft)')
        ylabel('Range, R (ft)')
        if saveplots
            saveas(plothndl,['R_e_' num2str(NumberOfPts) '.fig'])
        end

        plothndl=figure('Name',[...
'Angle-of-Attack Deviation vs. Energy Height for Numerically Optimized
Path for Case ' ...
            num2str(startingpoint)]);
        plot(e_pts, d_alpha_pts)
        grid
        xlabel('Energy Height, e (ft)')
        ylabel('Angle-of-Attack Deviation, \delta\alpha (deg)')
        if saveplots
            saveas(plothndl,['da_e_' num2str(NumberOfPts) '.fig'])
        end

        plothndl=figure('Name',[...
'Dynamic Pressure vs. Energy Height for Numerically Optimized Path for
Case ' ...
            num2str(startingpoint)]);
        plot(e, qbar)
        grid
        xlabel('Energy Height (ft)')
        ylabel('Dynamic Pressure, qbar (psf)')
        if saveplots
            saveas(plothndl,['q_e_' num2str(NumberOfPts) '.fig'])
        end

        % Define a label to describe whether gamma0 is free or fixed
        if isgammafree
            gammaindicator = 'FREE';
        else
            gammaindicator = 'FIXED';
        end
    end

    % Define plot style for current starting point
    switch startingpoint
        case 1
            linecolor = [1,0.5,0]; linestyle = '--'; linewidth = 2;
        case 2
            linecolor = [0,0,1]; linestyle = ':'; linewidth = 2;
        case 3
            linecolor = [0.5,0,1]; linestyle = '-.'; linewidth = 2;
    end

    % ONLY calculate drag valleys if dragvalleyflag == true
    if dragvalleyflag
        if startingpoint == 1
            % Evaluate and plot drag, flight-path angle, angle of
attack
            %   corresponding to QEG drag valleys
```

```
            [dragcontours(:,:,1),alphacontours(:,:,1), ...
                gammacontours(:,:,1),qbarcontours(:,:,1),Vgrid,hgrid,
...
                hndl] = DragValleyNewton(300:10:V0+150, ...
                8000:1000:h0+10000,'CVQEG',true);
            [dragcontours(:,:,2),alphacontours(:,:,2), ...
                gammacontours(:,:,2),qbarcontours(:,:,2)] ...
                = DragValleyNewton(300:10:V0+150,8000:1000:h0+10000,
...
                'CDPQEG',true);

            % FIND BOTTOM OF DRAG VALLEY
            % Minimize drag along contours of constant energy height
            disp('Time to find maxima and minima of grid')
            tic
            Vmax = max(max(Vgrid)); Vmin = min(min(Vgrid));
            hmax = max(max(hgrid)); hmin = min(min(hgrid));
            toc
            emax = Vmax^2/(2*g) + hmax;
            emin = Vmin^2/(2*g) + hmin;
            evalues = linspace(emax,emin,10);
            disp('Time required to find drag valleys of both QEG
methods:')
            tic
            for i = 1:2 % for each QEG method
                % SEARCH ENERGY CONTOURS-------------------------------
----
                % Contour plot
                switch i
                    case 1
                        figure('Name','Drag Valley for CVQEG')
                    case 2
                        figure('Name','Drag Valley for CDPQEG')
                end

                % Plot drag valley
                contour(Vgrid,hgrid,dragcontours(:,:,i),200)
                hold on

                % Select number of energy contours to search along
                numcontours = 10;

                % Search along contours and find minimum drag
                lenV = (Vgrid(1,end)-Vgrid(1,1))/5+1;
                econtour_V = [];
                for j = 1:numcontours
                    econtour_V =
[econtour_V,Vgrid(1,1):5:Vgrid(1,end)];
                    econtour_h((j-1)*lenV+1:j*lenV) = ...
                        1.5*hgrid(floor(end*j/numcontours),1) ...
                        - econtour_V((j-1)*lenV+1:j*lenV).^2 / g;
                    mindrag(j,i) = inf;
                    for k = (j-1)*lenV+1:j*lenV
                        % Use bilinear interpolation to evaluate drag
at a
```

```
                        %   single pt using grid of drag valley points
                        testmin = bilininterp(Vgrid,hgrid, ...
                            dragcontours(:,:,i),econtour_V(k), ...
                            econtour_h(k),0);
                        if testmin < mindrag(j,i)
                            mindrag(j,i) = testmin;
                            mindex = k;
                        end
                    end
                    mindragV(j,i) = econtour_V(mindex);
                    mindragh(j,i) = econtour_h(mindex);
                    % Use bilinear inerpolation to find states
                    %   corresponding states to minimum drag pt
                    alpha_mindrag(j,i) = bilininterp(Vgrid,hgrid, ...
                        alphacontours(:,:,i),mindragV(j,i), ...
                        mindragh(j,i),0);
                    gamma_mindrag(j,i) = bilininterp(Vgrid,hgrid, ...
                        gammacontours(:,:,i),mindragV(j,i), ...
                        mindragh(j,i),0);
                    qbar_mindrag(j,i) = bilininterp(Vgrid,hgrid, ...

qbarcontours(:,:,i),mindragV(j,i),mindragh(j,i),0);
                end
                % Plot contours of minimum drag
                plot(econtour_V,econtour_h,'ks','MarkerSize',1)
                % Plot minimum drag pts along bottom of drag valley
                plot(mindragV(:,i),mindragh(:,i),'ro-','LineWidth',2)
                grid
                xlabel('Velocity, V (ft/s)')
                ylabel('Altitude, h (ft)')
                colorbar
                legend('Drag Contours (lbf)','Energy Contours', ...
                    'Bottom of Drag Valley','Location','Southeast')
                % SEARCH ENERGY CONTOURS------------------------------
----
            end
            toc

            % Compute energy throughout drag valleys
            mindrage = mindragh+mindragV.^2/(2*g);

            % Plot drag valleys
            for f = 0:7
                figure(hndl(1)+f)
                i = floor(f/4)+1;
                switch f
                    case {0,4}
                        plot3(mindragV(:,i),mindragh(:,i), ...
                            alpha_mindrag(:,i),'ro-','LineWidth',2)
                    case {1,5}
                        plot3(mindragV(:,i),mindragh(:,i), ...
                            gamma_mindrag(:,i),'ro-','LineWidth',2)
                    case {2,6}
                        plot3(mindragV(:,i),mindragh(:,i), ...
                            qbar_mindrag(:,i),'ro-','LineWidth',2)
```

```
                    case {3,7}
                        plot3(mindragV(:,i),mindragh(:,i), ...
                            mindrag(:,i),'ro-','LineWidth',2)
                end
            end
        end

        % Plot maximum-range trajectory on drag contour if runopt == 1
or 2
        %   Uses plot3 to show trajectories in 3D space.  Can compare
to
        %   drag valley by plotting valleys as surface plots (use
SURF).
        if runopt ~= 0
            for f = 0:7
                figure(hndl(1)+f)
                hold on
                switch f
                    case {0,4,8}
                        plot3(V,h,d_alpha_plot + alpha_star_plot2, ...
                            'Color',linecolor,'LineStyle',linestyle,
...
                            'LineWidth',2)
                    case {1,5,9}
                        plot3(V,h,gamma,'Color',linecolor,'LineStyle',
...
                            linestyle,'LineWidth',2)
                    case {2,6,10}
                        plot3(V,h,qbar,'Color',linecolor,'LineStyle',
...
                            linestyle,'LineWidth',2)
                    case {3,7,11}
                        plot3(V,h,dragtraj,'Color',linecolor, ...
                            'LineStyle',linestyle,'LineWidth',2)
                end
            end
        end

        % PLOT States along bottom of drag valleys (AND numerically
        %   optimized trajectories if calculated)
        if startingpoint == 1
            hndl2 = figure('Name', ...
       'Angle of Attack Along Maximum-Range Trajectories and Drag
Valley');
            plot(mindrage(:,1),alpha_mindrag(:,1),'bo-','LineWidth',
...
                linewidth)
            grid on
            xlabel('Energy Height, e (ft)')
            ylabel('Angle of Attack, \alpha (deg)')
            hold on
            plot(mindrage(:,2),alpha_mindrag(:,2),'rx-','LineWidth',
...
                linewidth)
        else
```

```
            figure(hndl2)
        end
        if runopt ~= 0
            plot(e, d_alpha_plot +
alpha_star_plot2,'Color',linecolor,...
                'LineStyle',linestyle,'LineWidth',linewidth)
        end

        if startingpoint == 1
            figure('Name', ...
      'Flight-Path Angle Along Maximum-Range Trajectories and Drag
Valley')
            plot(mindrage(:,1),gamma_mindrag(:,1),'bo-','LineWidth',
...
                linewidth)
            grid on
            xlabel('Energy Height, e (ft)')
            ylabel('Flight-Path Angle, \gamma (deg)')
            hold on
            plot(mindrage(:,2),gamma_mindrag(:,2),'rx-','LineWidth',
...
                linewidth)
        else
            figure(hndl2+1)
        end
        if runopt ~= 0
            plot(e, gamma, 'Color',linecolor,'LineStyle',linestyle, ...
                'LineWidth',linewidth)
        end

        if startingpoint == 1
            figure('Name', ...
       'Dynamic Pressure Along Maximum-Range Trajectories and Drag
Valley')
            plot(mindrage(:,1),qbar_mindrag(:,1),'bo-','LineWidth', ...
                linewidth)
            grid on
            xlabel('Energy Height, e (ft)')
            ylabel('Dynamic Pressure, qbar (psf)')
            hold on
            plot(mindrage(:,2),qbar_mindrag(:,2),'rx-','LineWidth', ...
                linewidth)
        else
            figure(hndl2+2)
        end
        if runopt ~= 0
            plot(e, qbar, 'Color',linecolor,'LineStyle',linestyle, ...
                'LineWidth',linewidth)
        end

        % Plot range of numerically optimized trajectory
        if runopt ~= 0
            if startingpoint == 1
                figure('Name','Range Along Maximum-Range Trajectories')
                grid on
```

```
                xlabel('Energy Height, e (ft)')
                ylabel('Range, R (ft)')
                hold on
            else
                figure(hndl2+3)
            end
            plot(e, R, 'Color',linecolor,'LineStyle',linestyle, ...
                'LineWidth',linewidth)
        end
    end

    % ADD legends to open-loop plots
    if runopt == 0 % i.e., open-loop was conducted
        figure(hndlsim)
        legend(char(modes(1)),'CVQEG','CDPQEG','Location','Southeast')
        figure(hndlsim+1)
        legend(char(modes(1)),'CVQEG','CDPQEG','Location','Southeast')
        figure(hndlsim+2)
        legend(char(modes(1)),'CVQEG','CDPQEG','Location','Southeast')
        figure(hndlsim+3)
        legend(char(modes(1)),'CVQEG','CDPQEG','Location','Southeast')
        figure(hndlsim+4)
        legend(char(modes(1)),'CVQEG','CDPQEG','Location','Northeast')
        figure(hndlsim+5)
        legend(char(modes(1)),'CVQEG','CDPQEG','Location','Northeast')
        figure(hndlsim+6)
        legend(char(modes(1)),'CVQEG','CDPQEG','Location','Southeast')
    % ELSE ADD numerically optimized trajectory to open-loop plots
    else if runopt == 2 % i.e., open-loop AND optimization are
conducted
        figure(hndlsim) % switch to 1st plot generated with RangeSim.m
        plot(e, V, 'Color',linecolor,'LineStyle',linestyle)
        legend(char(modes(1)),'CVQEG','CDPQEG',['Numerical Optimum
('...
            num2str(NumberOfPts) ' nodes, \gamma_0 is ' gammaindicator
...
            ')'],'Location','Southeast')

        figure(hndlsim+1)
        plot(e, Mach_plot,'Color',linecolor,'LineStyle',linestyle)
        legend(char(modes(1)),'CVQEG','CDPQEG',['Numerical Optimum
('...
            num2str(NumberOfPts) ' nodes, \gamma_0 is ' gammaindicator
...
            ')'],'Location','Southeast')

        figure(hndlsim+2)
        plot(e, gamma,'Color',linecolor,'LineStyle',linestyle)
        legend(char(modes(1)),'CVQEG','CDPQEG',['Numerical Optimum
('...
            num2str(NumberOfPts) ' nodes, \gamma_0 is ' gammaindicator
...
            ')'],'Location','Southeast')

        figure(hndlsim+3)
```

```
        plot(e, h,'Color',linecolor,'LineStyle',linestyle)
        legend(char(modes(1)),'CVQEG','CDPQEG',['Numerical Optimum
('...
            num2str(NumberOfPts) ' nodes, \gamma_0 is ' gammaindicator
...
            ')'],'Location','Southeast')

        figure(hndlsim+4)
        plot(e, R,'Color',linecolor,'LineStyle',linestyle)
        legend(char(modes(1)),'CVQEG','CDPQEG',['Numerical Optimum ('
...
            num2str(NumberOfPts) ' nodes, \gamma_0 is ' gammaindicator
...
            ')'],'Location','Northeast')

        figure(hndlsim+5)
        plot(e, qbar,'Color',linecolor,'LineStyle',linestyle)
        legend(char(modes(1)),'CVQEG','CDPQEG',['Numerical Optimum ('
...
            num2str(NumberOfPts) ' nodes, \gamma_0 is ' gammaindicator
...
            ')'],'Location','Northeast')

        figure(hndlsim+6)
        plot(e, d_alpha_plot + alpha_star_plot2,'Color',linecolor,...
            'LineStyle',linestyle)
        legend(char(modes(1)),'CVQEG','CDPQEG',['Numerical Optimum ('
...
            num2str(NumberOfPts) ' nodes, \gamma_0 is ' gammaindicator
...
            ')'],'Location','Southeast')

