APPLICATION OF THEORY OF FUNCTIONAL CONNECTIONS FOR OPTIMAL CONTROL OF NONLINEAR SYSTEMS

A Thesis

by

GREGORY J. ARLETH

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Co-Chairs of Committee,	John Hurtado
	Daniele Mortari
Committee Members,	Yalchin Efendiev
Head of Department,	Srinivas Vadali

May 2021

Major Subject: Aerospace Engineering

ABSTRACT

Solving nonlinear optimal control problems can be a difficult and time intensive process. This is especially true when the state dynamics of the problem are computationally expensive to solve. Using the Theory of Functional Connections, linear combinations of orthogonal manifolds are used to approximate computationally expensive terms within the state dynamics. Doing so can yield significantly faster solution time with minimal error compared to directly calculating these terms.

To test this method, a longitudinal dynamic model of a hypersonic vehicle was created. The aerodynamic lift and drag forces, as well as the pitching moment, were calculated using a panel method along with Pradtl-Meyer expansion and compression wave equations. The calculated forces and moment were then verified by comparing to CFD solutions generated by SOLIDWORKS Flow Analysis Tool at various angles of attack and Mach numbers.

This panel method still took up a bulk of the computation time needed to solve optimal control problems as it has to run for every timestep in each trajectory iteration created in the solution process. A new method to approximate the aerodynamic forces and moment was created by using this panel method alongside a least squares algorithm to solve for weights of recursively generated orthogonal manifolds. Lift, drag, and pitching moment could then be approximated by evaluating each orthogonal manifold at locations corresponding to states and controls. Compared to the original panel method, which took 117.30 seconds to calculate 10,000 different flight conditions, this least squares method calculated the same conditions in 0.13 seconds, almost 1000 times faster.

The new least squares method was then used alongside a nonlinear optimal control problem algorithm known as Dynamic Programming with Interior Points. Using the hypersonic model with the aerodynamic forces and moment approximated using orthogonal manifolds, solutions to several optimal control problems were found in a less than a minute for problems which would take over 2 hours otherwise.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of advisor Professor Hurtado and co-chair Professor Mortari of the Department of Aerospace Engineering as well as Professor Efendiev of the Department of Mathematics.

The base code for the Dynamic Programming algorithm was provided by Professor Hurtado. All other work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was supported by Sandia National Laboratories.

NOMENCLATURE

0.1 Abbreviations

COM	Center of Mass
DPIP	Dynamic Programming with Interior Points
LQR	Linear Quadratic Regulator
OCP	Optimal Control Problem
TFC	Theory of Functional Connections

0.2 Variables

$a_{()}$	Speed of Sound at Point
A	Reshaped Manifold Array
$\mathcal{A}$	Manifold Array
$\boldsymbol{b}$	Reshaped Evaluated Mesh
$\mathcal{B}$	Manifold
$\mathbb{B}$	Collocated Mesh
$c_{()}$	Scaling Factor for Mapping Collocation Points
D	Drag
F_x	Total Force Acting in x Direction
$F_{x()}$	Force acting in x Direction due to Surface
F_z	Total Force Acting in x Direction
$F_{z()}$	Force acting in x Direction due to Surface
g	Gravitational Acceration
$\mathbb{G}$	Final State Penalty Function
I_{yy}	Moment of Inertia about y-axis
J	Cost Function
K	Feedback gain on Error
L	Lift
$L_{()}$	Horizontal Length of Surface
$\mathbb{L}$	State and Control Weighting Function
$\mathcal{L}$	Legendre Polynomial
m	Mass
M	Pitching Moment
$M_{()}$	Pitching Moment Induced by Panel
$M_{()}$	Mach Number at Point
$M_{()n}$	Normal Mach Number
n	Number of Collocation Points
$N_{()}$	Number of Instance
$P_{()}$	Pressure at Point
$P_{t()}$	Total Pressure at Point
Q	Pitch Rate
$T_{()}$	Temperature at Point
S_e	Elevon Area per Unit Span
v	Velocity
x	Horizontal Position
x_i	Collocated Input
$\bar{x}_{()}$	x Distance of Surface from COM
z	Altitude (ASL)
z_i	Collocation Point
$\bar{z}_{()}$	z Distance of Surface from COM
α	Angle of Attack
$\beta_{()}$	Oblique Shock Angle at Point
δ	Pressure Ratio
δ_e	Elevon Deflection Angle
γ	Specific Heat Ratio of Air
θ	Inertial Pitch Angle
$\theta_{()}$	Flow Deflection Angle at Point
$\nu_{()}$	Flow Turning Angle at Point
$\boldsymbol{\xi}$	Weighting Vector
$\tau_{()}$	Surface Angle
ϕ	Temperature Ratio

0.3 Subscripts

$0M$	Zero Moment Model
a	Aft Panel
$body$	Body
$\mathcal{B}$	Basis Manifolds
$corrected$	Corrected for Unstable Aircraft
$full$	Full State Model
$inputs$	Number of Inputs
l	Lower Panel
$\mathcal{L}$	Basis Functions
min	Minimum
max	Maximum
n	Nacelle Panel
u	Upper Surface
x	Before Interact
y	After Interact
δ	Elevon
δu	Upper Elevon
δl	Lower Elevon
ϵ	Error
∞	Freestream

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

1. INTRODUCTION AND LITERATURE REVIEW

1.1 Optimal Control

Performance optimization of dynamic systems is an ongoing field of control systems. The dynamics of a system can usually generalized as [1]:

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \tag{1.1}$$

Where $\boldsymbol{x}$ is a vector of the states of a system, $\boldsymbol{u}$ is a vector of the the controls applied to the system, and t represents time. The performance index of a continuous nonlinear dynamic system can be measured as [1]:

$$J = \mathbb{G}(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} \mathbb{L}(\boldsymbol{x}(t), \boldsymbol{u}(t), t)\ \mathrm{d}t \tag{1.2}$$

The goal of optimal control is to minimize the performance index for the system. Other parameters may be specified such as initial and final state conditions, minimum or maximum constraints on the control, a final time specified etc. But when calculating the derivatives of the states is computationally expensive, as is the case for some nonlinear problems, minimizing the performance index can become prohibitively long.

1.2 OCP for Hypersonic Vehicles

Solving optimal control problems for hypersonic vehicles is an ongoing investigation utilizing many different methods. The dynamics of the aircraft are highly nonlinear [2]. Unlike OCP involving linear dynamics with a quadratic cost function, which can be solved directly by calculating the optimal gains for a given solution of the algebraic Ricatti equation, nonlinear problems must be solved iteratively [1]. Previous methods used to solve OCP using hypersonic dynamics have involved ideas such as linearizing the dynamics, H-infinity controllers, or adaptive dynamic inversion [2] [3] [4]. This research will focus on developing new methods to simplify established

nonlinear dynamical model for use with robust nonlinear OCP solvers. In addition, a simplified dynamics model will also be considered for longer flight time problems.

1.3 Theory of Functional Connections

The Theory of Functional Connections (TFC) was conceived by Daniele Mortari at Texas A&M University. One of the concepts of TFC is to represent a function as a linear combination of orthogonal polynomials [5]. While the orthogonal polynomials themselves are given and can be generated recursively, the weighting for each is solved using a least squares method to best approximate the function. In a simple case, a single combination of orthogonal polynomials can approximate a function with one input and one output as in the figure below:

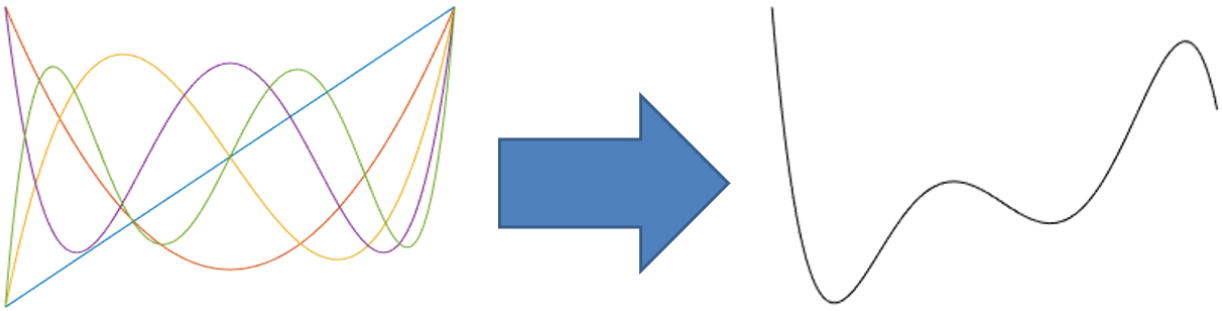

Figure 1.1: Linear combination of orthogonal polynomials

In the case for our hypersonic problem where aerodynamic forces can be a function of altitude, airspeed, angle of attack, etc. functions with multiple inputs must be considered. This can be achieved by creating orthogonal surfaces or manifolds by first generating orthogonal polynomials, then finding a linear combination of product combinations of these polynomials. In these products each polynomial is evaluated for a unique input variable. For example, the first polynomial in a product would be evaluated based on airspeed, and the second would be evaluated based on angle of attack. Many surfaces can be created this way and once the weighting for each is solved, the function can be approximated. An example for a 2 input case is shown below, with $\mathcal{B}$ as weighted

surfaces, $\mathcal{A}$ as the approximated function, and $\boldsymbol{\xi}$ as the solved weightings for each surface*:

$$\begin{aligned} \mathcal{B}_{i,j} &= \mathcal{L}_i(x)\mathcal{L}_j(y) \\ f(x,y) &\approx \sum \boldsymbol{\xi}_{i,j}\mathcal{B}_{i,j} \end{aligned} \tag{1.3}$$

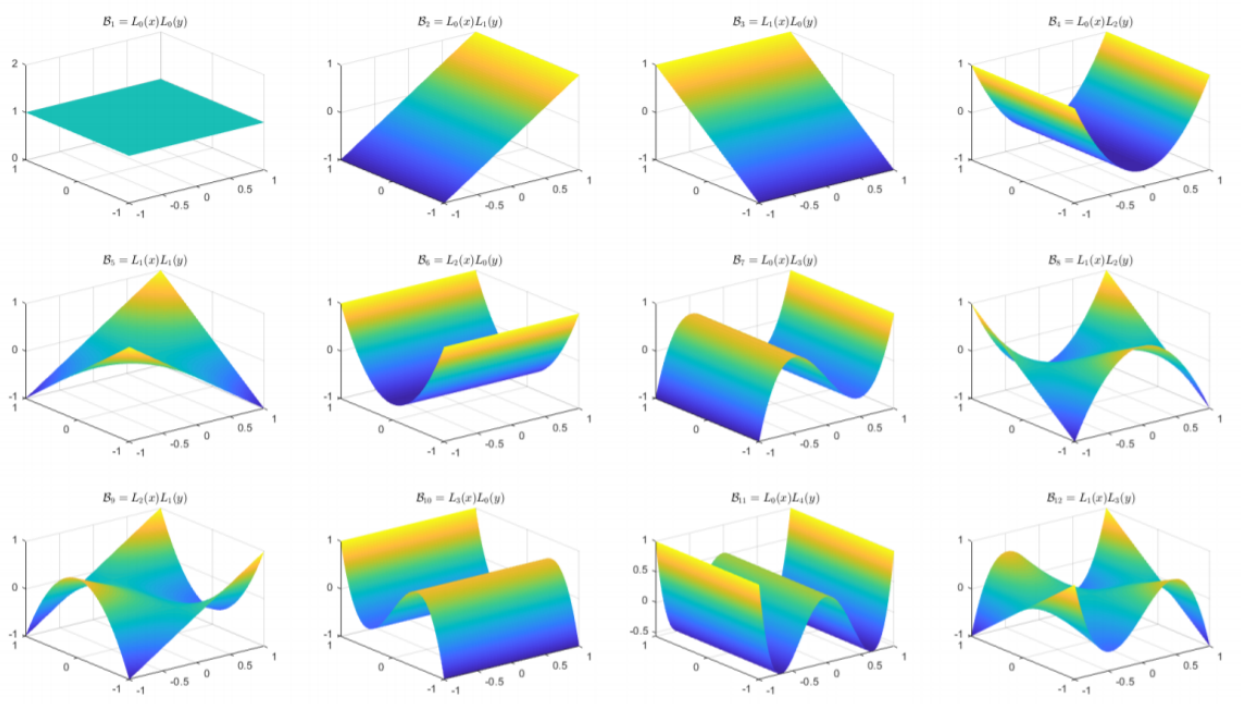

Figure 1.2: Orthogonal surfaces

The process to solve for the weightings for multiple input case is as follows:

1. Determine the lower and upper bounds of the N_{inputs} inputs to the function
2. For each input, create n collocation points and evaluate the function for every combination of these collocation points
3. Collect the results into a $n \times \cdots \times n \times N_{inputs}$ array $\mathbb{B}$
4. For each input, recursively create $N_{\mathcal{L}}$ orthogonal polynomials as basis functions and evaluate each polynomial at the n number of collocation points on the domain $[-1, +1]$

*Figure reprinted with permission from *The theory of functional connections*, by D. Mortari, H. Johnston, and C. Leake, Forthcoming book, 2021.

5. Multiply every combination of the collocation points evaluated for the basis functions to get $N_{\mathcal{L}}^{N_{inputs}}$ basis manifolds

6. Collect the results into a $n \times \cdots \times n \times N_{\mathcal{L}}^{N_{inputs}}$ array $\mathcal{A}$

7. Rearrange arrays $\mathcal{A}$ and $\mathbb{B}$ into matrices A and $\boldsymbol{b}$ of dimension $n^{N_{inputs}} \times N_{\mathcal{L}}$ and $nN \times 1$ respectively

8. Given the equation $A\boldsymbol{\xi} = \boldsymbol{b}$ calculate the pseudo-inverse for A and solve for $\boldsymbol{\xi} = A^{+}\boldsymbol{b}$

$\boldsymbol{\xi}$ is a vector containing the weighting of each orthogonal polynomial. To process to estimate the function for a given input using the calculated $\boldsymbol{\xi}$ vector is as follows:

1. Linearly map each input to the $[-1, +1]$ domain. The lower bound of each input should correspond to -1 and the upper bound to $+1$

2. Using the same set of orthogonal polynomials used to generate the $\boldsymbol{\xi}$ vector, evaluate each polynomial at the mapped inputs

3. Multiply every combination of the evaluated polynomials in the same order to get a set of $\mathcal{B}$ manifolds each evaluated at only one point

4. Weight each of the resulting elements by its respective $\boldsymbol{\xi}$ element

5. The sum of the weighted results is the estimated value of function evaluated at the given inputs

A good choice of basis functions would be orthogonal polynomials such as Legendre Polynomials, which are described in the domain $z \in [-1, +1]$ and can be generated recursively [5].

$$\mathcal{L}_{k+1} = \frac{2k+1}{k+1} z \mathcal{L}_k - \frac{k}{k+1} \mathcal{L}_{k-1} \qquad \text{where: } \begin{cases} \mathcal{L}_0 = 1 \\ \mathcal{L}_1 = z \end{cases} \tag{1.4}$$

2. HYPERSONIC MODEL

One of the goals of this research is to construct and verify a hypersonic model to simulate the dynamics of the vehicle on. Two models have been created, one using full state dynamics developed by previous research and another reduced order model based on simplifying assumptions made in this study [6]. The geometry and mass properties of a sample vehicle are given and slightly modified from prior examples [2]. Because the model considered is two dimensional, some parameters such as mass are given as per unit width of the aircraft.

Parameter	Value	Parameter	Value
I_{yy}	5E5 lb ft^2/ft	$\bar{x}$	-50 ft
L_a	10 ft	$\bar{z}$	0 ft
L_f	47 ft	τ_a	46.5deg
L_n	43 ft	τ_l	6.5deg
m	300 lbf/ft	τ_u	3deg
S_e	22 ft^2/ft		

Table 2.1: Aircraft Geometry and Mass Parameters

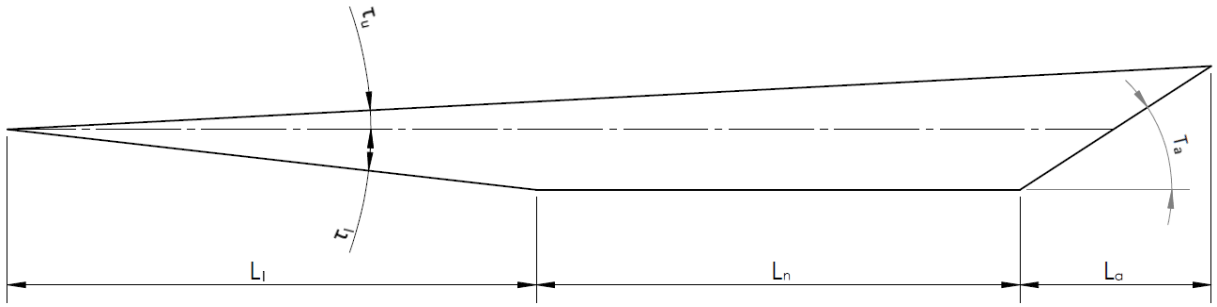

Figure 2.1: Hypersonic vehicle geometry

2.1 Full State Model

The equations of motion for an in-plane hypersonic model are with only elevon deflection are [6]:

$$\begin{aligned}
\dot{x} &= v\cos(\theta - \alpha) \\
\dot{z} &= v\sin(\theta - \alpha) \\
\dot{\theta} &= Q \\
\dot{\alpha} &= \frac{g\cos(\theta - \alpha) - L/m}{v} + Q \\
\dot{v} &= -\frac{D}{m} - g\sin(\theta - \alpha) \\
\dot{Q} &= \frac{M}{I_{yy}}
\end{aligned} \tag{2.1}$$

While these equations of motion seem simple at a glance, the aerodynamic forces L, D, M are computationally expensive to calculate. In this case, they were calculated using a panel method and Prandtl-Meyer wave equations as described in the following subsection.

2.1.1 Panel Method

The direct method used to calculate lift, drag, and pitching moment are derived using Prandtl-Meyer expansion and compression wave equations [2]. First, the vehicle was broken up into 6 distinct panels, the upper surface, the leading lower surface, the nacelle, the aft lower surface, and the upper and lower elevon surface. Pressure was assumed to be constant across these panels. When pressure on each panel was calculated, a normal force was applied at the center of each panel creates the lift and drag forces, and the resultant force acting about the center of mass created the pitching moment.

From the aircraft's altitude given by the z state, the atmospheric properties of the freestream air was be interpolated from a standard atmosphere table [7]. In the table used, the ratio of ambient pressure at the current altitude and sea level is given as δ and the equivalent temperature ratio is given as ϕ. These ratios can then be used to calculate the ambient pressure, temperature, and speed

of sound.

$$\begin{aligned} P_\infty &= \delta\, P_{std} \\ T_\infty &= \phi\, T_{std} \\ a_\infty &= \sqrt{\theta}\, a_{std} \end{aligned} \tag{2.2}$$

Once the speed of sound of the freestream is calculated, then the Mach speed of the aircraft was then calculated.

$$M_\infty = v/a_\infty \tag{2.3}$$

With the velocity of the aircraft now measured in Mach number, the stagnation or total pressure and temperature of the freestream could then be calculated. These values correspond to the pressure and temperature of the air if it is adiabatically slowed down with respect to the aircraft assuming it is calorically perfect [7].

$$P_{ti} = P_i \left(1 + \frac{\gamma - 1}{2} M_i\right)^{\frac{\gamma}{\gamma - 1}} \tag{2.4}$$

At the leading edge of the aircraft, two wave interactions occur. Depending on the angle of attack and geometry of the aircraft, these wave interactions can either be compression or expansion waves. An oblique compression wave occurs when a supersonic flow is suddenly deflected by angle θ_i due to the presence of a wall pushing into the flow. An expansion wave occurs when a supersonic flow is gradually deflected by angle θ_i due to the presence of a wall pulling away from the flow [7]. This angle θ is a function of angle of attack and vehicle geometry.

$$\begin{aligned} \theta_u &= \tau_u - \alpha \\ \theta_l &= \tau_l + \alpha \end{aligned} \tag{2.5}$$

When θ is positive, an oblique shock will occur on the corresponding surface. When θ is negative, an expansion wave will occur. When θ is zero, the freestream flows parallel to the surface geometry and there is no supersonic interaction whatsoever. Notice that the sign of α is flipped between the upper and lower θ calculations. This is because, as the aircraft pitches further up and

α becomes higher, the lower surface is rotated into the flow which encourages the formation of stronger oblique shocks. Likewise, as the upper surface of the aircraft rotates down and away from the flow, which encourages the formation of stronger expansion waves.

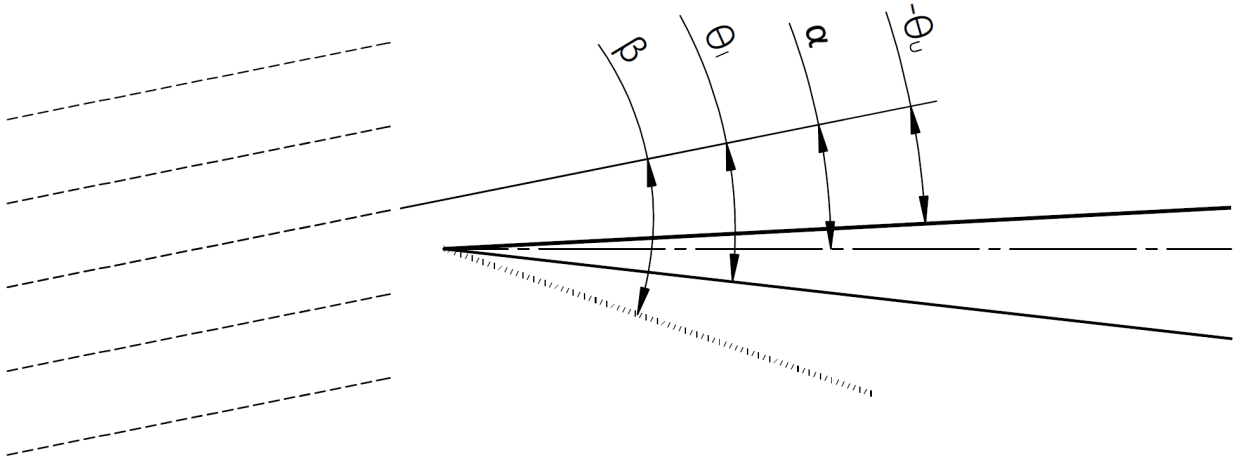

Figure 2.2: Shock Angles with respect to the leading edge of the aircraft

The oblique shock angle β which can occur in either of the forward surfaces was found using the following equation [7]. This is completed with MATLAB's iterative nonlinear equation solver fsolve(), which iteratively tests solutions until a convergence criteria is satisfied [8].

$$\tan(\beta_i - \theta_i) = \frac{2 + (\gamma - 1)M_i^2 \sin^2 \beta_i}{(\gamma + 1)M_i^2 \sin \beta_i \cos \beta_i} \tag{2.6}$$

While two unique solutions exist for the oblique shock angle, the smaller shock angle typically occurs to form a weak shock angle [7]. To account for this, an initial angle of 10^-6 was used to iteratively solve for the shock angle as it is closer to the shock angle associated with a weak oblique shock wave. Once the shock angle is found the flow properties behind it can be calculated by treating it as a normal shock wave acting on the component of the flow normal to the wave [7].

$$M_{xn} = M_x \sin \beta_i \tag{2.7}$$

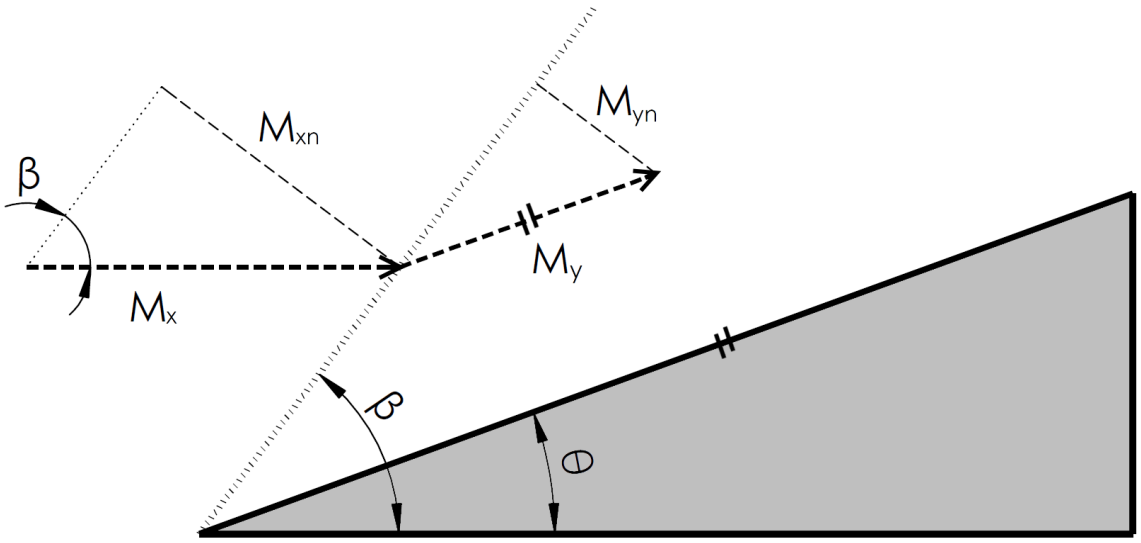

Figure 2.3: Oblique shock angles and Mach flow normal to the shock

$$M_{y_n} = \sqrt{\frac{M_{x_n}^2 + \frac{2}{\gamma-1}}{\frac{2\gamma}{\gamma-1}M_{x_n}^2 - 1}} \tag{2.8}$$

$$M_y = \frac{M_{y_n}}{\sin(\beta_i - \theta_i)} \tag{2.9}$$

$$P_y = P_x \left(\frac{2\gamma}{\gamma+1} M_{x_n}^2 - \frac{\gamma-1}{\gamma+1} \right) \tag{2.10}$$

$$P_{ty} = P_y \left(1 + \frac{\gamma-1}{2} M_y^2 \right)^{\frac{-\gamma}{\gamma-1}} \tag{2.11}$$

The post shock flow is assumed to be parallel to the subsequent surface of the aircraft after the flow has been deflected. The resulting Mach number and total pressure behind the shock wave are also computed to be used for subsequent supersonic interactions in the resulting flow. In the case of the vehicle geometry used for this experiment, since there is only one panel used for the upper surface of the vehicle, if an oblique shock occurs ahead of the upper surface only the equation used to calculate pressure behind the shock is needed.

If an expansion wave is expected to occur at the leading edge of the vehicle, during the transition from lower forward surface to nacelle, or from nacelle to lower aft surface, Pradtl-Meyer expansion wave equations are used. First we evaluate the flow turning angle for the pre-fan flow using the Pradtl-Meyer function [7].

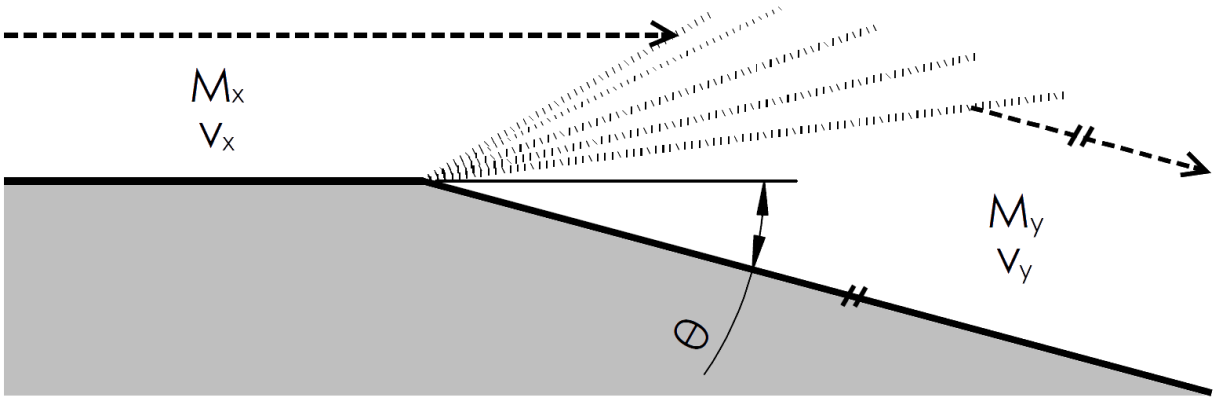

Figure 2.4: Expansion fan due deflection away from the flow

$$\nu_x = \left(\sqrt{\frac{\gamma+1}{\gamma-1}}\tan^{-1}\sqrt{\frac{\gamma-1}{\gamma+1}(M_x^2-1)} - \tan^{-1}\sqrt{M_x^2-1}\right) \tag{2.12}$$

$$\nu_y = \nu_x - \theta_i \tag{2.13}$$

Once the flow angle of the post-fan flow has been calculated, the resulting Mach number of the flow was iteratively solved by MATLAB's fsolve().

$$\nu_y = \left(\sqrt{\frac{\gamma+1}{\gamma-1}}\tan^{-1}\sqrt{\frac{\gamma-1}{\gamma+1}(M_y^2-1)} - \tan^{-1}\sqrt{M_y^2-1}\right) \tag{2.14}$$

Calculation for pressure behind the fan is simple as the expansion flow is adiabatic and isentropic, meaning that the total temperature and total pressure do not change [7]. Changes in pressure then only depend on the fact that the Mach number of the flow is higher behind the fan and can be calculated using equation 2.4.

$$P_{tx} = P_{ty} \tag{2.15}$$

On the lower surfaces, there is guaranteed to be expansion waves at the turning point from the lower forward surface and the nacelle and the nacelle to the aft surface. This is because after the supersonic interaction at the leading edge of the aircraft, flow will be deflected to run parallel to each panel. As such we can calculate the flow turning angles based on the geometry of the aircraft. Note that at the leading edge θ_i was negative in cases where expansion fans occurred and the term was negated to account for this. In the following two turns, the respective angles τ_{1l} and τ_2 are positive and as such are added to ν instead of subtracted.

$$\nu_y = \nu_x + \tau_i \tag{2.16}$$

The pressure acting on every panel composing the body of the aircraft was then calculated. All that remained was the pressure acting on the elevon. The elevon was treated as a flat plate in the freestream, the flow angle was then calculated as the negative sum of the aircraft's angle of attack and the elevon deflection.

$$\theta_e = -\delta_e - \alpha \tag{2.17}$$

When θ_e is positive, an oblique shock occurs in front of the upper surface of the elevon while an expansion wave occurs in front of the lower surface. The opposite is true when θ_e is negative, and no interaction occurs when θ_e is zero. The pressure across the entire aircraft could then be integrated and used to find the aerodynamic forces and pitching moment acting on the aircraft. The location of the pressure panels and the acting force in the x and z directions are shown in the figure and equations on the following page.