        % Compute percent improvement of optimal range over Max L/D
range
        percentOptimumAboveMaxLD = (-(f_opt*10^5)-max(Rsim(:,1))) ...
            /max(Rsim(:,1))*100
        percentOptimumAboveCVQEG = (-(f_opt*10^5)-max(Rsim(:,2))) ...
            /max(Rsim(:,2))*100
        percentOptimumAboveCDPQEG = (-(f_opt*10^5)-max(Rsim(:,3))) ...
            /max(Rsim(:,3))*100
        end
    end
end

% Add legends to plots
if dragvalleyflag
    for f = 0:7
        figure(hndl(1)+f)
        legend('Drag Contours (lbf)','Bottom of Drag Valley',...
            'Numerical Optimum (Case 1)',...
            'Numerical Optimum (Case 2)',...
            'Numerical Optimum (Case 3)','Location','Southeast')
    end
    for i = 0:2
```

```
        figure(hndl2+i)
        legend('Bottom of CVQEG Valley',...
            'Bottom of CDPQEG Valley',...
            'Numerical Optimum (Case 1)',...
            'Numerical Optimum (Case 2)',...
            'Numerical Optimum (Case 3)','Location','Southwest')
    end
    if runopt~=0
        figure(hndl2+3)
        legend('Numerical Optimum (Case 1)',...
            'Numerical Optimum (Case 2)',...
            'Numerical Optimum (Case 3)')
    end
end
```

B.2. Max-*L/D* and QEG Control System Simulations (`RangeSim.m`)

```
% Space Vehicle Range Simulation Script File
% Josiah Bryan -- MAE 8990 Masters Thesis Research
% Advised by Dr. Craig Kluever
% October 28, 2010

global V0 gamma0 h0 e0
global Vconstr gammaconstr hconstr econstr
global controlmode startingpoint
global aeroflag atmosflag
global g m S
global K_gamma lambdaCL

%  conversion
d2r = pi/180;
r2d = 1/d2r;

% Set acceleration due to gravity (assume constant)
g = 32.174;    % ft/s^2

% Switch whether certain plots are saved
saveplots = false;


modes = {'Max L/D','CVQEG','CDPQEG'}; %'Constant Dynamic Pressure'
nummodes = length(modes);
for mode = 1:3
    controlmode = char(modes(mode));

    % Select gamma0 for current control mode
    if mode > 1
        [mindrag,alpha_mindrag,gamma0] = DragValleyNewton( ...
            V0,h0,controlmode,false);
        gamma0 = gamma0 * d2r;
    end
```

```
    % Integrate trajectory using current control mode
    tic
    [Vtemp,gammatemp,Rtemp,ttemp,etemp] = Traj([]);
    toc

    len = length(Vtemp);

    if mode == 1
        % Initialize trajectory arrays
        Vsim = NaN(len,nummodes);
        gammasim = NaN(len,nummodes);
        Rsim = NaN(len,nummodes);
        tsim = NaN(len,nummodes);
        esim = NaN(len,nummodes);
        hsim = NaN(len,nummodes);
        rhosim = NaN(len,nummodes);
        asim = NaN(len,nummodes);
        Msim = NaN(len,nummodes);
        alpha_plot_sim = NaN(len,nummodes);
        qbarsim = NaN(len,nummodes);
        K_gamma = NaN(len,nummodes-1);
        dgammadotdgamma = NaN(len,nummodes-1);
        dgammadotdalpha = NaN(len,nummodes-1);
        gamma_error = NaN(len,nummodes-1);
    else
        % Expand trajectory arrays for size of current trajectory
        curlen = length(Vsim(:,1));
        if curlen ~= len
            Vsim = [Vsim; NaN(len-curlen,nummodes)];
            gammasim = [gammasim; NaN(len-curlen,nummodes)];
            Rsim = [Rsim; NaN(len-curlen,nummodes)];
            tsim = [tsim; NaN(len-curlen,nummodes)];
            esim = [esim; NaN(len-curlen,nummodes)];
            hsim = [hsim; NaN(len-curlen,nummodes)];
            rhosim = [rhosim; NaN(len-curlen,nummodes)];
            asim = [asim; NaN(len-curlen,nummodes)];
            Msim = [Msim; NaN(len-curlen,nummodes)];
            alpha_plot_sim = [alpha_plot_sim; NaN(len-
curlen,nummodes)];
            qbarsim = [qbarsim; NaN(len-curlen,nummodes)];
            K_gamma = [K_gamma; NaN(len-curlen,nummodes-1)];
        end
    end

    % Save trajectory data
    Vsim(1:len,mode) = Vtemp;
    gammasim(1:len,mode) = gammatemp;
    Rsim(1:len,mode) = Rtemp;
    tsim(1:len,mode) = ttemp;
    esim(1:len,mode) = etemp;

    % Calculate values of h along trajectory (parameterized in terms of
e)
    hsim(1:len,mode) = esim(1:len,mode) - Vsim(1:len,mode).^2 / (2*g);
        % array of h values along trajectory
```

```
    % Reconstruct control profile for max-L/D or QEG trajectory
    for i = 1:len
        % Calculate atmospheric density & Mach number
        [rhosim(i,mode), asim(i,mode), drhodh] = atmos(2, ...
            hsim(i,mode),atmosflag);
        Msim(i,mode) = Vsim(i,mode)/asim(i,mode);
        % Calculate alpha* for max-L/D
        if strcmp(controlmode,'Max L/D')
            alpha_plot_sim(i,mode) = AeroX34AlphaStar(Msim(i,mode), ...
                aeroflag);
        else % calculate QEG states for QEG
            [mindrag,alpha_mindrag,gamma_mindrag,qbar_mindrag] = ...
                DragValleyNewton(Vsim(i,mode),hsim(i,mode), ...
                controlmode,false);
            gamma_error(i,mode-1) = (gammasim(i,mode) ...
                -gamma_mindrag)*d2r; % rad

            % Constant Drag Polar (Mach 0.6)
            CL0 = 0.12502;
            CL1 = 0.051718;
            CD0 = 0.021348;
            K = 0.26647;

            % Evaluate partial derivatives
            dgammadotdgamma(i,mode-1) = g/Vsim(i,mode) ...
                *sin(gamma_mindrag*d2r); % s^-1
            dgammadotdalpha = rhosim(i,mode)*Vsim(i,mode) ...
                *S*CL1/(2*m); % s^-1
            K_gamma(i,mode-1) = (dgammadotdgamma(i,mode-1) ...
                - (lambdaCL)) / dgammadotdalpha;

            % Evaluate control law
            alpha_plot_sim(i,mode) = (alpha_mindrag*d2r - ...
                K_gamma(i,mode-1)*gamma_error(i,mode-1))*r2d; % deg
            % Limit control signal to range for which aerodynamic
            %   data is known
            if isnan(mindrag)
                alpha_plot_sim(i,mode) = 0;
            else if alpha_plot_sim(i,mode) < -6
                % i.e., if alpha outside range of known angles of
attack
                %   OR QEG algorithm did not converge
                alpha_plot_sim(i,mode) = -6;
            else if alpha_plot_sim(i,mode) > 21
                alpha_plot_sim(i,mode) = 21;
                end
                end
            end
        end
    end

    % Compute qbar for trajectory flown at Max L/D or QEG
    qbarsim(1:len,mode) = 1/2*rhosim(1:len,mode).*Vsim(1:len,mode).^2;
```

```
end

% Plot states along trajectories for max L/D, CVQEG, and CDPQEG
hndlsim=figure('Name',['Velocity vs. Energy Height for Case ' ...
    num2str(startingpoint)]);
plot(esim(:,1),Vsim(:,1),'k','LineWidth',2)
grid
xlabel('Energy Height, e (ft)')
ylabel('Velocity, V (ft/sec)')
hold on
plot(esim(:,2),Vsim(:,2),'b--','LineWidth',2)
plot(esim(:,3),Vsim(:,3),'r:','LineWidth',2)
if saveplots
    saveas(hndlsim,['V_e_maxLD.fig'])
end

plothndl=figure('Name',[...
    'Mach Number vs. Energy Height for Open-Loop Case ' ...
    num2str(startingpoint)]);
plot(esim(:,1),Msim(:,1),'k','LineWidth',2)
grid
xlabel('Energy Height, e (ft)')
ylabel('Mach Number, M')
hold on
plot(esim(:,2),Msim(:,2),'b--','LineWidth',2)
plot(esim(:,3),Msim(:,3),'r:','LineWidth',2)
if saveplots
    saveas(plothndl,['M_e_maxLD.fig'])
end

plothndl=figure('Name',['Flight-Path Angle vs. Energy Height for Case
'...
    num2str(startingpoint)]);
plot(esim(:,1),gammasim(:,1),'k','LineWidth',2)
grid
xlabel('Energy Height, e (ft)')
ylabel('Flight Path Angle, \gamma (deg)')
hold on
plot(esim(:,2),gammasim(:,2),'b--','LineWidth',2)
plot(esim(:,3),gammasim(:,3),'r:','LineWidth',2)
if saveplots
    saveas(plothndl,['g_e_maxLD.fig'])
end

plothndl=figure('Name',['Altitude vs. Energy Height for Case ' ...
    num2str(startingpoint)]);
plot(esim(:,1),hsim(:,1),'k','LineWidth',2)
grid
xlabel('Energy Height, e (ft)')
ylabel('Altitude, h (ft)')
hold on
plot(esim(:,2),hsim(:,2),'b--','LineWidth',2)
plot(esim(:,3),hsim(:,3),'r:','LineWidth',2)
if saveplots
    saveas(plothndl,['h_e_maxLD.fig'])
```