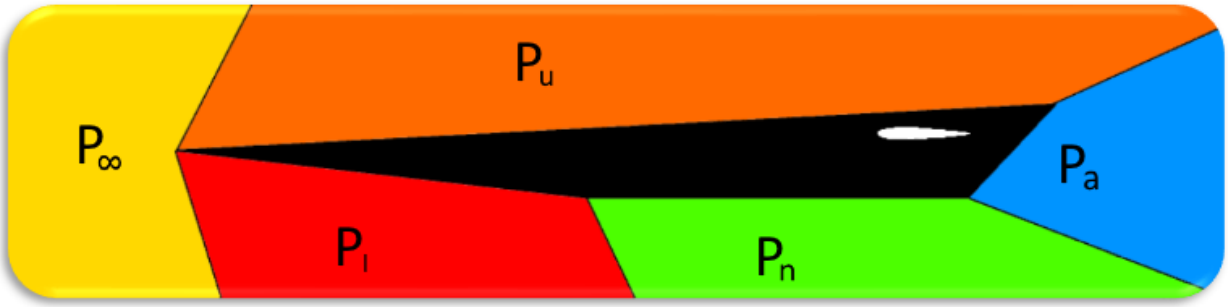

Figure 2.5: Pressure acting on each panel of the aircraft

$$
\begin{aligned}
F_{xu} &= -P_u L_u \tan \tau_u \\
F_{xl} &= -P_l L_l \tan \tau_l \\
F_{xn} &= 0 \\
F_{xa} &= -P_a L_a \tan \tau_a \\
F_{x\delta} &= (P_{\delta u} - P_{\delta l}) S_e \sin \delta_e \\
F_x &= F_{xu} + F_{xl} + F_{xn} + F_{xa} + F_{x\delta}
\end{aligned}
\tag{2.18}
$$

$$
\begin{aligned}
F_{zu} &= -P_u L_u \\
F_{zl} &= -P_l L_l \\
F_{zn} &= -P_n L_n \\
F_{za} &= -P_a L_a \\
F_{z\delta} &= (P_{\delta u} - P_{\delta l}) S_e \cos \delta_e \\
F_z &= F_{zu} + F_{zl} + F_{zn} + F_{za} + F_{z\delta}
\end{aligned}
\tag{2.19}
$$

From the forces acting in the body frame, lift and drag were then calculated as:

$$
\begin{aligned}
L &= F_x \sin \alpha - F_z \cos(\alpha) \\
D &= -F_x \cos \alpha - F_z \cos(\alpha)
\end{aligned}
\tag{2.20}
$$

Pitching moment was also calculated by multiplying the forces acting in the x and z directions by their respective moment arms. Because pressure is assumed to act uniformly across each panel, the moment arm is calculated from the center of mass of the aircraft to the center of each panel.

$$\begin{aligned} M_i &= \bar{z}_i F_{xi} - \bar{x} F_{zi} \\ M &= \sum M_i \end{aligned} \tag{2.21}$$

2.1.2 Temperature Model

Maximum temperature of the hypersonic aircraft is not necessary to calculate aerodynamic forces or moment. However, the heating of air due to compressible effects can present a constraint on the performance of the vehicle. This being the case, it is advantageous to be able to calculate maximum temperature of the airflow around the vehicle and to formulate optimal control problems around conditions where a temperature limit is not exceeded. Fortunately, the hypersonic model already permits the calculation of temperature ratio behind oblique shock waves and the freestream temperature.

Given the geometry of the vehicle model, the location of the maximum nearby airflow temperature could easily be deduced. The nacelle and aft surface of the aircraft must be behind an expansion wave, as the vehicle is convex and flow is parallel to the surfaces ahead of them. Because expansion waves reduce the temperature of the flow, the flow ahead of these expansion waves must be greater than the temperature measured at these panels [7]. This leaves the upper or lower forward surfaces as candidates for highest temperature flow. Due to the geometry of the vehicle, expansion waves cannot simultaneously occur in front of both surfaces. Therefore, if one surface develops an expansion fan in front of it, the other surface will have a higher temperature as it will either develop an oblique shock wave or run parallel to the flow with no supersonic interaction. In either case, the temperature of the opposite surface will be higher as oblique shocks always result in an increase in temperature of the flow [7]. At low angles of attack, it is possible for both surfaces to form an oblique shockwave in front of them. In this case, whichever surface deflects the flow

more will create a stronger oblique shockwave and thus have a higher temperature in front of it.

Ultimately the surface with a greater associated θ_i will have the highest temperature. Recall that this value describes the turning angle of the flow and an expansion wave will occur if θ_i is negative. If the maximum θ_i is zero, than the highest temperature is expected to be that of the freestream. Otherwise, the temperature can be found using the normal component of the flow velocity to the shock wave along with the freestream temperature [7].

$$T_y = T_x \frac{\left(1 + \frac{\gamma-1}{2} {M_{x_n}}^2\right)\left(\frac{2\gamma}{\gamma-1} {M_{x_n}}^2 - 1\right)}{\frac{(\gamma+1)^2}{2(\gamma-1)} {M_{x_n}}^2} \tag{2.22}$$

2.1.3 Verification of Full State Model

The accuracy of the hypersonic model was verified by comparing the aerodynamic pressure and temperature results to CFD results from a model created in SOLIDWORKS. Since the aerodynamic model is longitudinal and there is no good way to model an elevon within the cross section of the aircraft, the forces contributed by the elevon in both models is ignored. Essentially, only the aerodynamic body forces and moment are considered. To test the aerodynamic forces and moment, a variety of flight conditions with unique combinations of Mach numbers and angle of attacks are created. Freestream pressure and temperature are given to be that of sea level for both models. Since the MATLAB model calculates pressure and temperature on each panel directly proportionally to the freestream, varying the ambient pressure and temperature is not necessary. The pressure acting on each panel is recorded, as well as the total lift and drag force and the pitching moment. The pressure acting on each panel in the simulations is given in the table on the following page.

M_∞	α (rad)	MATLAB				CFD			
		P_u	P_l	P_n	P_a	P_u	P_l	P_n	P_a
3	-0.21	5.97E3	1.35E3	7.58E2	3.00E-1	5.97E3	1.38E3	7.89E2	1.64E2
	-0.14	4.64E3	1.88E3	1.09E3	1.10E0	4.65E3	1.91E3	1.13E3	1.43E2
	-0.07	3.55E3	2.56E3	1.54E3	3.00EE0	3.56E3	2.58E3	1.58E3	1.38E2
	0.00	2.66E3	3.42E3	2.12E3	7.00E0	2.68E3	3.44E3	2.17E3	1.67E2
	0.07	1.96E3	4.49E3	2.86E3	1.60E1	1.98E3	4.49E3	2.93E3	1.39E2
	0.14	1.41E3	5.79E3	3.80E3	3.30E1	1.45E3	5.73E3	3.91E3	2.98E2
	0.21	9.99E2	7.34E3	4.96E3	6.70E1	1.03E3	7.30E3	5.05E3	2.59E2
5	-0.21	1.01E4	1.02E3	3.82E2	0.00E0	1.01E4	1.07E3	4.38E2	1.38E2
	-0.14	7.09E3	1.75E3	7.13E2	0.00E0	7.11E3	1.80E3	7.87E2	1.09E2
	-0.07	4.75E3	2.87E3	1.26E3	0.00E3	4.79E3	2.92E3	1.36E3	8.90E1
	0.00	3.04E3	4.50E3	2.12E3	0.00E0	3.09E3	4.52E3	2.27E3	8.80E1
	0.07	1.87E3	6.76E3	3.40E3	7.00E-2	1.92E3	6.73E3	3.60E3	1.17E2
	0.14	1.10E3	9.70E3	5.21E3	8.60E-1	1.15E3	9.62E3	5.47E3	1.73E2
	0.21	6.13E2	1.33E4	7.64E3	5.79E0	6.75E2	1.32E4	7.95E3	2.61E2
7	-0.21	1.60E4	7.57E2	1.75E2	0.00E0	1.61E4	8.34E2	3.64E2	1.85E2
	-0.14	1.04E4	1.62E3	4.48E2	0.00E0	1.05E4	1.71E3	6.75E2	1.42E2
	-0.07	6.29E3	3.21E3	1.02E3	0.00E0	6.37E3	3.28E3	1.20E3	1.06E2
	0.00	3.48E3	5.87E3	2.12E3	0.00E0	3.58E3	5.84E3	2.42E3	1.21E2
	0.07	1.78E3	9.85E3	4.02E3	0.00E0	1.87E3	9.69E3	4.46E3	1.36E2
	0.14	8.38E2	1.53E4	7.00E3	0.00E0	9.17E2	1.50E4	7.58E3	2.37E2
	0.21	3.59E2	2.21E4	1.13E4	3.70E-1	5.66E2	2.18E4	1.19E4	4.63E2

Table 2.2: Pressure in psf/ft acting on each panel in MATLAB and CFD simulations

From the table, a good match between of pressures acting on the upper, lower forward, and nacelle panels. There is a discrepancy in the calculation of the pressure acting on the aft surface, with the MATLAB model underestimating the pressure acting against it. However, in either model pressure acting on the aft surface is the smallest pressure experienced so its contribution to aerodynamic forces will be relatively small. Plotting the calculated lift, drag, and pitching moment curves against each other on the following pages gives a better idea of the accuracy of the MATLAB model compared to CFD.

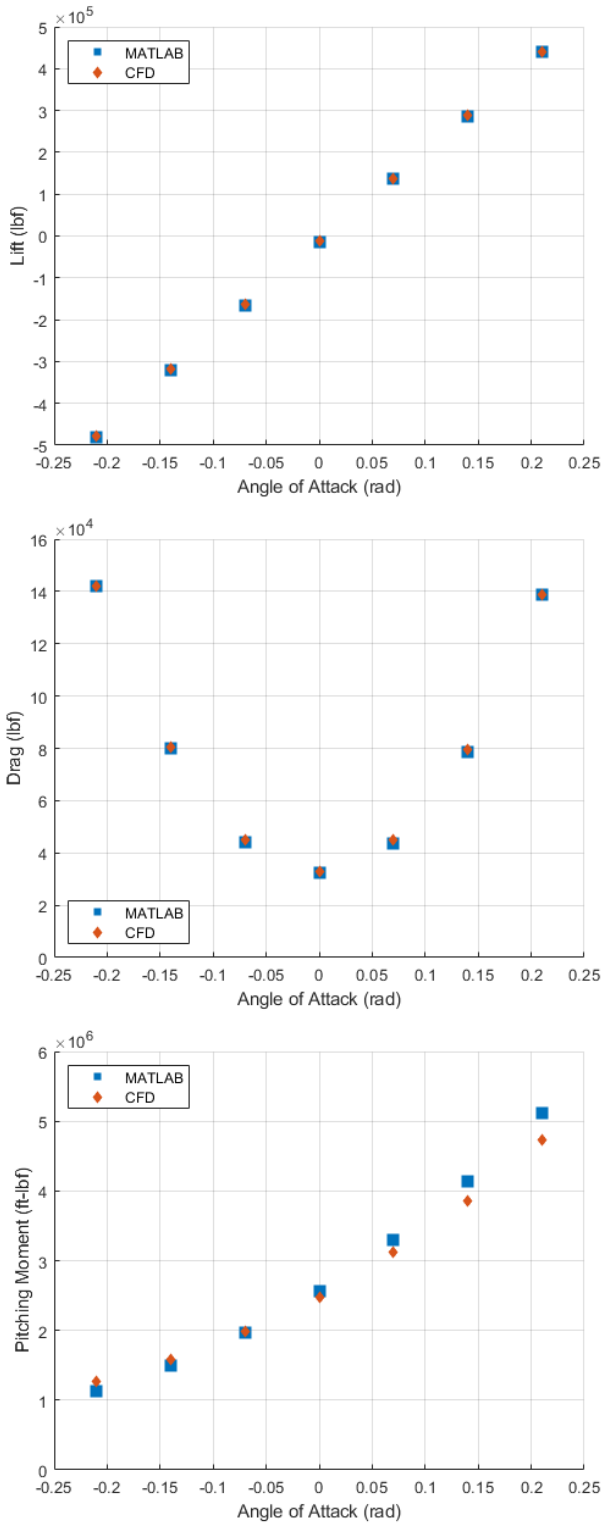

Figure 2.6: Panel method and CFD comparison for Mach 3

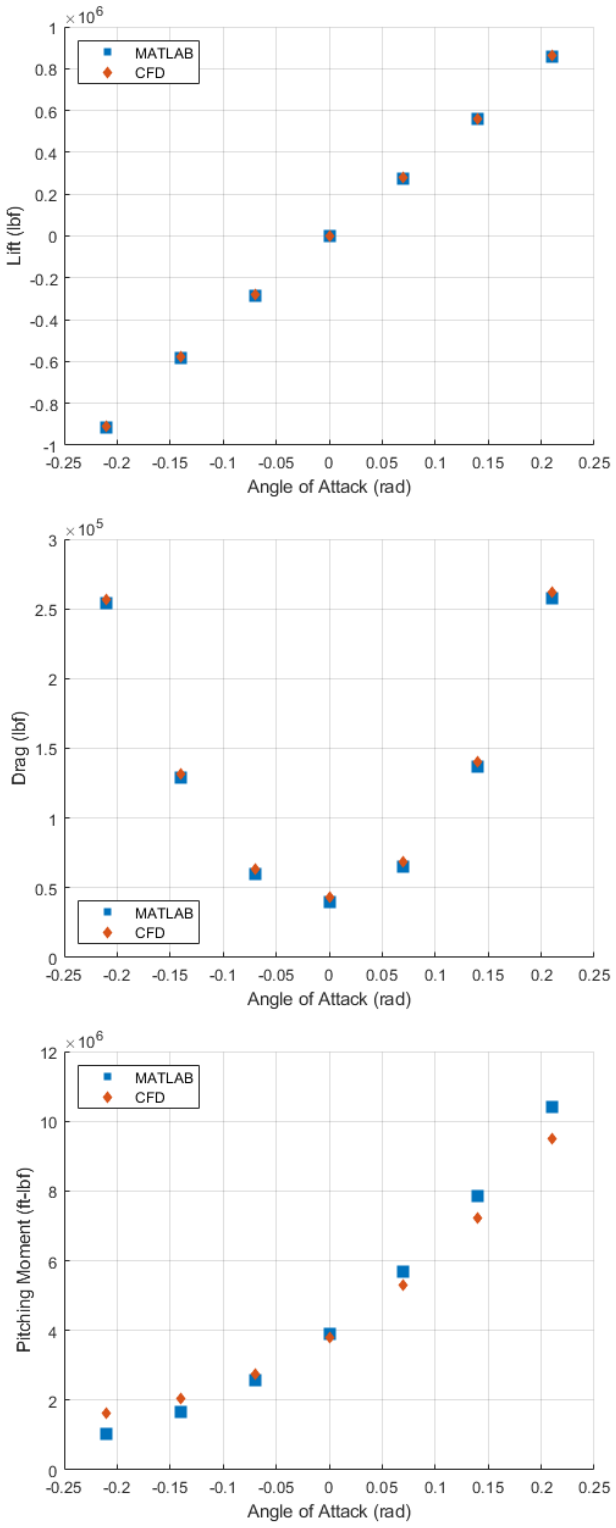

Figure 2.7: Panel method and CFD comparison for Mach 5

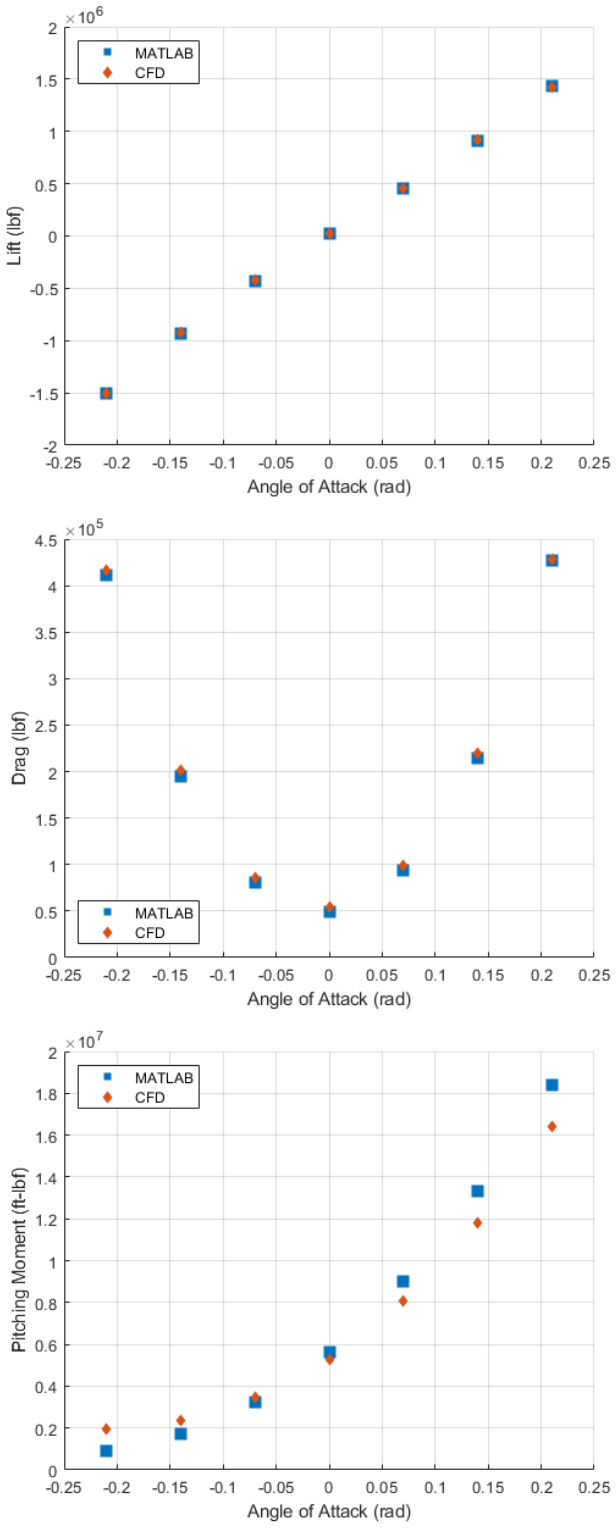

Figure 2.8: Panel method and CFD comparison for Mach 7

These results show good correlation of lift and drag as well as pitching moment for small angles of attack. The discrepancy in moment for large angles of attack can be explained by a nonuniform pressure across each panel. The average pressure across each panel matches well and thus lift and drag does as well. However, a shifting of the pressure towards the aft of the vehicle results at high angles of attack is not accounted for in the MATLAB model and thus results in MATLAB predicting a higher moment-curve slope. It should also be noted that while the moment-curve slope of the aircraft is positive, suggesting the vehicle is unstable, the results are generated by neglecting the contribution of the elevon which resides behind the center of mass and stabilizes the vehicle.