```
end

plothndl=figure('Name',['Range vs. Energy Height for Case ' ...
    num2str(startingpoint)]);
plot(esim(:,1),Rsim(:,1),'k','LineWidth',2)
grid
xlabel('Energy Height, e (ft)')
ylabel('Range, R (ft)')
hold on
plot(esim(:,2),Rsim(:,2),'b--','LineWidth',2)
plot(esim(:,3),Rsim(:,3),'r:','LineWidth',2)
if saveplots
    saveas(plothndl,['R_e_maxLD.fig'])
end

plothndl=figure('Name',['Dynamic Pressure vs. Energy Height for Case '
...
    num2str(startingpoint)]);
plot(esim(:,1), qbarsim(:,1),'k','LineWidth',2)
grid
xlabel('Energy Height, e (ft)')
ylabel('Dynamic Pressure, qbar (psf)')
hold on
plot(esim(:,2),qbarsim(:,2),'b--','LineWidth',2)
plot(esim(:,3),qbarsim(:,3),'r:','LineWidth',2)
if saveplots
    saveas(plothndl,['q_e_maxLD.fig'])
end

plothndl=figure('Name', ['Angle of Attack vs. Energy Height for Case '
...
    num2str(startingpoint)]);
plot(esim(:,1),alpha_plot_sim(:,1),'k','LineWidth',2)
grid
xlabel('Energy Height, e (ft)')
ylabel('Angle of Attack, \alpha (deg)')
hold on
plot(esim(:,2),alpha_plot_sim(:,2),'b--','LineWidth',2)
plot(esim(:,3),alpha_plot_sim(:,3),'r:','LineWidth',2)

plothndl=figure('Name', ['K_gamma vs. Energy Height for Case ' ...
    num2str(startingpoint)]);
plot(esim(:,1),K_gamma(:,1),'b--','LineWidth',2)
hold on
plot(esim(:,2),K_gamma(:,2),'r:','LineWidth',2)
grid
xlabel('Energy Height, e (ft)')
ylabel('Flight-Path Angle Gain, K_\gamma (deg/rad)')
legend('CVQEG','CDPQEG','Location','Northwest')

plothndl=figure('Name', ['Lambda_OL vs. Energy Height for Case ' ...
    num2str(startingpoint)]);
plot(esim(:,1),dgammadotdgamma(:,1),'b--','LineWidth',2)
hold on
```

```
plot(esim(:,2),dgammadotdgamma(:,2),'r:','LineWidth',2)
grid
xlabel('Energy Height, e (ft)')
ylabel('Open-Loop Eigenvalue, \lambda_O_L (s^-^1)')
legend('CVQEG','CDPQEG','Location','Southeast')

plothndl=figure('Name', ['delta_gamma vs. Energy Height for Case ' ...
    num2str(startingpoint)]);
plot(esim(:,1),gamma_error(:,1)*r2d,'b--','LineWidth',2)
hold on
plot(esim(:,2),gamma_error(:,2)*r2d,'r:','LineWidth',2)
grid
xlabel('Energy Height, e (ft)')
ylabel('Flight-Path-Angle Error, \delta\gammahat (deg)')
legend('CVQEG','CDPQEG')
```

B.3. Numerical Objective Function (`performance.m`)

```
% Space Vehicle Range Calculator Function File
% Josiah Bryan -- MAE 8990 Masters Thesis Research
% Advised by Dr. Craig Kluever
% Started September 16, 2009

function f = Performance(x)


[V,gamma,R,t,e] = Traj(x);

N = numel(V);
Rf = R(N);

f = -Rf/1e5;
```

B.4. Trajectory Setup and Integration (`traj.m`)

```
% Space Vehicle Trajectory Calculator Function File
% Josiah Bryan -- MAE 8990 Masters Thesis Research
% Advised by Dr. Craig Kluever
% Started September 16, 2009

function [V,gamma,R,t,e] = Traj(x)

global d_alpha_pts
global odestepsize
global V0 gamma0 e0 econstr
global isgammafree

%  conversion
d2r = pi/180;
```

```
r2d = 1/d2r;

% Set acceleration due to gravity (assume constant)
%g = 32.174;    % ft/s^2

% Update alpha_pts and t_pts with most recent values
if ~isempty(x) && isgammafree % USE FOR OPEN-LOOP OR IF GAMMA0 IS FREE
    d_alpha_pts = x(1:end-1);  % deg
    gamma0 = x(end);
else % USE FOR CLOSED-LOOP IF GAMMA0 IS FIXED
    d_alpha_pts = x;
end

R0 = 0; % ft
t0 = 0; % s

options = odeset('MaxStep', odestepsize);
[e, Y] = ode45(@EOM, [e0 econstr], [V0; gamma0; R0; t0], options);

% Store all trajectory data in arrays
V = Y(:,1); % ft/sec
gamma = r2d*Y(:,2); % deg
R = Y(:,3); % ft
t = Y(:,4); % s
```

B.5. Equations of Motion Evaluation (`EOM.m`)

```
% Space Vehicle Equations of Motion Calculator Function File
% Josiah Bryan -- MAE 8990 Masters Thesis Research
% Advised by Dr. Craig Kluever
% November 24, 2010

function funcdots = EOM(e, x)

global d_alpha_pts
global e_pts
global controlmode
global g m S
global aeroflag atmosflag
global isgammafree
global lambdaCL

%  conversion
d2r = pi/180;
r2d = 1/d2r;

% Current states
V = x(1); % ft/s
gamma = x(2); % rad
%R = x(3); % ft
%t = x(4); % s
```

```
% Compute altitude
h = e - V^2/(2*g); % ft

% Compute air density (lb/ft^3), speed of sound (ft/s) (NOT GIVEN in
%   exponential atmosphere), and drho/dh (lb/ft^4, rate of change of
%   density with respect to altitude)
[rho, a, drhodh] = atmos(2, h,atmosflag);

% Compute Mach number
M = V/a;

if strcmp(controlmode,'Max L/D') % if flying 'Max L/D', then find
alpha*
    % Compute alpha_star for current Mach Number
    alpha_star = AeroX34AlphaStar(M,aeroflag);

    % If optimizing (openloop == 0), then find d_alpha
    if ~isempty(d_alpha_pts)
        % Establish current angle of attack
        %  NOTE:  d_alpha_pts represents deviations from
alpha_star_pts.
        d_alpha = lininterp(e_pts,d_alpha_pts,e);
    else % if running open-loop simulation, fly at max L/D
        d_alpha = 0;
    end

    % Establish current angle of attack
    alpha = alpha_star + d_alpha;
else
    % Evaluate QEG states
    [mindrag,alpha_mindrag,gamma_mindrag] = ...
        DragValleyNewton(V,h,controlmode,false);
    gamma_error = gamma-gamma_mindrag*d2r; % rad
    %K_gamma = 0*r2d;
    CL0 = 0.12502;
    CL1 = 0.051718;
    CD0 = 0.021348;
    K = 0.26647;

    dgammadotdgamma = g/V*sin(gamma_mindrag*d2r); % s^-1
    dgammadotdalpha = rho*V*S*CL1/(2*m);        % s^-1

    K_gamma = (dgammadotdgamma - (lambdaCL)) / dgammadotdalpha;
    alpha = (alpha_mindrag*d2r - K_gamma*gamma_error)*r2d; % deg
    % Limit control signal to range for which aerodynamic data is known
    if isnan(mindrag)
        alpha = 0;
    else if alpha < -6
        % i.e., if alpha below minimum known angle of attack
        %   OR QEG algorithm did not converge
        alpha = -6;
    else if alpha > 21
        alpha = 21;
```

```
        end
        end
    end
end

% Compute aerodynamic model coefficients for the given Mach and alpha
[C_L, C_D] = AeroX34Piece(alpha, M, aeroflag);

% Compute dynamic pressure, lift, and drag
qbar = 1/2 * rho * V^2;
L = qbar * S * C_L;
D = qbar * S * C_D;

% Compute rate of change of specific energy
edot = -D*V / (m*g); % ft/s

% Compute trajectory rates--Vdot, gammadot, hdot, and Rdot
sgam = sin(gamma);
cgam = cos(gamma);
Vdot = -D/m - g*sgam; % ft/s^2
gammadot = L/(m*V) - (g/V)*cgam; % rad/s
%hdot = V*sgam; % ft/s
Rdot = V*cgam; % ft/s

% Compute dVde, dgammade, dRde, dtde
dVde = Vdot/edot; % s^-1
dgammade = gammadot/edot; % ft^-1
dRde = Rdot/edot; % non-dimensional
dtde = 1/edot; % s/ft

% Return derivative values
funcdots = [dVde; dgammade; dRde; dtde];
```

B.6. Nonlinear Constraints for Numerical Optimization (`nonlcon.m`)

```
% Space Vehicle Nonlinear Constraints Function File
% Josiah Bryan -- MAE 8990 Masters Thesis Research
% Advised by Dr. Craig Kluever
% September 16, 2009

function [c, ceq] = nonlcon(x)

global gammaconstr, global hconstr, global econstr
global g

% [V,gamma,R,t,e] = Traj(x);
%
% N = numel(V);
% Vf = V(N);
% gammaf = gamma(N);      %  deg
% Rf = R(N);
% tf = t(N);
```

```
% hf = econstr - Vf^2/(2*g); % ft

ceq = [];
c = [];
% INSERT DESIRED EQUALITY OR INEQUALITY CONSTRAINTS
% THE FOLLOWING ARE SOME EXAMPLES:
%c(1) = -Vf; % i.e., final velocity must be >= 0
%c(1) = (hf - 1e6)/1e5;
%c(1) = -min(V)/1e3; % Require that minimum velocity be >= 0
%ceq(1) = (hf - hconstr)/1e5;
%ceq(2) = gammaf - gammaconstr;
%ceq(3) = (Vf - Vdesired)/1e3;
%hlowerbound = 0; % ft
%c(2) = (-hf + hlowerbound)/1e5;
%ceq(1) = Vf - Vdesired;
%ceq(2) = hf - hdesired;
%c(3) = -Vf;
%c(4) = -hf;
```

B.7. Aerodynamic Model of X-34 (`AeroX34Piece.m`)

```
% X-34 Aerodynamics

function [CL,CD] = AeroX34Piece(alpha,Mach,flag)

% INPUTS:  alpha = current angle of attack (deg)
%           Mach = current Mach number
%               NOTE:  This data fit is only good from Mach 0.3 to 2.5.
%                      Inputs outside this range will return the
%                      aerodynamic coefficients at the nearest
available
%                      Mach number.
%           flag = selection of data source
%                  flag = 1 -> constant drag polar (Mach ~ 0.6)
%                  flag = 2 -> piecewise polynomial fit
%                  flag = 3 -> table lookup

% OUTPUTS:  CL = current lift coefficient
%           CD = current drag coefficient

if flag == 1
    CL0 = 0.12502;
    CL1 = 0.051718;
    CD0 = 0.021348;
    K = 0.26647;
    CL = CL0 + CL1*alpha; % from Mach = 0.6
    CD = CD0 + K*CL.^2; % from Mach = 0.6
else
    % Limit Mach numbers to within range of data fit
    if Mach > 2.5
        Mach = 2.5;
    else if Mach < 0.3
```

```
        Mach = 0.3;
    end
    end

    if flag == 3
        global Alphatable
        global Machtable
        global CLtable
        global CDtable
        CL = interp2(Alphatable,Machtable,CLtable,alpha,Mach);
        CD = interp2(Alphatable,Machtable,CDtable,alpha,Mach);
    else
    % THIRD-ORDER PIECEWISE FIT FOR DRAG COEFFICIENT (4th FOR MACH =
0.3)
        if (Mach >= 1.05)
            if (Mach >= 1.25)
                if (Mach >= 1.6)
                    if (Mach >= 2 && Mach <= 2.5)
                        CL = (2.5-Mach)/0.5.*(0.003173159328788 ...
                            +0.038331392819444*alpha ...
                            -0.000009417002104*alpha.^2) ...
                            +(Mach-2)/0.5.*(-0.012786233018182 ...
                            +0.031831064522222*alpha ...
                            +0.000096250436027*alpha.^2);
                        CD = (2.5-Mach)/0.5.*(0.078706070395105 ...
                            -0.000903218151968*alpha ...
                            +0.000641015107097*alpha.^2 ...
                            +0.000001960614226*alpha.^3) ...
                            +(Mach-2)/0.5.*(0.070740798759907 ...
                            -0.001332361985755*alpha ...
                            +0.000536386394328*alpha.^2 ...
                            +0.000004128794109*alpha.^3);
                    else if (Mach >= 1.8)
                        CL = (2-Mach)/0.2.*(0.013758203596970 ...
                            +0.041599119068182*alpha ...
                            -0.000300457564815*alpha.^2) ...
                            +(Mach-1.8)/0.2.*(0.003173159328788 ...
                            +0.038331392819444*alpha ...
                            -0.000009417002104*alpha.^2);
                        CD = (2-Mach)/0.2.*(0.083421829161306 ...
                            -0.000709322583269*alpha ...
                            +0.000690870205840*alpha.^2 ...
                            +0.000001618017993*alpha.^3) ...
                            +(Mach-1.8)/0.2.*(0.078706070395105 ...
                            -0.000903218151968*alpha ...
                            +0.000641015107097*alpha.^2 ...
                            +0.000001960614226*alpha.^3);
                    else % Mach >= 1.6
                        CL = (1.8-Mach)/0.2.*(0.027760065866667 ...
                            +0.045391187575758*alpha ...
                            -0.000029742639731*alpha.^2) ...
                            +(Mach-1.6)/0.2.*(0.013758203596970 ...
                            +0.041599119068182*alpha ...
                            -0.000300457564815*alpha.^2);
                        CD = (1.8-Mach)/0.2.*(0.087639235368764 ...
```

```
                    -0.000462916978762*alpha ...
                    +0.000743394199171*alpha.^2 ...
                    +0.000001962722237*alpha.^3) ...
                    +(Mach-1.6)/0.2.*(0.083421829161306 ...
                    -0.000709322583269*alpha ...
                    +0.000690870205840*alpha.^2 ...
                    +0.000001618017993*alpha.^3);
            end
            end
        else if (Mach >= 1.4)
            CL = (1.6-Mach)/0.2.*(0.062388126543939 ...
                +0.050273437591667*alpha ...
                -0.000110606575337*alpha.^2) ...
                +(Mach-1.4)/0.2.*(0.027760065866667 ...
                +0.045391187575758*alpha ...
                -0.000029742639731*alpha.^2);
            CD = (1.6-Mach)/0.2.*(0.093461532051049 ...
                +0.000102607718531*alpha ...
                +0.000797274776353*alpha.^2 ...
                +0.000000850082125*alpha.^3) ...
                +(Mach-1.4)/0.2.*(0.087639235368764 ...
                -0.000462916978762*alpha ...
                +0.000743394199171*alpha.^2 ...
                +0.000001962722237*alpha.^3);
        else % Mach >= 1.25
            CL = (1.4-Mach)/0.15.*(0.098344894300000 ...
                +0.056138175607576*alpha ...
                -0.000267505962121*alpha.^2) ...
                +(Mach-1.25)/0.15.*(0.062388126543939 ...
                +0.050273437591667*alpha ...
                -0.000110606575337*alpha.^2);
            CD = (1.4-Mach)/0.15.*(0.094370517387179 ...
                +0.000742240085794*alpha ...
                +0.000921010922947*alpha.^2 ...
                -0.000001595740777*alpha.^3) ...
                +(Mach-1.25)/0.15.*(0.093461532051049 ...
                +0.000102607718531*alpha ...
                +0.000797274776353*alpha.^2 ...
                +0.000000850082125*alpha.^3);
        end
        end
    else if (Mach >= 1.1)
        CL = (1.25-Mach)/0.15.*(0.139602805190909 ...
            +0.060988223715657*alpha ...
            -0.000476656287037*alpha.^2) ...
            +(Mach-1.1)/0.15.*(0.098344894300000 ...
            +0.056138175607576*alpha ...
            -0.000267505962121*alpha.^2);
        CD = (1.25-Mach)/0.15.*(0.094513090939860 ...
            +0.001515935925408*alpha ...
            +0.001020064533411*alpha.^2 ...
            -0.000005100002720*alpha.^3)...
            +(Mach-1.1)/0.15.*(0.094370517387179 ...
            +0.000742240085794*alpha ...
            +0.000921010922947*alpha.^2 ...
```

```
                -0.000001595740777*alpha.^3);
        else % Mach >= 1.05
            CL = (1.1-Mach)/0.05.*(0.147494721081818 ...
                +0.062149226147980*alpha ...
                -0.000554612324074*alpha.^2) ...
                +(Mach-1.05)/0.05.*(0.139602805190909 ...
                +0.060988223715657*alpha ...
                -0.000476656287037*alpha.^2);
            CD = (1.1-Mach)/0.05.*(0.095795336369464 ...
                +0.001822186134097*alpha ...
                +0.001020206368816*alpha.^2 ...
                -0.000005670559620*alpha.^3) ...
                +(Mach-1.05)/0.05.*(0.094513090939860 ...
                +0.001515935925408*alpha ...
                +0.001020064533411*alpha.^2 ...
                -0.000005100002720*alpha.^3);
        end
        end
    else if (Mach >= 0.8)
        if (Mach >= 0.95)
            CL = (1.05-Mach)/0.1.*(0.124584850103030 ...
                +0.059382875053535*alpha ...
                -0.000448020641414*alpha.^2) ...
                +(Mach-0.95)/0.1.*(0.147494721081818 ...
                +0.062149226147980*alpha ...
                -0.000554612324074*alpha.^2);
            CD = (1.05-Mach)/0.1.*(0.059728821099533 ...
                +0.001490800524929*alpha ...
                +0.001061648501943*alpha.^2 ...
                -0.000007766471560*alpha.^3) ...
                +(Mach-0.95)/0.1.*(0.095795336369464 ...
                +0.001822186134097*alpha ...
                +0.001020206368816*alpha.^2 ...
                -0.000005670559620*alpha.^3);
        else if (Mach >= 0.9)
            CL = (0.95-Mach)/0.05.*(0.144781904715151 ...
                +0.054418479921212*alpha ...
                -0.000412454754209*alpha.^2) ...
                +(Mach-0.9)/0.05.*(0.124584850103030 ...
                +0.059382875053535*alpha ...
                -0.000448020641414*alpha.^2);
            CD = (0.95-Mach)/0.05.*(0.047506354107227 ...
                +0.001390132320254*alpha ...
                +0.000984474559959*alpha.^2 ...
                -0.000006168948905*alpha.^3) ...
                +(Mach-0.9)/0.05.*(0.059728821099533 ...
                +0.001490800524929*alpha ...
                +0.001061648501943*alpha.^2 ...
                -0.000007766471560*alpha.^3);
        else % Mach >= 0.8
            CL = (0.9-Mach)/0.1.*(0.150403423575758 ...
                +0.056201102838384*alpha ...
                -0.000484386868687*alpha.^2) ...
                +(Mach-0.8)/0.1.*(0.144781904715151 ...
                +0.054418479921212*alpha ...
```

```
            -0.000412454754209*alpha.^2);
        CD = (0.9-Mach)/0.1.*(0.034037732790676 ...
            +0.000954079710697*alpha ...
            +0.001005127555685*alpha.^2 ...
            -0.000006361458704*alpha.^3) ...
            +(Mach-0.8)/0.1.*(0.047506354107227 ...
            +0.001390132320254*alpha ...
            +0.000984474559959*alpha.^2 ...
            -0.000006168948905*alpha.^3);
    end
    end
else if (Mach >= 0.6)
    CL = (0.8-Mach)/0.2.*(0.125843609978788 ...
        +0.052403844978283*alpha ...
        -0.000045702633838*alpha.^2) ...
        +(Mach-0.6)/0.2.*(0.150403423575758 ...
        +0.056201102838384*alpha ...
        -0.000484386868687*alpha.^2);
    CD = (0.8-Mach)/0.2.*(0.032454385122843 ...
        -0.001054567672688*alpha ...
        +0.000771121311383*alpha.^2 ...
        +0.000009174033972*alpha.^3) ...
        +(Mach-0.6)/0.2.*(0.034037732790676 ...
        +0.000954079710697*alpha ...
        +0.001005127555685*alpha.^2 ...
        -0.000006361458704*alpha.^3);
else if (Mach >= 0.4)
    CL = (0.6-Mach)/0.2.*(0.111777889442424 ...
        +0.047685938183838*alpha ...
        +0.000242495228956*alpha.^2) ...
        +(Mach-0.4)/0.2.*(0.125843609978788 ...
        +0.052403844978283*alpha ...
        -0.000045702633838*alpha.^2);
    CD = (0.6-Mach)/0.2.*(0.037050781616783 ...
        -0.001684552200466*alpha ...
        +0.000558370276871*alpha.^2 ...
        +0.000018776594578*alpha.^3) ...
        +(Mach-0.4)/0.2.*(0.032454385122843 ...
        -0.001054567672688*alpha ...
        +0.000771121311383*alpha.^2 ...
        +0.000009174033972*alpha.^3);
else if (Mach >= 0.3)
    CL = (0.4-Mach)/0.1.*(0.120243583681818 ...
        +0.044815973815657*alpha ...
        +0.000305253051347*alpha.^2) ...
        +(Mach-0.3)/0.1.*(0.111777889442424 ...
        +0.047685938183838*alpha ...
        +0.000242495228956*alpha.^2);
    % FOURTH-ORDER AT MACH = 0.3
    CD = (0.4-Mach)/0.1.*(0.031926423286132 ...
        +0.001057762779073*alpha ...
        +0.000635757419176*alpha.^2 ...
        -0.000026799605697*alpha.^3 ...
        +0.000001769082224*alpha.^4)...
        +(Mach-0.3)/0.1.*(0.037050781616783 ...
```

```
                -0.001684552200466*alpha ...
                +0.000558370276871*alpha.^2 ...
                +0.000018776594578*alpha.^3);
        end
        end
        end
        end
        end
    end
end
```

B.8. Angle of Attack for Max *L/D* (`AeroX34AlphaStar.m`)

```
% Space Vehicle AlphaStar vs. Mach Number Profile
% Josiah Bryan -- MAE 8990 Masters Thesis Research
% Advised by Dr. Craig Kluever
% October 27, 2010

function alphastar = AeroX34AlphaStar(M,aeroflag)

% Input:  M = Mach Number
%       NOTE:  This data fit is only good from Mach 0.3 to 2.5.
%              Inputs outside this range will return the
%              aerodynamic coefficients at the nearest available
%              Mach number.
%         flag = chooses aerodynamic model; 1 = constant drag polar;
%              2 = piecewise polynomial model;
% Output: alphastar = angle of attack for maximum L/D (degrees)

if aeroflag == 1
    % Assumes Mach ~ 0.6
    CL0 = 0.125020962569697;
    CL1 = 0.051718305470707;
    CD0 = 0.021348483786708;
    K = 0.266467732258516;
    alphastar = (sqrt(CD0/K)-CL0)/CL1;
else
    % Limit Mach numbers to within range of data fit
    if M > 2.5
        M = 2.5;
    else if M < 0.3
        M = 0.3;
    end
    end

    % Evaluate angle of attack for maximum lift-to-drag ratio
    if (M >= 1.05)
        if (M >= 1.25)
            if (M >= 1.6)
                if (M >= 2 && M <= 2.5)
                    alphastar = 1.25*M + 8.135;
                else if (M >= 1.8)
```

```
                    alphastar = 1.435*M + 7.765;
                else % M >= 1.6
                    alphastar = 2.005*M + 6.739;
                end
                end
            else if (M >= 1.4)
                alphastar = 3.095*M + 4.995;
            else % M >= 1.25
                alphastar = 7.753333333333306*M - 1.526666666666632;
            end
            end
        else if (M >= 1.1)
            alphastar = 6.853333333333348*M - 0.401666666666683;
        else % M >= 1.05
            alphastar = 2.14*M + 4.783;
        end
        end
    else if (M >= 0.8)
        if (M >= 0.95)
            alphastar = 16.16*M - 9.938;
        else if (M >= 0.9)
            alphastar = 19.82*M - 13.415;
        else % M >= 0.8
            alphastar = 9.76*M - 4.361;
        end
        end
    else if (M >= 0.6)
        alphastar = -5.045*M + 7.483;
    else if (M >= 0.4)
        alphastar = -5.66*M + 7.852;
    else if (M >= 0.3)
        alphastar = -0.59*M + 5.824;
    end
    end
    end
    end
    end
end
```

B.9. Atmospheric Model (`atmos.m`)

The majority of this file provided by Kluever [26]:

```
%
%    atmos.m
%
%  >> [rho,a,drhodh] = atmos(units,h,atmosflag)
%
%  atmos.m computes density (rho), rate of change of density with
altitude
%  (drhodh), and speed of sound (a).  Uses temperature gradients
(defined
%  pp. 14-19, Nelson)
%
```