To verify the results of the temperature model, a similar approach was used to check the MATLAB model against the CFD results. Mach number and angle of attack are varied and the greatest temperature experienced by either the upper and lower forward surfaces is recorded. The ratio between this maximum temperature and the freestream temperature is calculated for all test points as it is independent of altitude. The results are shown in the table below and the figure on the following page.

	M_∞ \ α (rad)	-0.21	-0.14	-0.07	0.00	0.07	0.14	0.21
MATLAB Temperature (R)	3	720	660	600	600	660	710	790
	4	800	710	630	620	700	790	900
	5	900	770	670	660	760	880	1040
	6	1020	840	700	690	820	990	1210
	7	1150	920	740	720	890	1120	1404
CFD Temperature (R)	3	750	680	630	620	680	750	830
	4	860	750	670	660	750	850	950
	5	970	830	700	690	800	960	1130
	6	1100	910	750	730	900	1100	1320
	7	1240	1010	830	770	960	1210	1510

Table 2.3: Maximum flow temperature in R adjacent in MATLAB and CFD simulations

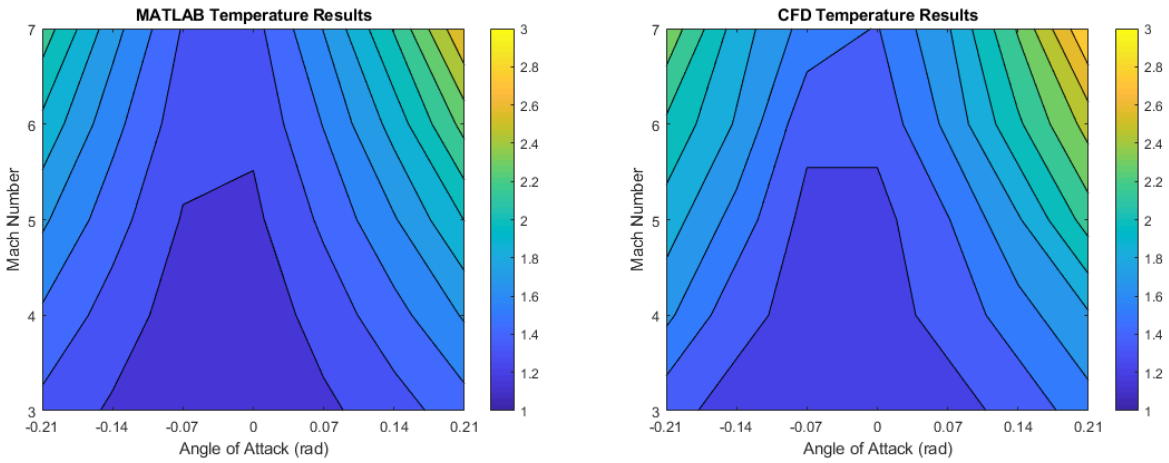

Figure 2.9: Leading surface temperature panel method and CFD comparison.

2.2 Zero Moment Model

With the full state model verified, a simplification was considered to reduce the complexity of the model. Specifically, for long flight durations, the aircraft can be expected to be trimmed for most of the flight. If the aircraft is statically stable, then the aircraft will naturally pitch to a stable angle of attack as deviations from the trimmed state will result in a counteracting pitching moment rotating the aircraft back to its trim state. In this case, only small perturbations are expected and can be verified. Based on the equation of motion for pitch rate from equation 2.1, the time derivative of pitch rate will be low when pitching moment is small or moment of inertia about the y axis is large. The latter condition is inherient to the aircraft and not something that can be controlled, in addition to the fact that a large moment of inertia would necessitate higher pitching moments experienced by the aircraft anyway. Reducing the pitching moment however, is something achievable with the elevon control available to the model.

The pitching moment of the aircraft is largely influenced by the elevon deflection. This is because the elevon is offset from the center of mass, can independently deflect relative to the rest of the aircraft, and can generate aerodynamic force necessary to overcome the pitching moment created by the lifting body. Elevon deflection may have a relatively small impact on lift and drag relative to the lifting body, but it principally affects pitching moment [9]. To reduce the complexity

of the current full state model, it is assumed that the elevon deflection required to trim the aircraft is always active. Elevon deflection is then no longer considered a control for the system but rather a function of states. When the aircraft is operating within expected flight conditions such as reasonable angle of attack, an elevon deflection which trims the aircraft should exist if the vehicle is well designed. With the assumption that the vehicle is always trimmed, pitching moment now becomes very small if not zero. When this happens, the time derivative of the pitch rate also goes to zero. Now there are only five equations of motion to consider, reducing the complexity of the problem to the following:

$$\begin{aligned}
\dot{x} &= v\cos(\theta - \alpha) \\
\dot{z} &= v\sin(\theta - \alpha) \\
\dot{\theta} &= Q \\
\dot{\alpha} &= \frac{g\cos(\theta - \alpha) - L/m}{v} + Q \\
\dot{v} &= -\frac{D}{m} - g\sin(\theta - \alpha) \\
\delta_e &= f(M_\infty, \alpha)
\end{aligned} \tag{2.23}$$

However, now the vehicle cannot be controlled to follow a given trajectory as elevon deflection is completely controlled by a closed loop that only wants to trim the aircraft. To open the design space for a control, pitch rate is assumed to be directly controllable as a 'psuedocontrol'. Provided pitch rate is very small, the elevon can deflect to a new position associated with the angle of attack after rotating and zero the small pitching moment. For a statically stable aircraft, adjusting the elevon deflection to this new trim condition will naturally rotate the aircraft to the new trim state. For statically unstable aircraft, this assumption will not work as any perturbation will cause the aircraft to pitch away from the trim state rather than towards it. Further feedback is needed to ensure the zero moment model cooperates with statically unstable aircraft.

As the elevon deflection does not influence the maximum temperature experienced on the body, the process for calculating the maximum temperature for the zero moment model is the same as the full state model.

2.2.1 Trimming Elevon Deflection

The elevon deflection required to trim the aircraft is associated with the angle required to reduce the pitching moment to zero. The total pitching moment can be broken up into each panel's components, and the moment created by the elevon must cancel out the moment applied by the body panels.

$$0 = M_{body} + M_{\delta_e} \tag{2.24}$$

Pitching moment induced by pressure acting on the body panels was calculated as in subsection 2.1.1. Neither elevon deflection nor the pressure acting on its upper and lower surface were considered at this point. After calculating the body moment, MATLAB's fsolve() command was used on equation 2.24. This iterative nonlinear equation solver considers the freestream Mach number and angle of attack while varying elevon deflection using an iterative method. fsolve() will solve for the angle of attack needed to cancel the body moment and also return the force acting on the elevon in the body frame x and z directions. These forces derived after solving the trimmed elevon deflection will be used to calculate the total aerodynamic force on the vehicle and finally the lift and drag acting on the trimmed aircraft.

2.2.2 Verification of Zero Moment Model

To verify the results of the zero moment model, two sample trajectories were created. The first trajectory will be a dynamic simulation of the zero moment model with the psuedocontrolled pitch rate set to zero. Meaning that the aircraft will not rotate with respect to an inertia reference frame, although angle of attack was expected to increase due to downwards acceleration by gravity. The second trajectory is more complex, a sinusoidal input with small magnitude will be commanded for the psuedocontrol making the aircraft oscillate in pitch. During the dynamic simulation of the zero moment model, the elevon deflection required to trim the vehicle was recorded for each point in the

simulation. Afterwards, two dynamic simulations of the full state model were created. This time, the commanded control of the elevon deflection was the open loop history derived from the zero moment model. If the zero moment model is a good representation of the full state dynamics, both models should follow similar trajectories. This process was performed for both a statically stable aircraft and a statically unstable aircraft using MATLAB's ode45() dynamic simulation function [10]. The unstable aircraft was identical to the stable aircraft with singular change that the center of gravity has been defined further down the length of the vehicle $\bar{x} = -55$ft. The two trajectories have the following initial states and final time:

x (ft)	z (ft)	θ (rad)	α (rad)	v (ft/s)	Q (rad/s)	t_f (s)
0	6e4	0	0	5e3	0	120

Table 2.4: Initial states and final time of zero moment model verification simulations

The two psuedo-controlled inputs for pitch rate are shown below, with the trajectories of the zero moment and full state model shown on the following page:

$$\begin{aligned} Q(t) &= 0 \\ Q(t) &= 0.01 \cos\left(\frac{\pi}{30}t\right) \end{aligned} \tag{2.25}$$

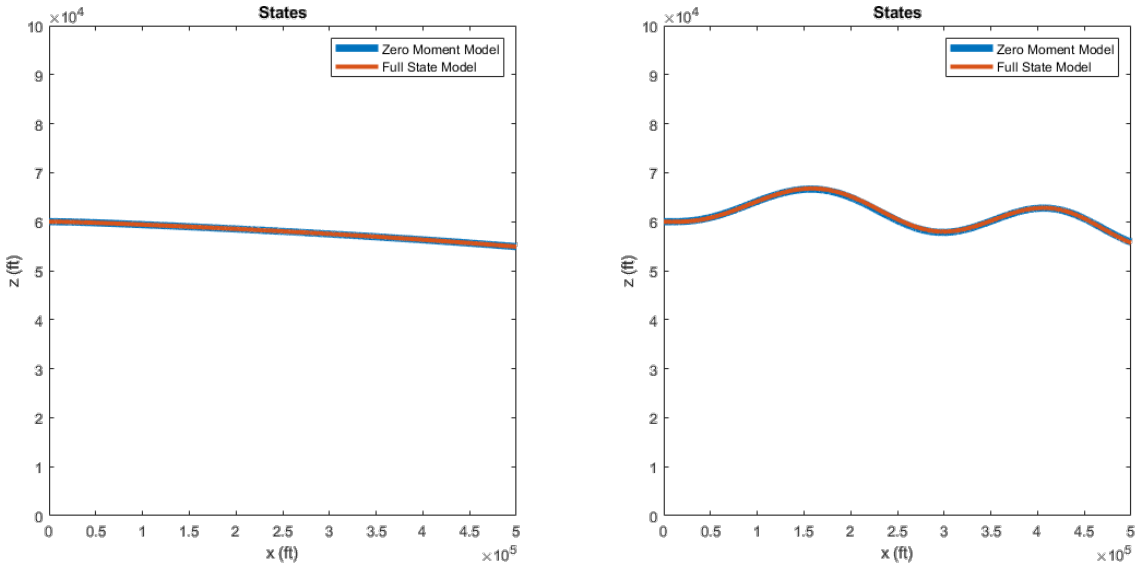

Figure 2.10: Stable zero moment and full state model comparison

These results show a strong correlation between the zero moment model and the full state model for the statically stable vehicle. Using only the elevon deflection output of the zero moment model, the dynamic model easily follows both trajectory and remains stable even in oscillatory cases such as the second trajectory.

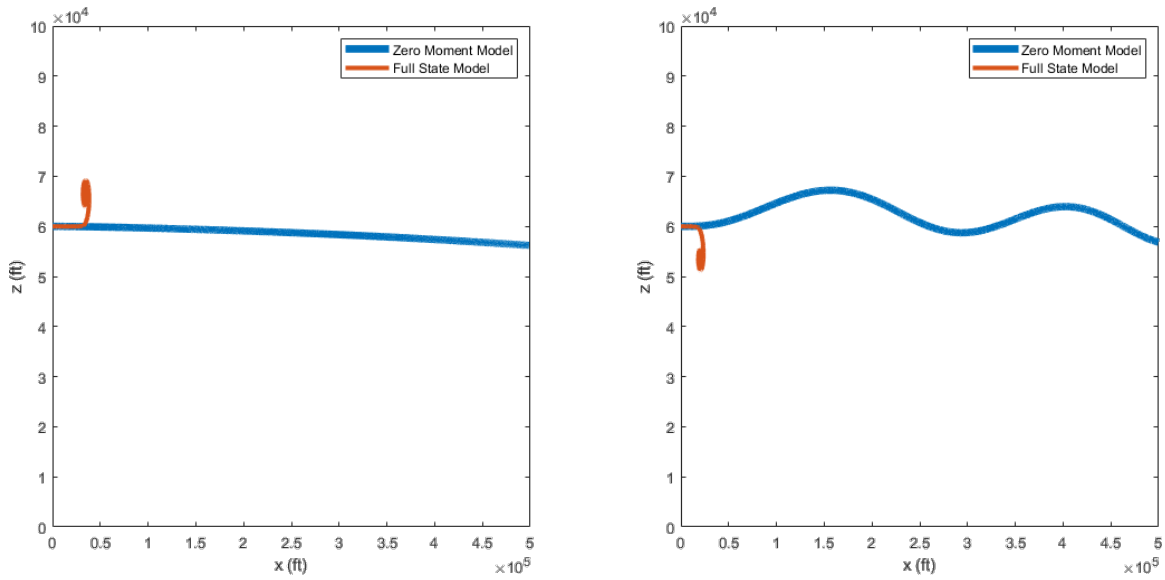

Figure 2.11: Deviation of zero moment model for unstable aircraft

Unfortunately, the full state dynamic model of the statically unstable aircraft does not match the zero moment model. Almost immediately, the vehicle destabilized and was thrown completely off course. The spiral singularity is a result of saturating the maximum angle of attack to 0.21 radians beyond this point, the vehicle is assumed to have been destabilized and lost. As predicted, when the vehicle is unstable perturbations in angle of attack are not naturally dampened but rather explode and destabilize the aircraft. Ideally the vehicle is statically stable, but it is possible to stabilize this behavior using feedback control.

From the zero moment trajectory, elevon deflection is not only recovered but also all state histories. In addition to using the elevon deflection history, a feedback loop will also be placed on the full state history. Effectively linearizing about the trim condition, the angle of attack error between the full state and zero moment solution will be used to offset the elevon deflection command. Only a small gain is needed to accomplish this.

$$\begin{aligned} \alpha_\epsilon &= \alpha_{full}(t) - \alpha_{0M}(t) \\ \delta_{ecorrected}(t) &= \delta_{e0M}(t) + K\alpha_\epsilon \end{aligned} \tag{2.26}$$

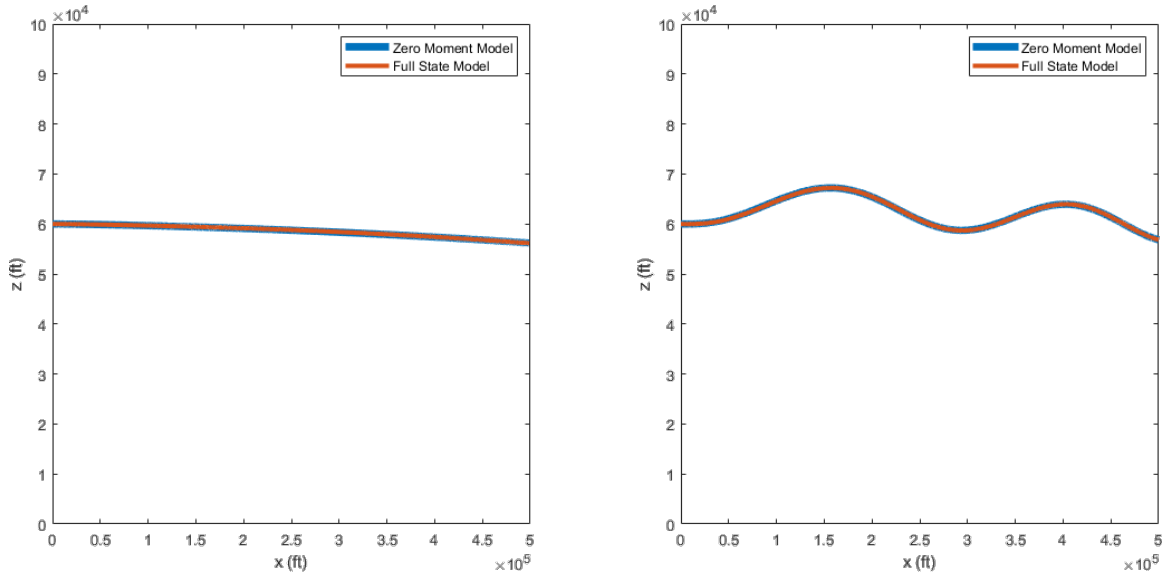

Figure 2.12: Feedback stabilization of unstable aircraft

The error between the reduced order model and the full state model is relatively small compared to the distance traveled. After moving over 500,000 feet, the zero moment model only deviates from the full state model by a few hundred feet at most. The error between the models for the two trajectories is shown below:

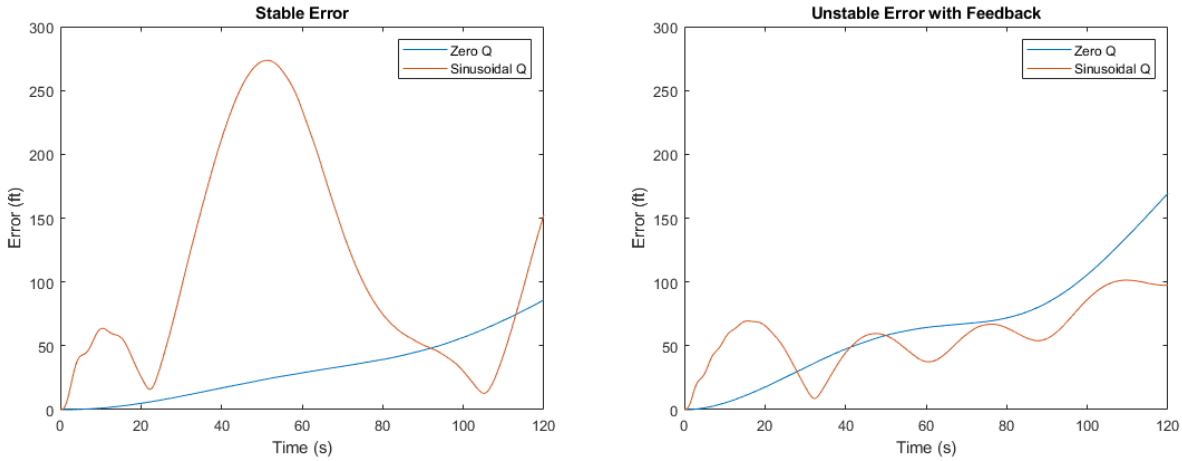

Figure 2.13: Error of stable and unstable with feedback zero moment models

The sinusoidal psuedocontrolled input does result in some oscillations in error, but this is expected given the nature of the input. Interestingly, the unstable aircraft has reduced error compared to the stable aircraft in the sinusoidal input. This is likely due to the proportional feedback controller steering the aircraft to keep it on course. There is nothing preventing a similar feedback controller from being implemented on a stable aircraft, which could reduce error in the stable case as well.

3. LOOKUP AND TFC METHODS

When simulating the hypersonic dynamics of the vehicle, calculating aerodynamic forces and moment take up a bulk of the computation time. This is expected as solving for the oblique shock angles is an iterative process that has to be run for every point along the trajectory of the aircraft. This creates problems with solving nonlinear optimal control problems using this model, as methods such as DPIP will iterate the entire control history. So not only will dynamics have to be calculated across the entire trajectory, but the trajectory will be updated each time a new control is solved for. New dynamics then have to be solved for given the new control history. This results in a situation where solving optimal control problems involving this hypersonic model will take a substantial time to solve.

Other methods can be used to approximate the aerodynamic forces of the vehicle. If the forces can be estimated accurately and in a more time efficient method then solving with the current method, the overall time to solve the dynamics of the vehicle and optimal control problems related to it could be reduced.

3.1 Lookup Method

The first method considered is a "Lookup Table" method. This method has been used in other studies and presents a way to approximate the aerodynamic forces quickly while solving for the dynamics of the vehicle [11]. This method works by creating a mesh of flight conditions using various combinations of states and control inputs. For each unique combination, aerodynamic forces and pitching moment can be solved and stored on this mesh. All of this is done before running the dynamics of the vehicle, during the actual solution process of the dynamics the values of lift drag and pitching moment are interpolated from the mesh.

3.1.1 4 Input Lookup

To apply this method, first the inputs to the mesh were determined. From the hypersonic model developed in the previous chapter, each state and control is considered depending on if it contributes to the calculation of the aerodynamic forces.

State/Control	Contribution
x	None
z	Affects freestream pressure, temparture, and Mach number
θ	None
α	Affects where oblique shock and expansion waves occur as well as their intensity
v	Affects freestream Mach number
Q	None
δ_e	Affects forces created by elevon

Table 3.1: State and Control Contribution to Aerodynamic Forces and Moment

This shows that the aerodynamic forces of the full state model are dependent on four inputs: altitude (z), angle of attack (α), velocity (v), and elevon deflection (δ_e). These 4 inputs correspond to the results published results [11]. Now, boundaries for these values will be established to create the mesh within. The boundaries should encompass all expected flight conditions so that aerodynamic forces and moment can be interpolated at any point. These boundaries were chosen to be:

Input	Boundaries
z (ft)	$[0 \quad 1\text{E}5]$
α (rad)	$[-0.21 \quad 0.21]$
v (ft/s)	$[3000 \quad 7000]$
δ_e (rad)	$[-0.21 \quad 0.21]$

Table 3.2: Boundaries on aerodynamic inputs

With these boundaries established, a linearly spaced set of points was created between each boundary. Each of these nodes functions as part of the mesh that will be created to be interpolated from. For this experiment, a set of 16 linearly spaced nodes were established for each input variable. This creates three empty $30 \times 30 \times 30 \times 30$ meshes for lift, drag, and pitching moment respectively. These meshes are formatted as 4-dimensional arrays in MATLAB with each dimension representing an input and each step along that dimension representing a defined node.

To store information in the arrays, a MATLAB script was created to iterate through each unique combination of inputs. Lift, drag, and pitching moment are then calculated at this point and stored in the corresponding indices in each array.

At the end of this process, lookup tables for lift, drag, and moment are completely filled in and saved. Now given a set of input conditions, the inputs can be used along with a n-dimension interpolator such as MATLAB's ninterp() to approximate each force and moment. ninterp() takes several arguments: the lookup table which was just created, lists of the nodes corresponding to each index of the lookup table, and the queried list of inputs of the current flight condition.

3.1.2 4 Input Lookup Results

With the verification of the direct method model, the direct model is assumed to be the most accurate model of the system. Therefore, this method and future methods will be compared to the direct model. To test the effectiveness of the 4 input lookup table method, a script was created in MATLAB which compares the output of the lookup table method to the output of directly calculating the aerodynamic forces. This is done by first randomly generating 10,000 flight conditions within the input domains. For each flight condition, the lift, drag, and pitching moment were calculated using the direct method and are stored in three separate arrays. Then the aerodynamic forces and moment are approximated using the lookup table method and stored in another three arrays.

The variance and average absolute value of the error between the direct calculation and the lookup table method was then calculated. Unfortunately, relative error was not be meaningful in this situation. For example, a flight condition which is trimmed would result in zero moment and the relative error in this case will be undefined. In cases where lift and moment are small, even

accurate approximations by the lookup method will result in large relative error.

To better show the accuracy of the lookup method, a histogram of all measurements was created and is shown below. Another bar is superimposed on the graph with the width of the bar equal to the maximum error. The thinner the bar compared to the domain of the histogram, the more accurate the method is.

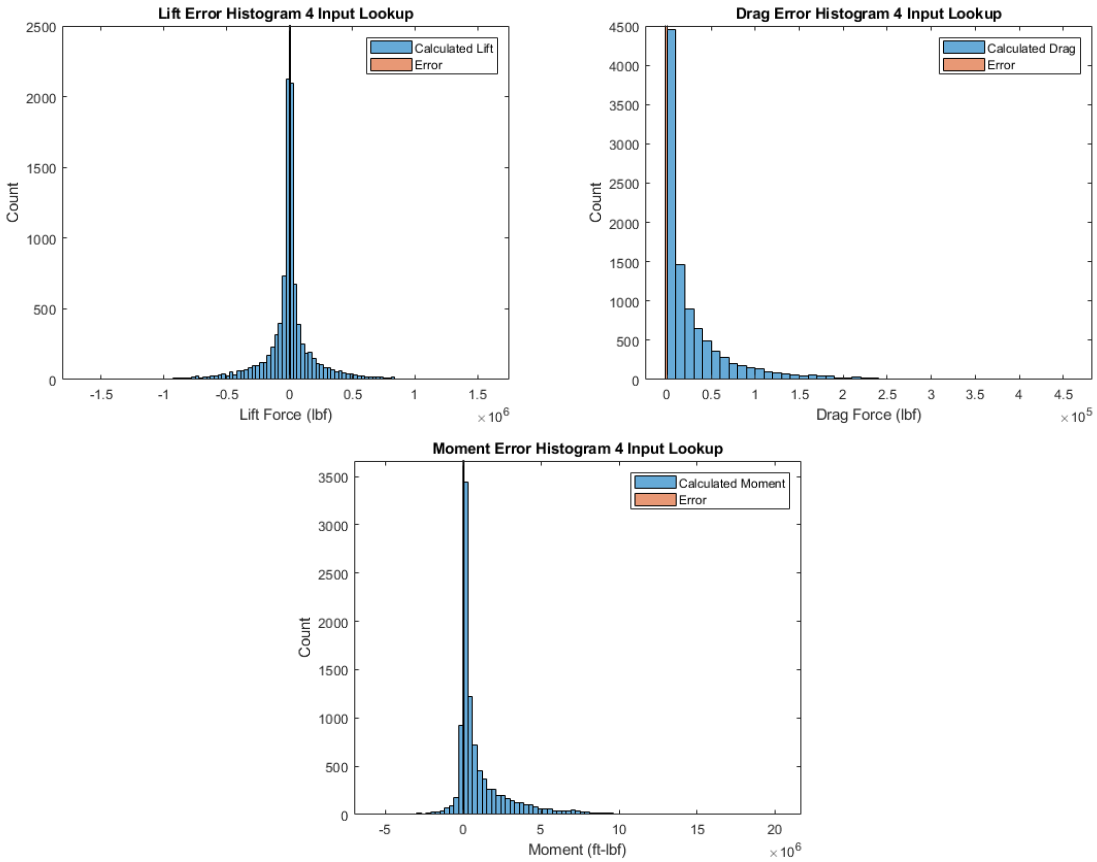

Figure 3.1: Histogram of tested forces and moment with 4 input lookup table error bounds

We see adequate performance of this 4 input lookup table method. An easy way to increase the accuracy of this method would be to increase the number of elements within the table. However, other methods were be investigated instead.

3.1.3 Mach Lookup Table

The limitations of the 4 input lookup table can be addressed. Of the four inputs, altitude has very predictable effects on the aerodynamic forces. Removing a dimension of the table will drastically reduce the memory space required. From the hypersonic model, it was apparent that all pressure acting on each panel of the aircraft is a fraction of the freestream pressure for a given Mach number, angle of attack, and elevon deflection. A new lookup table could be created that stores the aerodynamic forces and moment for a given flight condition normalized to sea level. Then outside of the lookup table, the pressure ratio between the freestream at the current altitude and sea level can be used to multiply the lookup table output.

The only remaining issue with this method is that temperature and the speed of sound change with altitude. Thus, the freestream Mach number also depends on altitude. To account for this, the velocity input to the lookup table was nondimensionalized to Mach number instead. This can be done before using the lookup table. First, the freestream pressure ratio and the freestream temperature ratio was interpolated from a standard atmosphere table given the altitude. Then, the speed of sound was calculated using equation 2.2. The velocity of the aircraft could then be nondimensionalized into Mach number using the speed of sound and equation 2.3. Each of the lookup tables was then used to calculate the aerodynamic forces and moment normalized to sea level. Finally, the true dynamic effects were found by multiplying the interpolated lookup table value by the pressure ratio.

For this research, three $30 \times 30 \times 30$ tables are created for lift, drag, and pitching moment and used alongside the same standard atmosphere table used in the hypersonic model.

3.1.4 Mach Lookup Table Results

With this simplification, each of the 6.48 MB 4 dimensional lookup tables have been simplified to 216 kB tables. The performance of these tables are actually expected to be better than the 4 input table in both accuracy and computation time. This is because 3 degree interpolation is less complex than 4 degree interpolation, which saves time, and because the standard atmosphere table is more refined than the lookup table dimension associated with altitude, which increases accuracy.

Much like the 4 input table, the Mach lookup table is tested against the direct calculation method in the same way. The performance is shown in the histograms below.

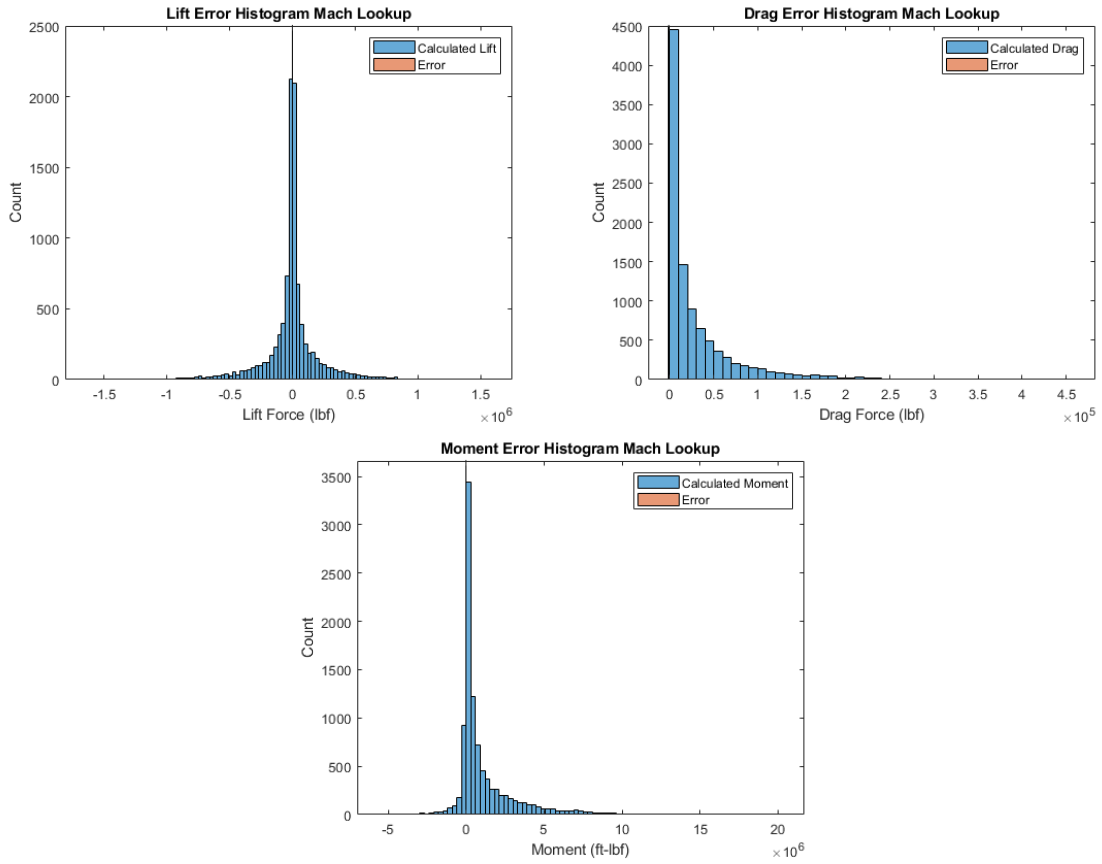

Figure 3.2: Histogram of tested forces and moment with Mach lookup table error bounds

3.2 Theory of Functional Connections

The equations of aerodynamic forces and moment as a function of Mach number, angle of attack, and elevon deflection are smooth and continuous and are confined to a specified domain. The Theory of Functional Connections allows the creation of orthogonal manifolds to approximate these multidimensional functions [5]. As described in the introduction, these orthogonal manifolds are scaled and combined linearly to form a single manifold which can be evaluated to estimate whatever function they have been trained on. In this case, manifolds estimating lift, drag, and pitching moment are desired. Fortunately, most of the groundwork has already been laid to train a vector of scaling factors for these manifolds.