```
%  Input:  units = unit flag for input and output:
%               1=SI units; 2=English units
%          h = altitude above sea level (m or ft; depends on units
flag)
%          atmosflag = flag for atmospheric model:
%               1=exponential; 2=piecewise thermal; 3=atmos_1976.m
%
%  Output:  rho = atmospheric density, kg/m^3 OR slugs/ft^3
%               (depends on units flag)
%           a = speed of sound, m/s or ft/s (depend on units flag)
%           drhodh = rate of change of density with altitude, kg/m^4 OR
%               slugs/ft^4 (depends on units flag)
%
%  Note: this Mfile performs all computations in SI
%
%  Modified:  7 March 2011 by Josiah Bryan (added drhodh)

function [rho,a,drhodh] = atmos(units,h,atmosflag)

switch atmosflag
    case 1
        if units == 2
            beta = 1/30500; % ft^-1
            rho0 = 0.00237717347417292; % slug/ft^3 @ sea level
            rho = rho0*exp(-beta*h); % slug/ft^3
            drhodh = -beta*rho; % slug/ft^4
            a = 0; % SHOULD NOT BE NEEDED IF CONSTANT DRAG POLAR
            return
        else
            error('UNITS MISMATCH')
        end
    case 3
        if units == 2
            [temp,a,press,rho,drhodh]=atmos_1976(h);
            return
        else
            error('UNITS MISMATCH')
        end
end


%  constants
ft2m = 0.3048;
m2ft = 1/ft2m;
rho_SL = 1.225;             %  density at sea level, kg/m^3
R = 287;                        %  m^2/K-s^2
gamma = 1.400253219;
g0 = 9.806;

%  temp gradient boudaries (deg K)
T_k = [ 288.15 216.65 216.65 228.65 270.65 ];

%  associated density break-pts (kg/m^3)
rho_k = [ rho_SL 3.639e-1 8.803e-2 1.332e-2 1.427e-3 ];
```

```
%  lapse rate (K/m)
lam_k = [ -6.5e-3 0 1e-3 2.8e-3 0 ];

%  alt break pts (m)
h_k = [ 0 11e3 20e3 32e3 47e3 ];

%  default values (SI units)
rho1 = rho_k(1);
T1 = T_k(1);
h1 = h_k(1);
lam = lam_k(1);

if units > 1
   h = h*ft2m;          %  altitude must be in m for logic...
end

%  find break pts and lapse rate
for i=1:5
    if h >= h_k(i)
      rho1 = rho_k(i);
      T1 = T_k(i);
      h1 = h_k(i);
      lam = lam_k(i);
   end
end

dh = h - h1;

%  linear Temp region
if lam ~= 0
   T = T1 + lam*dh;
   rho_ratio = (T/T1)^-(1 + g0/(R*lam) );
   rho = rho1*rho_ratio;
   drhodh = rho1*-(1 + g0/(R*lam))*(T/T1)^(-(1 + g0/(R*lam)) ...
       - 1)/T1*lam; % drhodh = drhodT * dTdh
else
   % isothermal region
   T = T1;
   rho = rho1*exp( -g0*dh/(R*T1) );
   drhodh = rho1*(-g0/(R*T1))*exp( -g0*dh/(R*T1) );
end

%  speed of sound
a = sqrt(gamma*R*T);

%  convert output to English (if desired)
if units > 1                              %  English units
   rho = rho*2.3769e-3/rho_SL;
   drhodh = drhodh*2.3769e-3/rho_SL/100*2.54*12;
   a = a*m2ft;
end
```

B.10. QEG States and Drag Valley (`DragValleyNewton.m`)

```
% Drag Contour Calculator
% Josiah Bryan -- MAE 8990 Masters Thesis Research
% Advised by Dr. Craig Kluever
% Started February 28, 2011

function [dragcontours,alphacontours,gammacontours,qbarcontours, ...
    Vgrid,hgrid,hndl] = DragValleyNewton(Vvalues,hvalues,controlmode,
...
    plotflag)

% INPUT:  Vvalues = values of velocity for which to evaluate drag
valley
%                   (vector), ft/s
%         hvalues = values of altitude for which to evaluate drag
valley
%                   (vector), ft
%     controlmode = 'CVQEG', 'CDPQEG', or 'QEG'
%        plotflag = true -> print out plots;
%                   false -> do not print out plots
%
% OUTPUT: dragcontours = matrix of drag values associated with QEG
flight
%                        at inputted Vvalues and hvalues, lbf
%                        NOTE:  This and all other contour variables
will
%                        equal NaN if the search algorithm (Newton-
Raphson)
%                        does not converge.
%        alphacontours = matrix of angles of attack associated with QEG
%                        flight at inputted Vvalues and hvalues, deg
%        gammacontours = matrix of flight-path angles associated with
QEG
%                        flight at inputted Vvalues and hvalues, deg
%         qbarcontours = matrix of dynamic pressures associated with
QEG
%                        flight at inputted Vvalues and hvalues, psf
%                Vgrid = matrix of velocities associated with contour
%                        matrices, ft/s
%                hgrid = matrix of altitudes associated with contour
%                        matrices, ft
%                 hndl = vector of handles to figures created

global g, global m, global S
global aeroflag, global atmosflag

%  conversion
d2r = pi/180;
r2d = 1/d2r;

% Define grid of velocities and altitudes for energy space
[Vgrid,hgrid] = meshgrid(Vvalues,hvalues);
```

```
% Set acceleration due to gravity (assume constant)
g = 32.174;    % ft/s^2

% Aircraft variables
m = 18000/g;   % slugs   (18000 lb, X-34)
S = 357.5; % ft^2

% Initialize maxVdot, maxqbardot, and maxgammadot
maxVdot = 0;
maxqbardot = 0;
maxgammadot = 0;

% Initialize drag contours
lenhvalues = length(hvalues);
lenVvalues = length(Vvalues);
dragcontours = NaN(lenhvalues,lenVvalues);
alphacontours = NaN(lenhvalues,lenVvalues);
gammacontours = NaN(lenhvalues,lenVvalues);
qbarcontours = NaN(lenhvalues,lenVvalues);

% Cycle through h- and V-values in energy space
for i = 1:lenhvalues % length(hvalues)
    if plotflag
        disp(['% of Altitudes Complete:  ' ...
            num2str(i/lenhvalues*100) ' (i = ' num2str(i) ')'])
    end

    for j = 1:lenVvalues % length(Vvalues)
        [rho_temporary, a_temporary, drhodh_temporary] = ...
            atmos(2,hvalues(i),atmosflag);
        M_temporary = Vvalues(j) / a_temporary;

        % Guess angle of attack (start with max L/D value)
        alpha_guess = AeroX34AlphaStar(M_temporary,aeroflag); % deg
        gamma_guess = -6*d2r; % rad

        % Initialize current and previous guess vectors
        xnew = [alpha_guess;gamma_guess];
        xprev = [-100;-100];

        % Initialize convergence error (in alpha and gamma)
        alpha_error = 100; % deg
        gamma_error = 100; % rad

        % Check velocity convergence error
        alpha_crit = 0.001; % deg
        gamma_crit = 0.001*d2r; % rad
        numtries = 0;
        while (alpha_error > alpha_crit || gamma_error > gamma_crit)
...
                && numtries < 60
```

```
            % Update guess
            alpha_guess = xnew(1); gamma_guess = xnew(2);
            % Calculate alpha and gamma convergence error
            alpha_error = abs(alpha_guess-xprev(1));
            gamma_error = abs(gamma_guess-xprev(2));

            % Calculate CL, CD
            if aeroflag == 1
                % CONSTANT DRAG POLAR
                CL0 = 0.12502;
                CL1 = 0.051718;
                CD0 = 0.021348;
                K = 0.26647;
                CL_temporary = CL0 + CL1*alpha_guess; % from Mach = 0.6
                CD_temporary = CD0 + K*CL_temporary^2; % from Mach =
0.6
            else
                [CL_temporary,CD_temporary] = ...
                    AeroX34Piece(alpha_guess,M_temporary,aeroflag);
            end

            if ~isreal(CL_temporary) || ~isreal(CD_temporary)
                break
            end

            % Calculate sine and cosine of gamma
            sgam = sin(gamma_guess);
            cgam = cos(gamma_guess);

            % Evaluate (Vdot or qbardot) and gammadot
            qbar = 1/2*rho_temporary*Vvalues(j)^2; % psf
            D = qbar*S*CD_temporary; % lbf
            Vdot = -D/m-g*sgam; % ft/s^2
            qbardot = 1/2*drhodh_temporary*Vvalues(j)^3*sgam ...
                + rho_temporary*Vvalues(j)*Vdot; % psf/s
            gammadot = (qbar*S*CL_temporary) ...
                /(m*Vvalues(j))-(g/Vvalues(j))*cgam; % rad/s

            % Check if criteria are met
            if alpha_error <= alpha_crit && gamma_error <= gamma_crit
...
                    && alpha_guess <= 21 && alpha_guess >= -6
                %************SAVE VALUE OF
DRAG*****************************
                dragcontours(i,j) = D; % lbf

%*********************************************************
                %************SAVE VALUE OF
ALPHA****************************
                alphacontours(i,j) = rem(alpha_guess,360); % deg

%*********************************************************
                %************SAVE VALUE OF
GAMMA****************************
```

```
                gammacontours(i,j) = rem(gamma_guess*r2d,360); % deg

%*********************************************************
                %************SAVE VALUE OF
QBAR*****************************
                qbarcontours(i,j) = qbar; % psf

%*********************************************************
                if abs(Vdot) > abs(maxVdot)
                    maxVdot = Vdot;
                end
                if abs(qbardot) > abs(maxqbardot)
                    maxqbardot = qbardot;
                end
                if abs(gammadot) > abs(maxgammadot)
                    maxgammadot = gammadot;
                end
            else % IF CRITERIA NOT MET, THEN UPDATE PREVIOUS GUESS
                % Update previous guess
                xprev = [alpha_guess;gamma_guess]; % deg;rad

                if aeroflag == 1 % constant drag polar
                                 %    --> compute analytical Jacobian
                    % Jacobian partials of gammadot
                    J(2,1) = 1/2*rho_temporary*Vvalues(j)*S*CL1/m;
                        % dgammadot_dalpha, rad/s/deg
                    J(2,2) = g/Vvalues(j)*sgam;
                        % dgammadot_dgamma, rad/s/rad
                else % NOT constant drag polar -->
                     %    compute numerical Jacobian with
                     %    finite-difference method
                    alpha_pertsize = 0.00001; % deg (perturbation size)
                    gamma_pertsize = 0.00001*d2r; % rad

                    sgam_pert = sin(gamma_guess+gamma_pertsize);
                    cgam_pert = cos(gamma_guess+gamma_pertsize);

                    [CL_pert,CD_pert] = ...
                        AeroX34Piece(alpha_guess+alpha_pertsize,...
                        M_temporary,aeroflag);
                    gammadot_pert = (qbar*S*CL_pert)...
                        /(m*Vvalues(j))-(g/Vvalues(j))*cgam; % rad/s

                    J(2,1) = (gammadot_pert - gammadot) ...
                        / alpha_pertsize; % dgammadot_dalpha, rad/s/deg

                    gammadot_pert = (qbar*S*CL_temporary)...
                        /(m*Vvalues(j))-(g/Vvalues(j))*cgam_pert; %
rad/s

                    J(2,2) = (gammadot_pert - gammadot) ...
                        / gamma_pertsize; % dgammadot_dgamma, rad/s/rad
                end
```

```
                % Calculate Jacobian for Newton-Raphson Method
                switch controlmode
                    case 'CVQEG' % constant velocity (Vdot = 0)
                        if aeroflag == 1 % Compute analytical Jacobian
                            % Jacobian partials of Vdot
                            J(1,1) = -rho_temporary*Vvalues(j)^2 ...
                                *S*K*CL_temporary*CL1/m;
                                    % dVdot_dalpha, ft/s/deg
                            J(1,2) = -g*cgam; % dVdot_dgamma, ft/s/rad
                        else % Compute numerical Jacobian with
                             %    finite-difference method
                            % Jacobian partials of Vdot
                            D_pert = qbar*S*CD_pert; % lbf
                            Vdot_pert = -D_pert/m-g*sgam; % ft/s^2
                            J(1,1) = (Vdot_pert - Vdot) ...
                                / alpha_pertsize; % dVdot_dalpha,
ft/s/deg

                            Vdot_pert = -D/m-g*sgam_pert; % ft/s^2
                            J(1,2) = (Vdot_pert - Vdot) ...
                                / gamma_pertsize; % dVdot_dgamma,
ft/s/rad
                        end
                        % Calculate next guess with Newton-Raphson
Method
                        xnew = xprev - J \ [Vdot;gammadot];
                    case 'CDPQEG' % constant dynamic pressure (qbardot
= 0)
                        if aeroflag == 1 % Compute analytical Jacobian
                            % Jacobian partials of qbardot
                            J(1,1) = -rho_temporary^2*Vvalues(j)^3 ...
                                *S*K*CL_temporary*CL1/m;
                                    % dqbardot_dalpha, psf/s/deg
                            J(1,2) = (1/2*drhodh_temporary ...
                                *Vvalues(j)^3-rho_temporary ...
                                *Vvalues(j)*g)*cgam;
                                    % dqbardot_dgamma, psf/s/rad
                        else % Compute numerical Jacobian with
                             %    finite-difference method
                            % Jacobian partials of qbardot
                            D_pert = qbar*S*CD_pert; % lbf
                            Vdot_pert = -D_pert/m-g*sgam; % ft/s^2
                            qbardot_pert = 1/2*drhodh_temporary ...
                                *Vvalues(j)^3*sgam + rho_temporary ...
                                *Vvalues(j)*Vdot_pert; % psf/s
                            J(1,1) = (qbardot_pert - qbardot) ...
                                / alpha_pertsize;
                                    % dqbardot_dalpha, psf/s/deg

                            Vdot_pert = -D/m-g*sgam_pert; % ft/s^2
                            qbardot_pert = 1/2*drhodh_temporary ...
                                *Vvalues(j)^3*sgam_pert ...
                                + rho_temporary*Vvalues(j)*Vdot_pert;
                                    % psf/s
                            J(1,2) = (qbardot_pert - qbardot) ...
```

```
                                / gamma_pertsize;
                                    % dqbardot_dgamma, psf/s/rad
                            end
                            % Calculate next guess with Newton-Raphson
Method
                            xnew = xprev - J \ [qbardot;gammadot];
                    end

                    % Update number of tries
                    numtries = numtries + 1;
                end
            end
        end
end

% PLOT DRAG SURFACE AND VALLEY IF
if plotflag % i.e., if supposed to print out plots
    % Report Vdot, qbardot, and gammadot with maximum absolute values
    maxVdot
    maxqbardot
    maxgammadot

    % Plot drag valley
    hndl(1) = figure('Name',['Angles-of-Attack in V-H Space for ' ...
        controlmode]);
    %surf(Vgrid,hgrid,alphacontours) % FOR SURFACE INSTEAD OF CONTOUR
    contour(Vgrid,hgrid,alphacontours,150)
    xlabel('Velocity, V (ft/s)')
    ylabel('Altitude, h (ft)')
    zlabel('Angle-of-Attack, \alpha (deg)')
    grid
    hold on
    legend(['Drag Contours (lbf)'],'Location','Southeast')
    colorbar

    hndl(2) = figure('Name',['Flight-Path Angle in V-H Space for ' ...
        controlmode]);
    %surf(Vgrid,hgrid,gammacontours)
    contour(Vgrid,hgrid,gammacontours,150)
    xlabel('Velocity, V (ft/s)')
    ylabel('Altitude, h (ft)')
    zlabel('Flight-Path Angle, \gamma (deg)')
    grid
    hold on
    legend(['Drag Contours (lbf)'],'Location','Southeast')
    colorbar

    hndl(3) = figure('Name',['Dynamic Pressure in V-H Space for ' ...
        controlmode]);
    %surf(Vgrid,hgrid,qbarcontours)
    contour(Vgrid,hgrid,qbarcontours,150)
    xlabel('Velocity, V (ft/s)')
    ylabel('Altitude, h (ft)')
    zlabel('Dynamic Pressure, qbar (psf)')
```

```
    grid
    hold on
    legend(['Drag Contours (lbf)'],'Location','Southeast')
    colorbar

    hndl(4) = figure('Name',['Drag Valley in V-H Space for ' ...
        controlmode]);
    %surf(Vgrid,hgrid,dragcontours)
    contour(Vgrid,hgrid,dragcontours,150)
    xlabel('Velocity, V (ft/s)')
    ylabel('Altitude, h (ft)')
    zlabel('Drag, D (lbf)')
    grid
    hold on
    legend(['Drag Contours (lbf)'],'Location','Southeast')
    colorbar
else
    hndl = []; % no new figure handles
end
```

B.11. Fast Linear Interpolation (`lininterp.m`)

```
% Linear Interpolation
% Josiah Bryan -- MAE 8990 Masters Thesis Research
% Advised by Dr. Craig Kluever
% April 5, 2011

function [y,i] = lininterp(xvalues,yvalues,x)

% INPUTS:  xvalues = monotonically increasing vector of x-values
%          yvalues = corresponding vector of y-values
%                x = x-value for which y-value is desired
%
% OUTPUTS:       y = linearly interpolated y-value corresponding to
%                   inputted x
%            i = index of x-value (from xvalues) at nearest point below
%                   inputted x, plus weighting factor indicating how
%                   far between the two neighboring values x is
%                   e.g., If x = 5 and xvalues = [4 8], then i = 1.25
%                NOTE: If x is outside range of xvalues, i equals the
%                         index of the x-value nearest to x.
%                   e.g., If x = 0.5 and xvalues = [1 2], then i = 1
%                         If x = 10  and xvalues = [1 2], then i = 2

len = length(yvalues);

% IF current x is within range of known x data
if (x > xvalues(1) && x < xvalues(len))
    % LINEAR INTERPOLATION -- NOTE:  Other interpolation functions
    % might be used for better results

    % Find index of nearest lower data point
    j = len-1;
    xdiff = x-xvalues(j);
```

```
    while j > 1 && xdiff < 0
        j = j-1;
        xdiff = x-xvalues(j);
    end

    idiff = xdiff / (xvalues(j+1)-xvalues(j));
    y = yvalues(j) + (yvalues(j+1)-yvalues(j)) * idiff;
    i = j + idiff;
else if (x <= xvalues(1))
     % (i.e., if current x is less than first x-value,
     %      then let y = first y-value)
    y = yvalues(1);
    i = 1;
else
     % (i.e., if current x is greater than last x-value,
     %      then let y = last y-value)
    y = yvalues(len);
    i = len;
    end
end
```

B.12. Fast Bilinear Interpolation (`bilininterp.m`)

```
% Bilinear Interpolation
% Josiah Bryan -- MAE 8990 Masters Thesis Research
% Advised by Dr. Craig Kluever
% July 27, 2011

function [z,ix,iy,grad] = bilininterp(xgrid,ygrid,zgrid,x,y,flag)

% INPUTS:  xgrid = array of constant-increment x-values as generated by
%                  meshgrid
%          ygrid = array of constant-increment y-values as generated by
%                  meshgrid
%          zgrid = array of z-values corresponding to xgrid and ygrid
%              x = x-value for which z-value is desired
%              y = y-value for which z-value is desired
%           flag = indicator of whether to output interpolated gradient
%                  0 -> no gradient; 1 -> output interpolated gradient
%
% OUTPUTS:     z = bilinearly interpolated z-value corresponding to
%                  inputted x and y
%            ix = x-index of x-value at nearest grid point below
%                  inputted x (unless x is less than lowest x-value,
%                  then ix = 1)
%            iy = y-index of y-value at nearest grid point below
%                  inputted y (unless y is less than lowest y-value,
%                  then iy = 1)
%      grad = approximation of gradient at point [x,y] found by taking
%                analytical gradient of bilinear interpolation formula

% Find step size
```

```
xstep = xgrid(1,2)-xgrid(1,1); ystep = ygrid(2,1)-ygrid(1,1);

% Find ix and iy-----------------------------------------------------------
% Find size of x-data and y-data
xlen = size(xgrid,2);
ylen = size(ygrid,1);

% IF current x is within range of x data
if (x > xgrid(1,1) && x < xgrid(1,xlen))
    % Find index of nearest lower data point
    ix1 = xlen-1;
    dif = x-xgrid(1,ix1);
    while ix1 > 1 && dif < 0
        ix1 = ix1-1;
        dif = x-xgrid(1,ix1);
    end
    ix = ix1;
else if (x <= xgrid(1,1))
     % (i.e., if current x is less than or equal to first x-value)
    ix1 = 1;
    ix = 1;
    % Limit x to boundary
    x = xgrid(1,1);
else
     % (i.e., if current x is greater than or equal to last x-value)
    ix1 = xlen-1;
    ix = xlen;
    % Limit x to boundary
    x = xgrid(1,xlen);
    end
end

% IF current y is within range of y data
if (y > ygrid(1,1) && y < ygrid(ylen,1))
    % Find index of nearest lower data point
    iy1 = ylen-1;
    dif = y-ygrid(iy1,1);
    while iy1 > 1 && dif < 0
        iy1 = iy1-1;
        dif = y-ygrid(iy1,1);
    end
    iy = iy1;
else if (y <= ygrid(1,1))
     % (i.e., if current y is less than or equal to first y-value)
    iy1 = 1;
    iy = 1;
    % Limit y to boundary
    y = ygrid(1,1);
else
     % (i.e., if current y is greater than or equal to last y-value)
    iy1 = ylen-1;
    iy = ylen;
    % Limit y to boundary
    y = ygrid(ylen,1);
    end
```

```
end
%-------------------------------------------------------------------------

% Define indices of grid points surrounding desired point
ix2 = ix1 + 1;
iy2 = iy1 + 1;

% Define x- and y-values of grid points surrounding desired point
x1 = xgrid(1,ix1); x2 = xgrid(1,ix2);
y1 = ygrid(iy1,1); y2 = ygrid(iy2,1);

% Define z-values of grid points surrounding desired point
z11 = zgrid(iy1,ix1);  z21 = zgrid(iy1,ix2);
z12 = zgrid(iy2,ix1);  z22 = zgrid(iy2,ix2);

% Approximate z-value at desired point using bilinear interpolation
z = 1/(xstep*ystep) ...
    *(z11*(x2-x).*(y2-y)+z21*(x-x1).*(y2-y) ...
     +z12*(x2-x).*(y-y1)+z22*(x-x1).*(y-y1));

% Compute gradient approximation using bilinear interpolation if
desired
if flag
    grad = 1/(xstep*ystep)* ...
     [(z21-z11)*(y2-y)+(z22-z12)*(y-y1); ...
      (z12-z11)*(x2-x)+(z22-z21)*(x-x1)];
end
```

B.13. Two-Dimensional Numerical Optimization (`TwoDTest.m`)

This code generates the plot in Fig. 3.14 of Section 3.3:

```
% Space Vehicle Range Optimization Script File
% Josiah Bryan -- MAE 8990 Masters Thesis Research
% Advised by Dr. Craig Kluever
% October 28, 2010

clc
clear all
close all
format long

global e_pts
global d_alpha_pts
global odestepsize
global V0 gamma0 h0 e0
global Vconstr gammaconstr hconstr econstr
global g m S
global controlmode startingpoint
global aeroflag atmosflag
global isgammafree
global K_gamma
```

```
%  conversion
d2r = pi/180;
r2d = 1/d2r;

% Switch among open-loop (0), optimization (1), and both (2)
runopt = 1;

% Switch aerodynamic model (1 = drag polar, 2 = piecewise poly, 3 =
% table lookup)
aeroflag = 2;  % DO NOT USE AEROFLAG = 2 WITH ATMOSFLAG = 1
    %(No Mach # generated with atmosflag = 1)

% Switch atmospheric model (1 = exponential, 2 = default atmos.m, 3 =
% atmos_1976.m)
atmosflag = 2;

% Switch whether gamma is (TRUE) or is not (FALSE) a free variable to
be
%   optimized
isgammafree = false;

% Switch drag valley calculations on (TRUE) or off (FALSE)
dragvalleyflag = false;

% Set gain for flight-path-angle feedback control for open-loop sim
K_gamma = 1000*d2r;