For the lookup table methods, evaluation points were created by linearly spacing a mesh along each axis of Mach number, angle of attack, and elevon deflection. A similar mesh must be created for the method derived from Theory of Functional Connections. However, instead of linearly spacing evaluation points, nodes were be chosen as collocation points as they give a more accurate mapping [5]. The set of collocation points is first described on the z-domain $[-1, +1]$.

$$z_i = \cos\left(\pi \frac{i-n}{n-1}\right) \qquad i \in [1, n] \tag{3.1}$$

Like the lookup table method, 30 nodes will be chosen across the three inputs of Mach number, angle of attack, and elevon deflection. From these nodes, another mesh of unique combination of input values was created in the same way as the lookup table method only with a collocated mesh instead of a linear one. To evaluate the corresponding nodes of the input values based on the collocation points created on the z-domain, a scaling factor was be created for each input value [5].

$$c_x = \frac{z_{max} - z_{min}}{x_{max} - x_{min}} = \frac{2}{x_{max} - x_{min}} \tag{3.2}$$

The scaling factor can be used with the domain of the inputs as described in table 3.2. The associated points with each collocation point for the three inputs are [5].

$$x_i = x_{min} + \frac{z_i + 1}{c_x} \tag{3.3}$$

With these collocation points on the input variable domains, a collocated mesh was then created in a similar way to the lookup table method. This mesh was stored as an array $\mathbb{B}$.

3.2.1 Creation of Xi Vector

After the creation of the collocated mesh, a least squares solution algorithm can then be used to optimize the weights of a series of orthogonal manifolds. As the orthogonal polynomials needed to generate the orthogonal manifolds are recursive, the number of polynomials and the number of manifolds used can be chosen. The maximum number of unique manifolds that can be created from $N_{\mathcal{L}}$ polynomials is reliant on how many input variables are considered.

$$N_{\mathcal{B}} = N_{\mathcal{L}}^{N_{input}} \tag{3.4}$$

As more manifolds are considered, the accuracy of the approximated solution cannot decrease [5]. In a case where a manifold has no possible way to increase accuracy of the estimation, its weight will be optimized to zero and it will have no effect on the solution. In single input cases where many orthogonal polynomials are considered, the accuracy of the estimation can converge to machine error after around 10 basis functions [5]. Later, several tests where the number of basis functions will be varied and the error of each iteration were compared. It is expected that every manifold should contribute at least some accuracy to the least squares solution, so the maximum number of manifolds as calculated by equation 3.4 will be considered for each collection of basis functions.

Using Legendre orthogonal polynomials, the basis functions for the manifolds were created through the process described in section 1.3. Each collocation point mapped to the z domain of the Legendre domain were evaluated for each basis function. To determine the points of a manifold, each basis function is considered as a multiplicand for two other basis function, the order of which corresponds each of the inputs. For example, the first manifold $\mathcal{B}_0$ was the product of the

first Legendre polynomial evaluated on the z domain location corresponding to a mapped Mach number value, the first Legendre polynomial evaluated for angle of attack, and the first Legendre polynomial evaluated for elevon deflection. The second manifold $\mathcal{B}_1$ would be in a similar way as $\mathcal{B}_0$, but the final multiplicand would be the second Legendre polynomial was evaluated for elevon deflection instead of the first polynomial.

As the manifolds are generated, they were stored in an array for the purposes of finding a least squares solution. As each manifold has 3 inputs, a 3 dimensional array is needed to store an individual manifold. However, to collect all of the manifolds, a 4 dimensional array $\mathcal{A}$ will be created with the 4th dimension corresponding to each unique manifold. After creating the 4 dimensional array, one last adjustment is needed to allow an least squares algorithm to train the manifold weightings to the collocated lookup table.

A robust method for solving a least squares problem requires both the orthogonal manifolds and lookup table to be presented in a 2 dimensional matrix A and a 1 dimensional vector $\boldsymbol{b}$ respectively. To reorganize the elements of each of the arrays, the MATLAB function reshape() command will be used [12]. For the newly arranged A array, each column of must correspond to a unique manifold, with the additional dimensions of the $\mathbb{A}$ array appended as additional rows.

$$\begin{aligned} A &= \text{reshape}(\mathcal{A}, [\,], N_{\mathcal{B}}) \\ \boldsymbol{b} &= \text{reshape}(\mathbb{B}, [\,], 1) \end{aligned} \tag{3.5}$$

For rearranging $\mathcal{A}$, the first argument gives $\mathcal{A}$ as the array to be rearraged, the second argument allows the as many rows as necessary to exist, and the final argument will ensure that the number of columns of the array is equal to the number of manifolds [12]. The collocated mesh was rearranged into a vector in a similar way, with the first argument being the mesh, the second argument permitting as many rows as necessary to rearrange, and the final argument ensuring the rearraged array is a vector.

Solving for the $\boldsymbol{\xi}$ weighting vector was done using MATLAB's matrix left division operator \

[13]. A was not expected to be a square matrix and the left division operator will tend to use QR decomposition to solve the least squares problem.

$$\begin{aligned} A\boldsymbol{\xi} &= \boldsymbol{b} \\ \boldsymbol{\xi} &= A\backslash\boldsymbol{b} \end{aligned} \tag{3.6}$$

It should be noted that the space complexity for creating the $\mathcal{A}$ is quite poor. Because $\mathcal{A}$ contains collocated meshes for each basis manifold, its space complexity is $\mathcal{O}\left((nN_{\mathcal{L}})^{N_{inputs}}\right)$. In this case, increasing n and $N_{\mathcal{L}}$ will increase the accuracy of the solution, whereas N_{inputs} is dictated by the problem itself. Clearly, N_{inputs} is the most critical term in determining the size of $\mathcal{A}$ to the point where it can be a limiting factor.

3.2.2 Estimating Aerodynamic Forces using the Xi Vector

With a $\boldsymbol{\xi}$ vector now solved for lift, drag, and moment, it was possible to estimate these parameters by evaluating the manifolds used to create them. Given a set of input states, the states are mapped to the z-domain of the orthogonal polynomials used, the manifolds are evaluated at those points, and a weighted sum of the evaluated points was taken [5].

In a similar manner to equation 3.3, the inputs were mapped to the z-domain using the scaling factor calculated by equation 3.2. The equation for mapping back to the z-domain is:

$$z_i = (x_i - x_{min})c_x - 1 \tag{3.7}$$

Once each input was mapped to the z-domain, each manifold is evaluated at the point of interest. This was done by evaluating every orthogonal polynomial that composes the manifolds at the mapped points, then multiplying every unique combination to get the evaluated point on the manifold. All that remained was to scale each manifold by the corresponding solved scaling value in the $\boldsymbol{\xi}$ vector and sum the result [5]. Interestingly, because the orthogonal polynomials used as basis functions can be generated recursively and all the "information" needed to compute aerody-

namic forces is stored in the ξ which is very small compared to a lookup table, the total memory required to use the TFC method during simulation is quite small even if the space complexity to find the solution is very large.

3.2.3 Effect of Number of Basis Functions

To test how the number of basis vectors affects the accuracy and computation time of the TFC method, several ξ vectors were created of varying length. This was done by incrementing the number of polynomials used to create unique manifolds from the range of 1 to 10. The verification method used in subsections 3.1.2 and 3.1.4 was repeated for each created ξ vector. The memory required to store the ξ vector was considered for all cases, but it was so small as to be irrelevant to any memory constraints. After running all test cases and comparing to the direct calculation, the average absolute value of the error in lift was recorded as a way to measure accuracy. The time to calculate all forces and moment for all test cases was also recorded. The results are shown in the table below:

Polynomials	Manifolds	Memory (Bytes)	Average Lift Error (lbf)	Time (s)
1	1	8	1.29E5	0.076
2	8	64	7.22E3	0.068
3	27	216	3.20E3	0.059
4	64	512	1.83E2	0.056
5	125	1000	1.16E2	0.077
6	216	1728	8.77E0	0.088
7	343	2744	3.43E0	0.120
8	512	4096	2.10E0	0.193
9	729	5832	7.61E-1	0.180
10	1000	8000	5.16E-1	0.237

Table 3.3: Performance of TFC method based on number of basis function

As expected, increasing the number of basis functions increases the accuracy of the solution. However, computation time penalties begin to collect beyond 5 basis functions. Given that there are diminishing returns in terms of accuracy as number of basis functions increases for future tests ξ vectors created from 6 basis functions will be used to optimize accuracy without hindering computation time greatly.

3.3 Method Comparison

The average absolute value of lift, drag, and moment absolute error were calculated for each method when they were tested in previous sections. For all methods, the error was calculated in relation to the output of the direct method. The 10,000 flight conditions used are also the same conditions tested across all methods to eliminate any random variation from method to method. The results are shown in the subsequent table and bar graphs:

Method	Panel	4 Input Lookup	Mach Lookup	TFC
Computation Time (s)	117.30	10.60	9.62	0.13
Average Lift Error (lbf)	—	225.5	35.7	8.8
Average Drag Error (lbf)	—	115.0	65.5	9.0
Average Moment Error (ft-lbf)	—	2480	675	271

Table 3.4: Performance of TFC method based on number of basis function

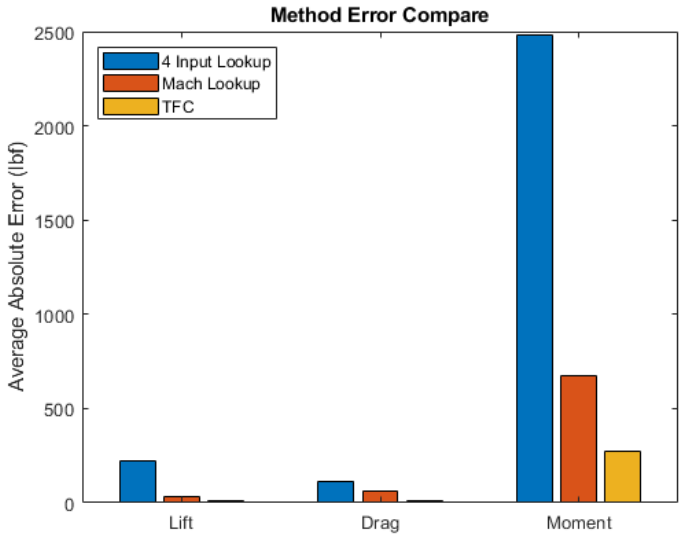

Figure 3.3: Average absolute error of lift, drag, and moment for each developed method

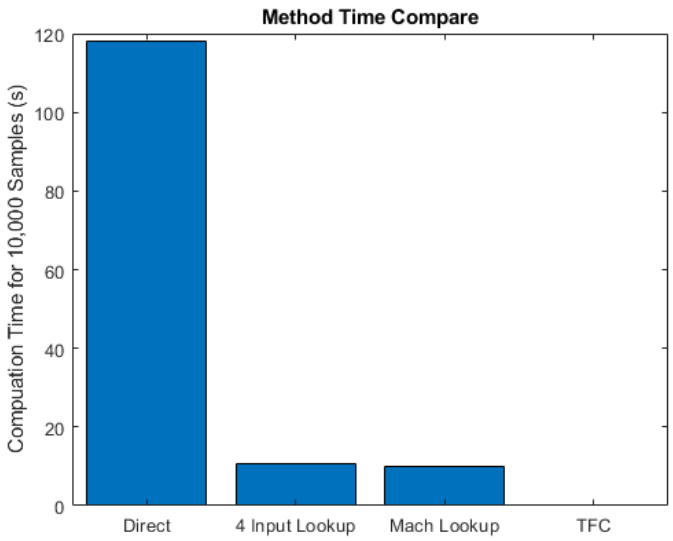

Figure 3.4: Average absolute error of lift, drag, and moment for each developed method

The direct panel method, although assumed to be the most accurate, takes far longer than any other approximation. The long computation time to calculate is what drove the need to develop faster approximations. As expected, the Mach lookup table method performed significantly better in terms of accuracy and slightly better in terms of computation time. The best performing method however is the TFC method, which computes almost instantly compared to all other methods and has significantly less error in all cases. The Theory of Functional Connections gives us a strong method for approximating aerodynamic forces.

It should also be noted that the development of the Mach Lookup table is integral to the creation of the TFC method. Attempting to solve a least squares problem for a 4 input mesh is currently infeasible for the hypersonic dynamics. Consider that a $30 \times 30 \times 30 \times 30$ collocated mesh would be needed for each basis manifold. To achieve a similar accuracy with 6 basis functions, $6^4 = 1292$ manifolds are needed. This results in an array with over a billion elements. With 8 bytes needed to store each element, the $\mathbb{A}$ array alone is over 8GB in memory size which is too unwieldy to use effectively.

3.4 TFC Applied to Zero Moment Model

It has been shown that lift, drag, and moment can be accurately estimated using various methods including the method derived from the Theory of Functional Connections. With aerodynamic forces and moment derived, the full state model can be simulated quickly with minimal error. However, the zero moment model is also of interest as it offers a simplified state space to create flight trajectories. This section will focus on applying the TFC method to the zero moment model.

3.4.1 Approximating Trimmed Lift and Drag

The method to calculate the lift and drag of the trim condition has been developed in section 2.2. The lift and drag forces depend on atmospheric pressure, velocity, and angle of attack. However, these forces for the zero moment model were simplified through the same process as in section 3.1.3. The velocity of the aircraft was nondimensionalized to Mach number and air pressure was normalized to sea level. When the aerodynamic forces were calculated at sea level, they are multiplied by the ratio of air pressure at the current altitude to air pressure at sea level. Unlike the full state model, pitching moment is not needed and only aerodynamic forces are required.

Calculating lift and drag normalized to sea level for the trimmed aircraft now only depends on two inputs: Mach number and angle of attack. Elevon deflection is already a function of these two input variables and is already accounted for in the lift and drag. Using TFC, a linear combination of 2 input orthogonal manifolds (the manifolds can also be referred to as orthogonal surfaces in the case of 2 inputs) can be created to approximate lift and drag.

Before creating the collocated meshes for lift and drag, the input boundaries must be established such that a nontrimmable flight condition does not occur on the mesh. An untrimmable state would be where the elevon deflection required to trim the aircraft exceeds the deflection limits (0.21 rad). The flight domain of Mach 3 to 7 and angle of attack -0.21 to 0.21 is investigated and the elevon deflection required to trim is recorded for each point.