% Set step size for integration of system
odestepsize = 1000; % ft

% Set acceleration due to gravity (assume constant)
g = 32.174;    % ft/s^2

% Aircraft variables
m = 18000/g;   % slugs   (18000 lb, X-34)
S = 357.5; % ft^2

% Define number of points in angle-of-attack profile for maximum range
NumberOfPts = 2; % number of energy points at which to find
                 %   angle of attack

% Cycle through various starting points
for startingpoint = 1:1 % SET TO 3 TO USE ALL THREE STARTING POINTS
    switch startingpoint
        case 1
            V0 = 1500; % ft/s
            gamma0 = -7.56*d2r; % rad
            h0 = 70000; % ft
        case 2
            V0 = V0 + 100;% ft/s
            gamma0 = -7.56*d2r; % rad
            h0 = e0 - V0^2 / (2*g); % ft
```

```
        case 3
            V0 = V0 - 200; % ft/s
            gamma0 = -7.56*d2r; % rad
            h0 = e0 - V0^2 / (2*g); % ft
    end

        % Initial energy
        e0 = V0^2 / (2*g) + h0; % ft

        % Compute final energy height
        Vconstr = 539; % ft/s (target velocity)
        hconstr = 10000; % ft (target altitude)
        econstr = Vconstr^2 / (2*g) + hconstr; % ft

    % Run open-loop simulation only if runopt == 0 or 2
    if (runopt == 0 || runopt == 2)
        RangeSim
    end

    % If runopt switch ~= 0, then continue with optimization
    if runopt ~= 0
        controlmode = 'Max L/D'; % default = Max L/D is ref trajectory

        if NumberOfPts > 0
            e_pts = linspace(econstr, e0, NumberOfPts); % engy hts (ft)
        end

        % Initialize x-array as zero-degree deviations from
        %   alpha_star profile
        alpha1range = -10:1:10;
        alpha2range = -10:1:15;
        for a1 = 1:length(alpha1range)
            for a2 = 1:length(alpha2range)
                x0 = [alpha1range(a1) alpha2range(a2)]
                [V,gamma,R,t,e] = Traj(x0);
                range_plot(a2,a1) = R(length(R));
            end
        end

        % Define x- and y- grids
        [alpha1,alpha2] = meshgrid(alpha1range,alpha2range);

        % Plot contours of range values
        surf(alpha1,alpha2,range_plot)
        xlabel('d\alpha_1 (deg)')
        ylabel('d\alpha_2 (deg)')
        zlabel('Final Range (ft)')
        title(...
'Range with Two \alpha Deviation Nodes (Nodes defined w.r.t. Mach #)')
        colorbar

        figure
        contour(alpha1,alpha2,range_plot,25)
        xlabel('d\alpha_1 (deg)')
```

```
        ylabel('d\alpha_2 (deg)')
        zlabel('Final Range (ft)')
        title(...
  'Range vs. Two \alpha Deviation Nodes (Nodes defined w.r.t. Mach #)')
        colorbar
        legend('Final Range (ft)')
    end
end
```

B.14. Time-Trial/Validation of Aero Model (`plotAeroDataPiece.m`)

This code generates the plot in Fig. 2.1 of Section 2.3 and other validation plots. Time trials appear in command window.

```
% Plot aerodynamic data and fit

clc
clear all
close all

% Declare global variables
global Alphatable
global Machtable
global CLtable
global CDtable

% Load aerodynamic data
Alphadata = [-6;-3;0;3;6;9;12;15;18;21;-6;-3;0;3;6;9;12;15;18;21;-6;-
3;0;3;6;9;12;15;18;21;-6;-3;0;3;6;9;12;15;18;21;-6;-
3;0;3;6;9;12;15;18;21;-6;-3;0;3;6;9;12;15;18;21;-6;-
3;0;3;6;9;12;15;18;21;-6;-3;0;3;6;9;12;15;18;21;-6;-
3;0;3;6;9;12;15;18;21;-6;-3;0;3;6;9;12;15;18;21;-6;-
3;0;3;6;9;12;15;18;21;-6;-3;0;3;6;9;12;15;18;21;-6;-
3;0;3;6;9;12;15;18;21;-6;-3;0;3;6;9;12;15;18;21;];
Machdata =
[0.300000000000000;0.300000000000000;0.300000000000000;0.30000000000000
0;0.300000000000000;0.300000000000000;0.300000000000000;0.3000000000000
00;0.300000000000000;0.300000000000000;0.400000000000000;0.400000000000
000;0.400000000000000;0.400000000000000;0.400000000000000;0.40000000000
0000;0.400000000000000;0.400000000000000;0.400000000000000;0.4000000000
00000;0.600000000000000;0.600000000000000;0.600000000000000;0.600000000
000000;0.600000000000000;0.600000000000000;0.600000000000000;0.60000000
0000000;0.600000000000000;0.600000000000000;0.800000000000000;0.8000000
00000000;0.800000000000000;0.800000000000000;0.800000000000000;0.800000
000000000;0.800000000000000;0.800000000000000;0.800000000000000;0.80000
0000000000;0.900000000000000;0.900000000000000;0.900000000000000;0.9000
00000000000;0.900000000000000;0.900000000000000;0.900000000000000;0.900
000000000000;0.900000000000000;0.900000000000000;0.950000000000000;0.95
0000000000000;0.950000000000000;0.950000000000000;0.950000000000000;0.9
50000000000000;0.950000000000000;0.950000000000000;0.950000000000000;0.
950000000000000;1.05000000000000;1.05000000000000;1.05000000000000;1.05
000000000000;1.05000000000000;1.05000000000000;1.05000000000000;1.05000
000000000;1.05000000000000;1.05000000000000;1.10000000000000;1.10000000
```

```
000000;1.10000000000000;1.10000000000000;1.10000000000000;1.10000000000
000;1.10000000000000;1.10000000000000;1.10000000000000;1.10000000000000
;1.25000000000000;1.25000000000000;1.25000000000000;1.25000000000000;1.
25000000000000;1.25000000000000;1.25000000000000;1.25000000000000;1.250
00000000000;1.25000000000000;1.40000000000000;1.40000000000000;1.400000
00000000;1.40000000000000;1.40000000000000;1.40000000000000;1.400000000
00000;1.40000000000000;1.40000000000000;1.40000000000000;1.600000000000
00;1.60000000000000;1.60000000000000;1.60000000000000;1.60000000000000;
1.60000000000000;1.60000000000000;1.60000000000000;1.60000000000000;1.6
0000000000000;1.80000000000000;1.80000000000000;1.80000000000000;1.8000
0000000000;1.80000000000000;1.80000000000000;1.80000000000000;1.8000000
0000000;1.80000000000000;1.80000000000000;2;2;2;2;2;2;2;2;2;2;2.5000000
0000000;2.50000000000000;2.50000000000000;2.50000000000000;2.5000000000
0000;2.50000000000000;2.50000000000000;2.50000000000000;2.5000000000000
0;2.50000000000000;];
CLdata = [-0.140064910000000;-
0.0106564750000000;0.122245050000000;0.259211510000000;0.40081476000000
0;0.547626630000000;0.700218980000000;0.859163660000000;1.02503250000000
0;1.19839740000000;-0.174558830000000;-
0.0261138280000000;0.119236990000000;0.263624790000000;0.40918075000000
0;0.558036040000000;0.712321830000000;0.874169290000000;1.04570960000000
0;1.22907390000000;-0.189355390000000;-
0.0320690340000000;0.125119140000000;0.282002150000000;0.43837299000000
0;0.594024680000000;0.748750240000000;0.902342660000000;1.05459500000000
0;1.20530010000000;-0.197890850000000;-
0.0246761250000000;0.145111530000000;0.309960150000000;0.46835775000000
0;0.618792380000000;0.759752050000000;0.889724800000000;1.00719870000000
0;1.11066170000000;-0.189146120000000;-
0.0246627030000000;0.138589220000000;0.298840300000000;0.45432120000000
0;0.603262590000000;0.743895130000000;0.874449480000000;0.99315629000000
00;1.09824620000000;-0.229379820000000;-
0.0637497370000000;0.109200410000000;0.285075080000000;0.45947870000000
0;0.628015720000000;0.786290590000000;0.929907750000000;1.05447170000000
0;1.15558680000000;-0.239147160000000;-
0.0460176410000000;0.142311790000000;0.324360290000000;0.49864703000000
0;0.663691160000000;0.818011850000000;0.960128270000000;1.08855960000000
0;1.20182490000000;-0.238650490000000;-
0.0492636640000000;0.135573080000000;0.314708380000000;0.48699090000000
0;0.651269280000000;0.806392170000000;0.951208220000000;1.08456610000000
0;1.20531440000000;-0.242271620000000;-
0.0744247710000000;0.0934759340000000;0.260039360000000;0.42387437000000
00;0.583589850000000;0.737794650000000;0.885097660000000;1.02410770000000
00;1.15343370000000;-0.236261530000000;-
0.0917519130000000;0.0565774570000000;0.207066380000000;0.35805469000000
00;0.507882160000000;0.654888620000000;0.797413880000000;0.93379776000000
000;1.06238000000000;-0.238965280000000;-
0.110912020000000;0.0221829720000000;0.158726230000000;0.29712431000000
0;0.435783750000000;0.573111090000000;0.707512880000000;0.83739566000000
00;0.961165990000000;-0.231082350000000;-
0.113263480000000;0.00888267400000000;0.133963120000000;0.26058483000000
00;0.387354820000000;0.512880070000000;0.635767570000000;0.75462431000000
000;0.868057280000000;-0.222707610000000;-0.113387970000000;-
0.00053234300000000;0.114800570000000;0.231552050000000;0.34866337000000
000;0.465075840000000;0.579730730000000;0.691569320000000;0.79953291000
0000;-0.198815860000000;-0.107910420000000;-
```

```
0.0140293550000000;0.0824721630000000;0.181238960000000;0.2819158500000
00;0.384147650000000;0.487579200000000;0.591855320000000;0.696620820000
000;];
CDdata =
[0.0577269250000000;0.0325923160000000;0.0321228180000000;0.04240157300
00000;0.0588426480000000;0.0838343070000000;0.124382280000000;0.1897530
10000000;0.289116950000000;0.429191830000000;0.0632036800000000;0.04662
28030000000;0.0370507820000000;0.0375294260000000;0.0511005420000000;0.
0808059410000000;0.129687430000000;0.200786820000000;0.297145910000000;
0.421806520000000;0.0645605670000000;0.0423104810000000;0.0324543850000
000;0.0364784730000000;0.0558689380000000;0.0921119740000000;0.14669377
0000000;0.221100530000000;0.316818440000000;0.435333690000000;0.0658719
220000000;0.0403934010000000;0.0340377320000000;0.0457743600000000;0.07
45727280000000;0.119402280000000;0.179232460000000;0.253032700000000;0.
339772470000000;0.438421190000000;0.0759391380000000;0.0523627890000000
;0.0475063540000000;0.0603704600000000;0.0899557390000000;0.13526282000
0000;0.195292340000000;0.269044910000000;0.355521180000000;0.4537217800
00000;0.0906809220000000;0.0650209510000000;0.0597288210000000;0.073546
3640000000;0.105215410000000;0.153477800000000;0.217075350000000;0.2947
49900000000;0.385243280000000;0.487297330000000;0.122814490000000;0.099
6637400000000;0.0957953350000000;0.110290650000000;0.142231040000000;0.
190697890000000;0.254772560000000;0.333536420000000;0.426070850000000;0
.531457200000000;0.123241400000000;0.0992835630000000;0.094513090000000
0;0.108103780000000;0.139229430000000;0.187063840000000;0.2507808100000
00;0.329554140000000;0.422557630000000;0.528965080000000;0.123418150000
000;0.100475980000000;0.0943705190000000;0.104843250000000;0.1316356700
00000;0.174489270000000;0.233145530000000;0.307345950000000;0.396832020
000000;0.501345220000000;0.121364160000000;0.100306230000000;0.09346153
10000000;0.100967780000000;0.122962690000000;0.159583970000000;0.210969
330000000;0.277256500000000;0.358583180000000;0.455087080000000;0.11675
4980000000;0.0956655410000000;0.0876392360000000;0.0929940260000000;0.1
12047870000000;0.145118740000000;0.192524580000000;0.254583360000000;0.
331613050000000;0.423931590000000;0.112199600000000;0.0917239420000000;
0.0834218300000000;0.0875553810000000;0.104386710000000;0.1341779500000
00;0.177191200000000;0.233688600000000;0.303932250000000;0.388184280000
000;0.106778430000000;0.0871319250000000;0.0787060710000000;0.081818488
0000000;0.0967867960000000;0.123928620000000;0.163561570000000;0.216003
270000000;0.281571340000000;0.360583400000000;0.0971530620000000;0.0794
538840000000;0.0707407980000000;0.0716826680000000;0.0829483580000000;0
.105206730000000;0.139126650000000;0.185376990000000;0.244626600000000;
0.317544360000000;];

% Reorganize data into tables
for i = 1:length(Alphadata)/10
    Alphatable(i,:) = Alphadata(10*(i-1)+1:10*i)';
    Machtable(i,:) = Machdata(10*(i-1)+1:10*i)';
    CLtable(i,:) = CLdata(10*(i-1)+1:10*i)';
    CDtable(i,:) = CDdata(10*(i-1)+1:10*i)';
end

% Plot table data and polynomial values of X-34 lift and drag
coefficients
alpharange = -6:0.5:21;
% ADJUST Machrange TO GET SLICES AT DIFFERENT MACH NUMBERS
Machrange = [0.3,0.4,0.5,0.7,0.95,1.05,1.3,1.5,1.7,2,2.25,0.3];
```

```
% Loop for different Mach numbers
for i = 1:length(Machrange)
    disp(['Mach = ' num2str(Machrange(i))])
    disp('Table lookup:')
    CLinterp = zeros(1,length(alpharange));
    CDinterp = zeros(1,length(alpharange));
    CLpoly = zeros(1,length(alpharange));
    CDpoly = zeros(1,length(alpharange));
    tic
    [CLinterp,CDinterp] = AeroX34Piece(alpharange,Machrange(i),3);
    toc
    disp('Polynomial evaluation:')
    tic
    [CLpoly,CDpoly] = AeroX34Piece(alpharange,Machrange(i),2);
    toc
    disp(' ')

    figure('Name',['C_L vs. Alpha at Mach = ' num2str(Machrange(i))])
    plot(alpharange,CLinterp)
    hold on
    plot(alpharange,CLpoly,'--')
    xlabel('Angle of Attack, alpha (deg)')
    ylabel('Lift Coefficient, C_L')
    title([...
    'Lift Coefficient vs. Angle-of-Attack Profile of X-34 at Mach ' ...
        num2str(Machrange(i))])
    grid
    legend('Interpolated Data', 'Polynomial Fit')

    figure('Name',['C_D vs. Alpha at Mach = ' num2str(Machrange(i))])
    plot(alpharange,CDinterp)
    hold on
    plot(alpharange,CDpoly,'--')
    xlabel('Angle of Attack, alpha (deg)')
    ylabel('Drag Coefficient, C_D')
    title([...
    'Drag Coefficient vs. Angle-of-Attack Profile of X-34 at Mach ' ...
        num2str(Machrange(i))])
    grid
    legend('Interpolated Data', 'Polynomial Fit')

    figure('Name',['L/D vs. Alpha at Mach = ' num2str(Machrange(i))])
    plot(alpharange,CLinterp./CDinterp)
    hold on
    plot(alpharange,CLpoly./CDpoly,'--')
    xlabel('Angle of Attack, alpha (deg)')
    ylabel('Lift-To-Drag Ratio, L/D')
    title([...
    'Lift-To-Drag Ratio vs. Angle-of-Attack Profile of X-34 at Mach
'...
    num2str(Machrange(i))])
    grid
    legend('Interpolated Data', 'Polynomial Fit')
end
```

```
% ------------------------------------------------------------------------
-
% SURFACE PLOT
% Define range of Mach numbers to plot over
Machrange = 0.3:0.1:2.5;

% Open three new figures for plotting
a=figure('Name','Surface Plot of Lift Coefficient');
hold on
b=figure('Name','Surface Plot of Drag Coefficient');
hold on
c=figure('Name','Surface Plot of Lift-To-Drag Ratio');
hold on

% Loop evaluation of aerodynamic data for one Mach number at a time
for i = 1:length(Machrange)
    % Add row to plotting matrices for alpha and Mach
    alphaplot(i,:) = alpharange;
    Machplot(i,:) = Machrange(i)*ones(1,length(alpharange));

    % Evaluate aerodynamic coefficients and add rows to plotting
matrices
    [CLplotinterp(i,:),CDplotinterp(i,:)] = AeroX34Piece(...
        alphaplot(i,:),Machplot(i),3);
    [CLplotpoly(i,:),CDplotpoly(i,:)] = AeroX34Piece(...
        alphaplot(i,:),Machplot(i),2);

    % Plot next rows of interpolated data
    figure(a)
    plot3(alphaplot(i,:),Machplot(i,:),CLplotinterp(i,:),'.')
    figure(b)
    plot3(alphaplot(i,:),Machplot(i,:),CDplotinterp(i,:),'.')
    figure(c)
    plot3(alphaplot(i,:),Machplot(i,:),CLplotinterp(i,:)...
        ./CDplotinterp(i,:),'.')
end

% Plot surface plots of polynomial fits
figure(a)
surf(alphaplot,Machplot,CLplotpoly)
xlabel('Angle of Attack, alpha (deg)')
ylabel('Mach number')
zlabel('Lift Coefficient, C_L')
title('Lift Coefficient vs. Angle-of-Attack and Mach Number of X-34')
colorbar
grid

figure(b)
surf(alphaplot,Machplot,CDplotpoly)
xlabel('Angle of Attack, alpha (deg)')
ylabel('Mach number')
zlabel('Drag Coefficient, C_D')
title('Drag Coefficient vs. Angle-of-Attack and Mach Number of X-34')
```

```
colorbar
grid

figure(c)
surf(alphaplot,Machplot,CLplotpoly./CDplotpoly)
xlabel('Angle of Attack, alpha (deg)')
ylabel('Mach number')
zlabel('Lift-To-Drag Ratio, L/D')
title('Lift-To-Drag Ratio vs. Angle-of-Attack and Mach Number of X-34')
colorbar
grid
```

REFERENCES

1. Federal Aviation Administration's Office of the Associate Administrator for Commercial Space Transportation, 2001, "Reusable Launch Vehicles & Spaceports: Programs & Concepts for 2001," http://www.faa.gov/library/reports/commercial_space/dev_concepts/media/2001RLV.pdf, accessed 2 Nov 2011.

2. National Aeronautics and Space Administration, 2011, "Space Shuttle Launches," KSC Historical Report No. 1B (KHR-1B), http://www.nasa.gov/pdf/537939main_ss-launches-080311.pdf, accessed 2 Nov 2011.

3. National Aeronautics and Space Administration, 2001, "2nd Generation Reusable Launch Vehicle Program: Level 1 Requirements," Document No. MSFC-RGMT-3221, http://www.spaceref.com/docs/NASA/SLI/sli_level_1_requirements.pdf, accessed 2 Nov 2011.

4. Hanson, J. M., 2002, "A Plan for Advanced Guidance and Control Technology for 2nd Generation Reusable Launch Vehicles," AIAA Guidance, Navigation, and Control Conference and Exhibit, AIAA paper 2002-4557.

5. Hanson, J. M., March 2003, "New guidance for new launchers," *Aerospace America*, **41**, No. 3, 36-41.

6. Ehlers, H. L., and Kraemer, J. W., 1977, "Shuttle orbiter guidance system for the terminal flight phase," *Automatica*, **13**, Issue 1, 11-21.

7. Minott, G. M., Peller, J. B., and Cox, K. J., 1976, "Space Shuttle Digital Flight Control System," NASA Technical Report N76-31146, NASA Technical Reports Server (NTRS), http://ntrs.nasa.gov, accessed 2 Nov 2011.

8. Moore, T. E., 1991, "Space Shuttle Entry Terminal Area Energy Management," NASA Technical Memorandum 104744, NASA Technical Reports Server, http://ntrs.nasa.gov, accessed 4 Nov 2011.

9. Carman, G. L., and Montez, M. N., 1980, "MCC Level C Formulation Requirements: Shuttle TAEM Guidance and Flight Control (Optional TAEM Targeting)," 80-FM-29, NASA Technical Reports Server (NTRS), http://ntrs.nasa.gov, accessed 4 Nov 2011.

10. Schierman, J. D., Hull, J. R., and Ward, D. G., August 2002, "Adaptive Guidance with Trajectory Reshaping for Reusable Launch Vehicles," AIAA Paper 02-4458.

11. Barton, G. H., and Tragesser, S. G., August 1999, "Autolanding Trajectory Design for the X-34," AIAA Paper 99-4161.

12. Kluever, C. A., 2004, "Unpowered Approach and Landing Guidance Using Trajectory Planning," *Journal of Guidance, Control, and Dynamics*, **27**, No. 6, 967-974.

13. Kluever, C. A., and Horneman, K. R., August 2005, "Terminal Trajectory Planning and Optimization for an Unpowered Reusable Launch Vehicle," AIAA Paper 2005-6058.

14. Burchett, B. T., 2004, "Fuzzy Logic Trajectory Design and Guidance for Terminal Area Energy Management," *Journal of Spacecraft and Rockets*, **41**, No. 3, 444-450.

15. Da Costa, R. R., August 2003, "Studies for Terminal Area GNC of Reusable Launch Vehicles," AIAA Paper 2003-5438.

16. Grantham, K., January 2003, "Adaptive Critic Neural Network Based Terminal Area Energy Management/Entry Guidance," AIAA Paper 2003-0305.

17. Hull, J. R., Gandhi, N., and Schierman, J. D., September 2005, "In-Flight TAEM/Final Approach Trajectory Generation for Reusable Launch Vehicles," AIAA Paper 2005-7114.

18. Kluever, C. A., 2007, "Terminal Guidance for an Unpowered Reusable Launch Vehicle with Bank Constraints," *Journal of Guidance, Control, and Dynamics*, **30**, No. 1, 162-168.

19. Baek, J., Lee, D., Kim, J., Cho, K., and Yang, J., 2008, "Trajectory optimization and the control of a re-entry vehicle in TAEM phase," *Journal of Mechanical Science and Technology*, **22**, 1099-1110.

20. De Ridder, S., 2011, "Optimal Longitudinal Trajectories for Reusable Space Vehicles in the Terminal Area," *Journal of Spacecraft and Rockets*, **48**, No. 4, 642-653.

21. "X-34 Advanced Technology Demonstrator," 9 Dec 2009, Dryden Flight Research Center, http://www.nasa.gov/centers/dryden/news/FactSheets/FS-060-DFRC.html, accessed 28 March 2011.

22. "On the Road Again: X-34s Moved to Mojave for Inspection," 17 Nov 2010, Dryden Flight Research Center, http://www.nasa.gov/centers/dryden/status_reports/X-34s_moved_11_17_10.html, accessed 28 March 2011.

23. Pamadi, B. N., 2009, "X-34 Free Flight Database R5.3," Vehicle Analysis Branch, SACD, NASA Langley Research Center, Hampton, VA.

24. Chapra, S. C., 2008, *Applied Numerical Methods with MATLAB for Engineers and Scientists*, McGraw-Hill, Boston.

25. Kluever, C. A., Fall 2010, "MAE 7720: Modern Control," course materials.

26. Kluever, C. A., Spring 2009, "MAE 4620: Aircraft Flight Mechanics," course materials.

VITA

Josiah Bryan lived in St. Charles, MO until he was six, then moved to Columbia, MO, where he graduated from Rock Bridge High School in May 2006. He graduated from the University of Missouri in May 2010 with a BS in mechanical engineering with an emphasis in aerospace and minors in mathematics and business. He hopes to complete an MS in mechanical and aerospace engineering at the University of Missouri in December 2011. Throughout his undergraduate and graduate careers he has focused on studies in dynamics, controls, and optimization.

In addition to studying engineering, Josiah is an avid pianist, having played in the Columbia Jazz Orchestra and the University Concert Jazz Band, and helping to lead worship in the Mizzou campus ministry he attends. He enjoys composing music, running and, above all, getting to know his Lord and Savior Jesus Christ, Who alone is worthy of all glory and praise for every good thing in His Creation, including this thesis.