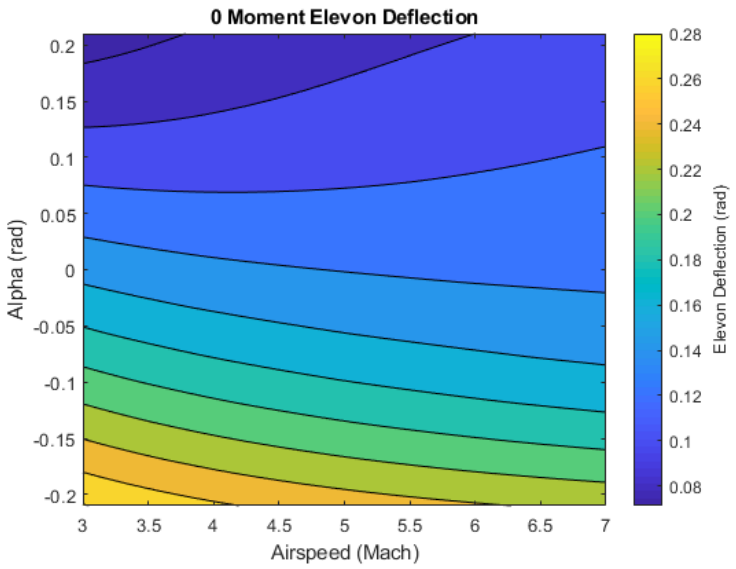

Figure 3.5: Elevon deflection required to trim the aircraft at various alpha and Mach speeds

According to the contour plot, a valid elevon deflection does not exist when angle of attack is less than -0.10 at high Mach numbers. The flight domain was then be restricted to angle of attack from -0.10 to 0.21. A collocated mesh could now be created with this restricted domain.

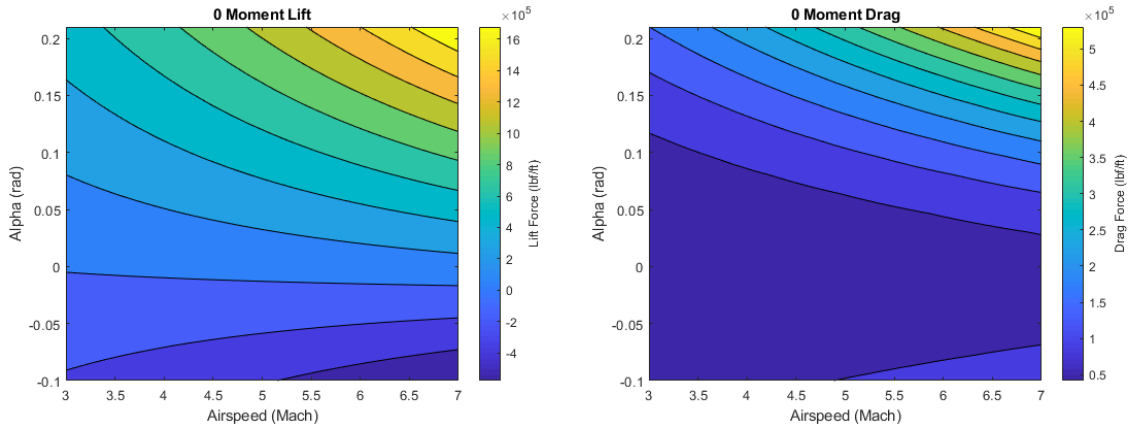

Figure 3.6: Lift and drag forces of the trimmed aircraft

With the new collocated mesh, the TFC method was successfully applied using 6 Legendre polynomials as basis functions resulting in an approximation with minimal error.

3.4.2 Approximating Maximum Temperature and Trimmed Elevon Deflection

One of the reasons why the TFC method approximates lift and drag well is because they are continuous differentiable functions [5]. TFC should also be able to be applied to calculating maximum temperature on the forward surface and the trimmed elevon deflection for the same reasons. Both functions are continuous and a function of Mach number and angle of attack of the aircraft. The trimmed elevon deflection is differentiable but there are points where the maximum temperature calculation is not smooth. Specifically, at the point where two oblique shock waves occur in front of the aircraft and the shock angles in front of the upper and lower surface are identical. Perturbing the angle of attack in either direction will result in the temperature of either surface increasing, so the function is not smooth at this location.

These nondifferentiable points should not be a problem despite the fact that the TFC method will incur small inaccuracies near this point as it tries to approximate with smooth functions. The point where shock angles are identical in front of the upper and lower surface will also correspond to a local minima in temperature with respect to angle of attack as perturbing angle of attack in either direction will increase temperature. In the case of this hypersonic vehicle, only maximum temperature constraints are a concern so it is acceptable for small error in temperature calculation to occur when temperature is minimized so long as it remains an inactive constraint.

A final consideration when calculating maximum temperature is that much like the Mach lookup table simplification, when airspeed is nondimensionalized to Mach number the calculation for temperature presents itself as proportional to the freestream temperature. From equation 2.22 the fraction is a function of Mach number of angle of attack. This 2 input function could then be approximated by TFC assuming sea level temperature and then scaled by the temperature ratio of the current altitude and sea level much like how sea level pressure was assumed in the Mach lookup table method.

With this in mind, the TFC method was then used to approximate maximum temperature ratio and elevon deflection.

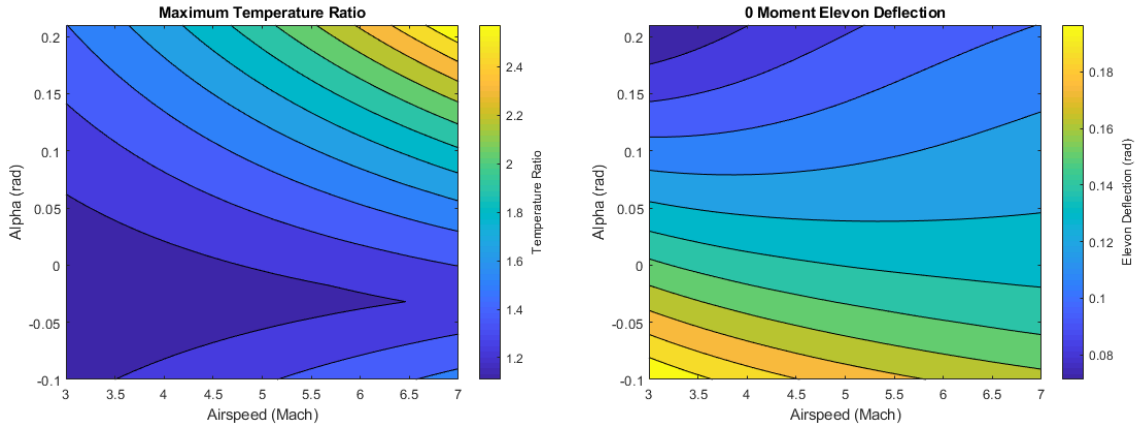

Figure 3.7: Temperature ratio and elevon deflection of the trimmed aircraft

3.5 Testing Approximation Methods in ODE45()

It has been shown that the lookup table and TFC methods are able to approximate the aerodyanmic forces and moment calculated by the panel method. It was expected that this would translate into allowing the approximation methods to speed up the solution time of flight trajectories derived using ODE solvers within MATLAB. This section ensures this is the case by comparing the flight trajectories of dynamics using the panel, lookup table, and TFC methods of determining aerodynamic forces.

Flight trajectories were created for both the full state and zero moment model. For both of these trajectories, each method was used to compute the aerodynamic forces acting on the aircraft. The zero moment model does not use elevon deflection as an input so the 4 input lookup table is not needed. This creates 4 dynamic models following the full state trajectory, and 3 following the zero moment trajectory. The flight path derived by the panel method was assumed to be the most accurate, and the deviation of the other flight paths are recorded as a function of time. The initial conditions for both trajectories are shown below.

Dynamic Model	x (ft)	z (ft)	θ (rad)	α (rad)	v (ft/s)	Q (rad/s)	t_f (s)
Full State	0	6e4	0	0	5e3	0	10
Zero Moment	0	6e4	0	0	5e3	—	120

Table 3.5: Initial states and final time of approximation method verification

The open loop control equations for the full state and zero moment model are calculated as needed within ode45() and respectively are:

$$\begin{aligned} u_{full}(t) &= 0.15 + 0.03\cos\left(\frac{\pi}{5}t\right) \\ Q_{0M}(t) &= 0.01\cos\left(\frac{\pi}{30}t\right) \end{aligned} \tag{3.8}$$

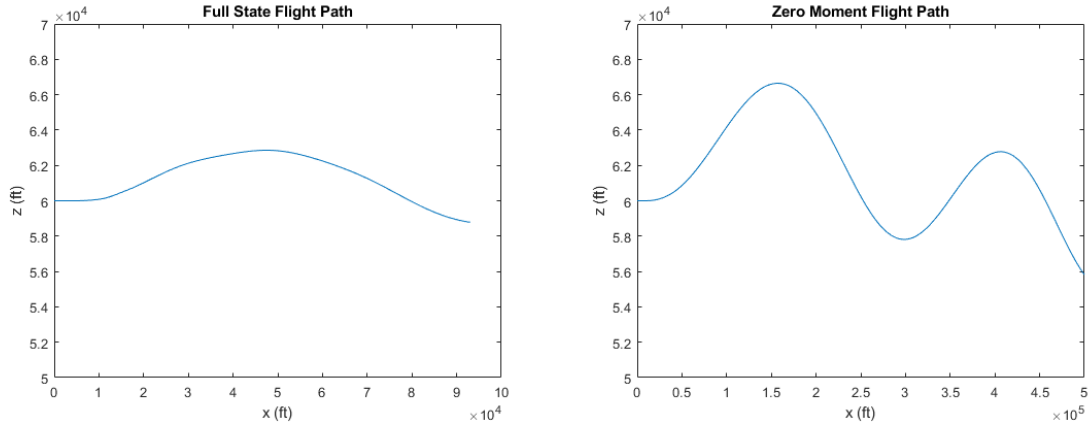

Figure 3.8: Flight paths of the method error trajectories

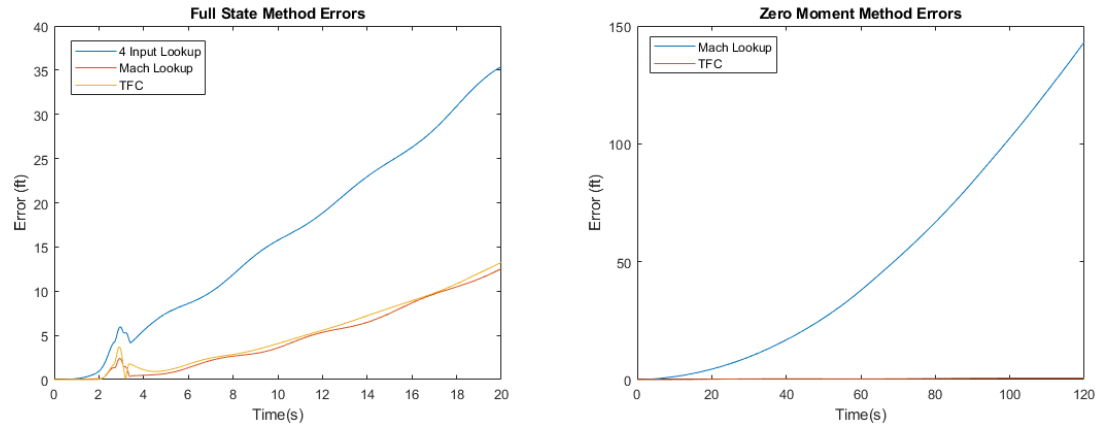

Figure 3.9: Positional Error of the method error trajectories

Method	Panel	4 Input Lookup	Mach Lookup	TFC
Maximum Full State Error (ft)	—	35.4	12.5	13.2
Maximum Zero Moment Error (ft)	—	—	143.2	0.6
Full State Computation Time (s)	3.55	25.84	1.24	0.18
Zero Moment Computation Time (s)	12.70	—	0.32	0.16

Table 3.6: Computation time and maximum positional error of each approximation method]

While its clear that all methods do a reasonable job of predicting the dynamics of the vehicle, several interesting results emerge from this test. For the full state model, the Mach lookup and TFC method return approximately the same positional error as each other. This was unexpected as TFC was shown to be more accurate in approximating aerodynamic forces. In addition, the calculation time for the panel method is lower than the 4 input lookup table. This was also unexpected as per case the 4 input lookup table performed faster than the panel method, but a possible explanation is that the convergence criteria is more difficult to meet for the lookup table, which extends the calculation time.

For the zero moment model, we see very small positional error for the TFC approximated forces. The aircraft deviates less than a foot for a 2 minute flight. In this case, the Mach lookup table also performs similarly to it's full state test. In both cases, the positional error is about 13 feet at the 20 second timestamp. A major advantage of the zero moment method is also shown here, as because the aircraft is assumed to be trimmed the zero moment model is able to calculate a much longer trajectory in less time. This is because the time step size between nodes can be increased without much error.

3.6 Recovering Partial Derivatives with TFC

A major advantage of the TFC method is the ability to quickly compute partial derivatives of parameters it approximates [5]. Calculating aerodynamic forces, elevon deflection, and temperature ratio are already lengthy processes. Finding their partial derivatives with respect to angle of attack, velocity, etc. analytically is beyond the scope of this research. However, it has been shown that the TFC method can approximate these parameters accurately using the same weighting vector used to estimate the function [5].

Consider that the TFC method works by representing each parameter as a linear combination of manifolds. Each manifold is composed of basis functions, which typically are orthogonal polynomials. Polynomials are much simpler to differentiate compared to the other complex iterative functions used up until this point. To recover partial derivatives instead of the value of the parameter itself, all that is needed is to construct new manifolds composed of the partial derivatives of the necessary inputs. The final approximating function is a linear combination of these manifolds, so the partial derivative found on each manifold can simply be weighted and added together. The weightings have already been calculated, so all that is needed is to differentiate the underlying orthogonal polynomials.

While each polynomial can be differentiated as many times as needed, only the first and second partial derivatives are required in the following chapter. The equation for the first and second partial derivative of a Legendre polynomial is.

$$\begin{aligned}
\mathcal{L}_{k+1} &= \frac{2k+1}{k+1} z\mathcal{L}_k - \frac{k}{k+1}\mathcal{L}_{k-1} \qquad \text{where: } \begin{cases} \mathcal{L}_0 = 1 \\ \mathcal{L}_1 = z \end{cases} \\
\mathcal{L}'_{k+1} &= \frac{2k+1}{k+1}\left(\mathcal{L}_k + z\mathcal{L}'_k\right) - \frac{k}{k+1}\mathcal{L}'_{k-1} \qquad \text{where: } \begin{cases} \mathcal{L}'_0 = 0 \\ \mathcal{L}'_1 = 1 \end{cases} \\
\mathcal{L}''_{k+1} &= \frac{2k+1}{k+1}\left(2\mathcal{L}_k + z\mathcal{L}''_k\right) - \frac{k}{k+1}\mathcal{L}''_{k-1} \qquad \text{where: } \begin{cases} \mathcal{L}''_0 = 0 \\ \mathcal{L}''_1 = 0 \end{cases}
\end{aligned} \tag{3.9}$$

The only other consideration needed is that because the Legendre polynomials are mapped to the z-domain of $[-1, +1]$ they must be multiplied by the scaling factor of the considered partial derivatives. Below is an example of a 1st order partial, a 2nd order partial, and a 2nd order mixed partial for a 3 input manifold.

$$\begin{aligned}
\frac{\partial \mathcal{B}_{i,j,k}}{\partial i} &= (c_i)\mathcal{L}'_i(z_i)\mathcal{L}_j(z_j)\mathcal{L}_k(z_k) \\
\frac{\partial^2 \mathcal{B}_{i,j,k}}{\partial i \partial j} &= (c_i c_j)\mathcal{L}'_i(z_i)\mathcal{L}'_j(z_j)\mathcal{L}_k(z_k) \\
\frac{\partial^2 \mathcal{B}_{i,j,k}}{\partial^2 k} &= (c_k^2)\mathcal{L}_i(z_i)\mathcal{L}_j(z_j)\mathcal{L}''_k(z_k)
\end{aligned} \tag{3.10}$$

4. DYNAMIC PROGRAMMING WITH INTERIOR POINTS

With the hypersonic model and approximation methods completed and verified, it was then possible to formulate optimal control problems based within the context of a hypersonic vehicle. The dynamic model of the vehicle was still highly nonlinear even after simplifying the aerodynamic forces and moment with the TFC method and the reduced zero moment model. These nonlinear dynamics cannot be solved using traditional linear OCP solution methods [1]. To solve the nonlinear dynamics, a set of DPIP scripts within MATLAB were used.

4.1 OCP Specific Scripts

DPIP is a very robust OCP solver whose intricacies are beyond the scope of this research [14]. Instead, DPIP functioned as a tool to solve these problems so that the performance of the TFC method can be thoroughly investigated. DPIP operates off of several key functions, a majority of which are not inherent to the specific OCP being solved and have been provided by Dr. Hurtado. However, for each OCP a set of 5 problem-specific functions were written for this research [14]. The structure and function of these scripts are described below.

4.1.1 DPIP Executable

The executable function sets up most of the OCP for DPIP to solve. It defines the number of states and controls, the final time, and lagrange multipliers for equality and inequality constraints. It then simulates the uncontrolled dynamics of the problem before the OCP is solved. After the uncontrolled dynamics are solved, the executable function sets up a loop to iteratively update the control until a locally minimizing solution is found [14].

Within the loop, the main DPIP function is called with the parameters of the OCP as well as the names of the other four OCP specific functions as arguments. The main DPIP function will take the current control history, which should be zero for all time on the first iteration, and calculates new offsets for control for every time step. The new control history is created by adding the control offsets to the previous control history. A weighting parameter *a* which ranges from $(0, 1]$ can scale

the control offsets as well for cases where the problem is highly nonlinear. In which case, *a* can be reduced from it's typical value of 1, but should never be less than or equal to 0 as doing so would cause DPIP to iterate control in the wrong direction or not at all respectively.

For the current iteration of the state trajectory, equality and inequality constraints are collected at each node. For the OCP to be solved completely, all equality constraints must be 0 and all inequality constraints must be less than or equal to 0. The constraints themselves are calculated in the equality and inequality OCP specific functions.

Because DPIP uses an iterative process to calculate the optimal control, it must be given stopping criteria otherwise it will continuously iterate smaller and smaller perturbations in control beyond the accuracy of the model. Stopping criteria for exiting the iterative loop must be defined. A good way for DPIP to automatically detect when it has converged on a solution is to check the norm of the control offset matrix when constraints are met. In this model, there is only one control so the control history will appear as a vector from which the Euclidean norm can be taken. In other OCP where multiple controls exist, control history will be a 2 dimension array. Finding the 2-norm will return the maximum singular value of the control history. In either case, the norm of the control history is guaranteed to be positive and will decrease in value as the control steps become smaller. In most cases, the control step will converge but never reach zero, so a tolerance is needed. The stopping criteria is then defined when the norm of the equality constraints is within a tolerance, there are no positive inequality constraints, and the norm of the control step is within a tolerance. With the stopping criteria defined, the executable function can be run after the remaining 4 functions are written.

4.1.2 DPIP Equations of State

The equations of state function determines the dynamics of the vehicle. It takes arguments of current time, the state vector, and the control vector at that time. Given the arguments, this function returns the derivatives of all of the states with respect to time [14]. For OCP involving the full state dynamics, equation 2.1 is used to calculate state derivatives. For OCP involving the reduced state dynamics of the zero moment model, equation 2.23 is used.

Before aerodynamic forces or any state derivatives are calculated, the atmospheric pressure and speed of sound are interpolated from a standard atmosphere table [7]. Mach number can then be calculated using equation 2.3. Using the TFC method for the full state model, Mach number, angle of attack, and elevon deflection are linearly mapped to the z-domain before using TFC to calculate lift, drag, and pitching moment. In the case of the reduced order model, only Mach number and angle of attack are mapped to the z-domain and then lift and drag of the trimmed aircraft is calculated. Given the aerodynamic forces, the state derivatives can then be calculated.

4.1.3 DPIP Cost Derivative Function

The cost function of the associated OCP is considered by this function. DPIP determines the control step each iteration by quadraticizing the problem and then stepping the control in the direction of cost minimization [14]. To do this, DPIP does not take the cost function into consideration directly, but rather considers the partial derivative of the cost function with respect to states and control. The value of the cost function itself is irrelevant so long as it is optimized to a minima.

The cost derivative function takes the arguments of the state and control history as well as the time step and calculates the partials of the cost function at the current node. To fully represent the 2nd order derivatives of the problem, the following derivatives are necessary: 1st partial with respect to states, 1st partial with respect to controls, 2nd mixed partial with respect to states, 2nd mixed partial with respect to states and control, and 2nd mixed partial with respect to controls. All of this information is collected into 5 matrices of the following form.

$$
\begin{aligned}
\frac{\partial J}{\partial \boldsymbol{x}} &= \begin{bmatrix} \frac{\partial J}{\partial \boldsymbol{x}_1} \\ \frac{\partial J}{\partial \boldsymbol{x}_2} \\ \vdots \\ \frac{\partial J}{\partial \boldsymbol{x}_n} \end{bmatrix} & \frac{\partial J}{\partial \boldsymbol{u}} &= \begin{bmatrix} \frac{\partial J}{\partial \boldsymbol{u}_1} \\ \frac{\partial J}{\partial \boldsymbol{u}_2} \\ \vdots \\ \frac{\partial J}{\partial \boldsymbol{u}_m} \end{bmatrix} & \frac{\partial^2 J}{\partial \boldsymbol{x}^2} &= \begin{bmatrix} \frac{\partial^2 J}{\partial \boldsymbol{x}_1^2} & \frac{\partial^2 J}{\partial \boldsymbol{x}_1 \boldsymbol{x}_2} & \cdots & \frac{\partial^2 J}{\partial \boldsymbol{x}_1 \boldsymbol{x}_n} \\ \frac{\partial^2 J}{\partial \boldsymbol{x}_2 \boldsymbol{x}_1} & \frac{\partial^2 J}{\partial \boldsymbol{x}_2^2} & \cdots & \frac{\partial^2 J}{\partial \boldsymbol{x}_2 \boldsymbol{x}_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 J}{\partial \boldsymbol{x}_n \boldsymbol{x}_1} & \frac{\partial^2 J}{\partial \boldsymbol{x}_n \boldsymbol{x}_2} & \cdots & \frac{\partial^2 J}{\partial \boldsymbol{x}_n^2} \end{bmatrix} \\
\frac{\partial^2 J}{\partial \boldsymbol{u}^2} &= \begin{bmatrix} \frac{\partial^2 J}{\partial \boldsymbol{u}_1^2} & \frac{\partial^2 J}{\partial \boldsymbol{u}_1 \boldsymbol{u}_2} & \cdots & \frac{\partial^2 J}{\partial \boldsymbol{u}_1 \boldsymbol{u}_m} \\ \frac{\partial^2 J}{\partial \boldsymbol{u}_2 \boldsymbol{u}_1} & \frac{\partial^2 J}{\partial \boldsymbol{u}_2^2} & \cdots & \frac{\partial^2 J}{\partial \boldsymbol{u}_2 \boldsymbol{u}_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 J}{\partial \boldsymbol{u}_m \boldsymbol{u}_1} & \frac{\partial^2 J}{\partial \boldsymbol{u}_m \boldsymbol{u}_2} & \cdots & \frac{\partial^2 J}{\partial \boldsymbol{u}_m^2} \end{bmatrix} & \frac{\partial^2 J}{\partial \boldsymbol{x} \boldsymbol{u}} &= \begin{bmatrix} \frac{\partial^2 J}{\partial \boldsymbol{x}_1 \boldsymbol{u}_1} & \frac{\partial^2 J}{\partial \boldsymbol{x}_1 \boldsymbol{u}_2} & \cdots & \frac{\partial^2 J}{\partial \boldsymbol{x}_1 \boldsymbol{u}_m} \\ \frac{\partial^2 J}{\partial \boldsymbol{x}_2 \boldsymbol{u}_1} & \frac{\partial^2 J}{\partial \boldsymbol{x}_2 \boldsymbol{u}_2} & \cdots & \frac{\partial^2 J}{\partial \boldsymbol{x}_2 \boldsymbol{u}_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 J}{\partial \boldsymbol{x}_n \boldsymbol{u}_1} & \frac{\partial^2 J}{\partial \boldsymbol{x}_n \boldsymbol{u}_2} & \cdots & \frac{\partial^2 J}{\partial \boldsymbol{x}_n \boldsymbol{u}_m} \end{bmatrix}
\end{aligned}
\tag{4.1}
$$

4.1.4 DPIP Equality and Inequality Constraints

The final two OCP specific functions describe the constraints on the problem. The first function calculates the equality constraints, while the second calculates the inequality constraints. DPIP will iterate the control history to ensure the constraints are not violated. The equality constraints are formatted such that when evaluated, the result is zero when the constraint is met. For example, an equality constraint on the initial altitude of the aircraft may look like:

$$
C = z_0 - z_{init} \tag{4.2}
$$

Inequality constraints are formatted so that they evaluate to less than or equal to zero when the constraint is met. A minimum or maximum constraint on the altitude of the aircraft may respectively look like:

$$
\begin{aligned}
C_i &= -z_i + z_{min} \\
C_i &= z_i - z_{max}
\end{aligned}
\tag{4.3}
$$

For both equality and inequality constraints, the constraint equations are evaluated and stored

in a vector $\boldsymbol{C}$

$$\boldsymbol{C} = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_{N_C} \end{bmatrix} \tag{4.4}$$

Much like the cost derivative function, DPIP also requires the derivative of constraints with respect to states and control. Both 1st and 2nd order partial derivatives are needed. The constraint partial derivative arrays are built in a similar way as the derivative arrays for the cost function. The main difference is that each row of the array corresponds to a unique constraint and each column corresponds to a state or a control. For 2nd order derivatives each row is expanded to multiple rows, one for each of the 2nd derivative taken. When constructed the constraint partial derivative arrays have the following form.

$$\begin{aligned}
\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{x}} &= \begin{bmatrix} \frac{\partial C_1}{\partial \boldsymbol{x}_1} & \frac{\partial C_1}{\partial \boldsymbol{x}_2} & \cdots & \frac{\partial C_1}{\partial \boldsymbol{x}_n} \\ \frac{\partial C_2}{\partial \boldsymbol{x}_1} & \frac{\partial C_2}{\partial \boldsymbol{x}_2} & \cdots & \frac{\partial C_2}{\partial \boldsymbol{x}_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial C_{N_C}}{\partial \boldsymbol{x}_1} & \frac{\partial C_{N_C}}{\partial \boldsymbol{x}_2} & \cdots & \frac{\partial C_{N_C}}{\partial \boldsymbol{x}_n} \end{bmatrix} \qquad \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{u}} = \begin{bmatrix} \frac{\partial C_1}{\partial \boldsymbol{u}_1} & \frac{\partial C_1}{\partial \boldsymbol{u}_2} & \cdots & \frac{\partial C_1}{\partial \boldsymbol{u}_m} \\ \frac{\partial C_2}{\partial \boldsymbol{u}_1} & \frac{\partial C_2}{\partial \boldsymbol{u}_2} & \cdots & \frac{\partial C_2}{\partial \boldsymbol{u}_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial C_{N_C}}{\partial \boldsymbol{u}_1} & \frac{\partial C_{N_C}}{\partial \boldsymbol{u}_2} & \cdots & \frac{\partial C_{N_C}}{\partial \boldsymbol{u}_m} \end{bmatrix} \\
\frac{\partial^2 \boldsymbol{C}}{\partial \boldsymbol{x}^2} &= \begin{bmatrix} \left[\frac{\partial^2 C_1}{\partial \boldsymbol{x}^2}\right] \\ \left[\frac{\partial^2 C_2}{\partial \boldsymbol{x}^2}\right] \\ \vdots \\ \left[\frac{\partial^2 C_{N_C}}{\partial \boldsymbol{x}^2}\right] \end{bmatrix} \qquad \frac{\partial^2 \boldsymbol{C}}{\partial \boldsymbol{u}^2} = \begin{bmatrix} \left[\frac{\partial^2 C_1}{\partial \boldsymbol{u}^2}\right] \\ \left[\frac{\partial^2 C_2}{\partial \boldsymbol{u}^2}\right] \\ \vdots \\ \left[\frac{\partial^2 C_{N_C}}{\partial \boldsymbol{u}^2}\right] \end{bmatrix} \qquad \frac{\partial^2 \boldsymbol{C}}{\partial \boldsymbol{x} \partial \boldsymbol{u}} = \begin{bmatrix} \left[\frac{\partial^2 C_1}{\partial \boldsymbol{x} \partial \boldsymbol{u}}\right] \\ \left[\frac{\partial^2 C_2}{\partial \boldsymbol{x} \partial \boldsymbol{u}}\right] \\ \vdots \\ \left[\frac{\partial^2 C_{N_C}}{\partial \boldsymbol{x} \partial \boldsymbol{u}}\right] \end{bmatrix}
\end{aligned} \tag{4.5}$$

Fortunately, most of the relevant constraints are on the states and controls themselves. This results in the 1st order partial derivatives for equality constraints being 1 for the relevant state and zero for all others. For inequality constraints, the 1st order partial derivatives are -1 for minimum constraints and $+1$ for maximum constraints. All 2nd order partials are zero in these cases.

Calculating partial derivatives for non-states and non-controls can be more difficult. In the case of elevon constraints for the zero moment model, the elevon can be prevented from exceeding it's upper and lower limit of 0.21 radians of deflection by enforcing the restricted flight domain angle of attack from -0.10 to 0.21 radians as found in section 3.4.1. Another way to enforce these constraints is to use the partial derivatives recovered by TFC as in section 3.6. Given a $\boldsymbol{\xi}$ vector trained to approximate a parameter such as maximum temperature, the $\boldsymbol{\xi}$ vector can be used both to calculate the temperature in determining if the constraint is violated or not and also it's derivative with respect to the states [5]. This is extremely convenient as without the TFC method, the partial derivative of maximum temperature with respect to a state such as angle of attack would be difficult to calculate.

4.2 Solving OCP with DPIP

To test the effectiveness of the TFC methods with DPIP, three OCP have been formulated to test some aspect of solving OCP. The first OCP involved stabilizing the aircraft about a set angle of attack using the full state model, testing the minimization of a cost function. The second OCP involved maneuvering the aircraft around set "no fly" zones, testing active inequality constraints placed on states using the zero moment dynamic model. The final OCP involved ensuring the aircraft does not exceed strict temperature constraints, testing active inequality constraints placed on non-states with the zero moment dynamic model. The initial state constraints and final time for each OCP are shown in the table below.

OCP	x (ft)	z (ft)	θ (rad)	α (rad)	v (ft/s)	Q (rad/s)	t_f (s)
1	0	6e4	0	0.00	5e3	0	10
2	0	6e4	0	0.00	5e3	—	120
3	0	6e4	0	0.00	5e3	—	120

Table 4.1: Initial states and final time each OCP

4.2.1 Stabilize Angle of Attack OCP

The following OCP was formulated to be solved with the full state system. The goal was to stabilize the aircraft about a set angle of attack of 0.045 radians. The DPIP simulation uses 51 nodes spaced 0.2 seconds apart for a 10 second long OCP. Because the full state model is highly nonlinear, the control step scaling between iterations was reduced to 0.6. In addition, the stopping criteria for this problem was for the norm of the control step to be less than or equal to 0.01. The cost function for this problem is defined as:

$$J = 1\text{E}6 * \frac{1}{2} \int_0^{10} (\alpha - 0.025) \, \mathrm{d}t \tag{4.6}$$

The initial test for this full state model was unsuccessful. With the initial elevon deflection assumed to be zero, the aircraft pitches up beyond it's limit during the first iteration of DPIP. As this happens, DPIP loses ability solve for an adequate stabilizing control, it returns errors within the main DPIP function that matrices used to solve for the new control step are singular. DPIP cannot solve this problem without other considerations.

The main reason DPIP fails to solve the full state model is because the initial condition already results in a stalled aircraft. To account for this, the initial control of the aircraft was changed such that the aircraft trims at a reasonable angle of attack before control was iterated by DPIP. Using figure 3.5, the elevon deflection required to trim the aircraft at the initial flight condition is roughly 0.13 radians. The OCP was then attempted again with the nonzero initial control history.

These results are much better than the zero initial control case and clearly show that DPIP is capable of solving this OCP. This problem was solved after 9 iterations of DPIP in 32.2 seconds. The cost function after each iteration of DPIP decreases, and it is clear that the optimized angle of attack history tightens closely about the desired 0.025 radians.

However, there are clear limitations to this problem. For one, only 10 seconds of flight time are accounted for in this problem and it would be infeasible to greatly extend the length of the time period due to increased computation time. Also, the control had to be seeded with an initial

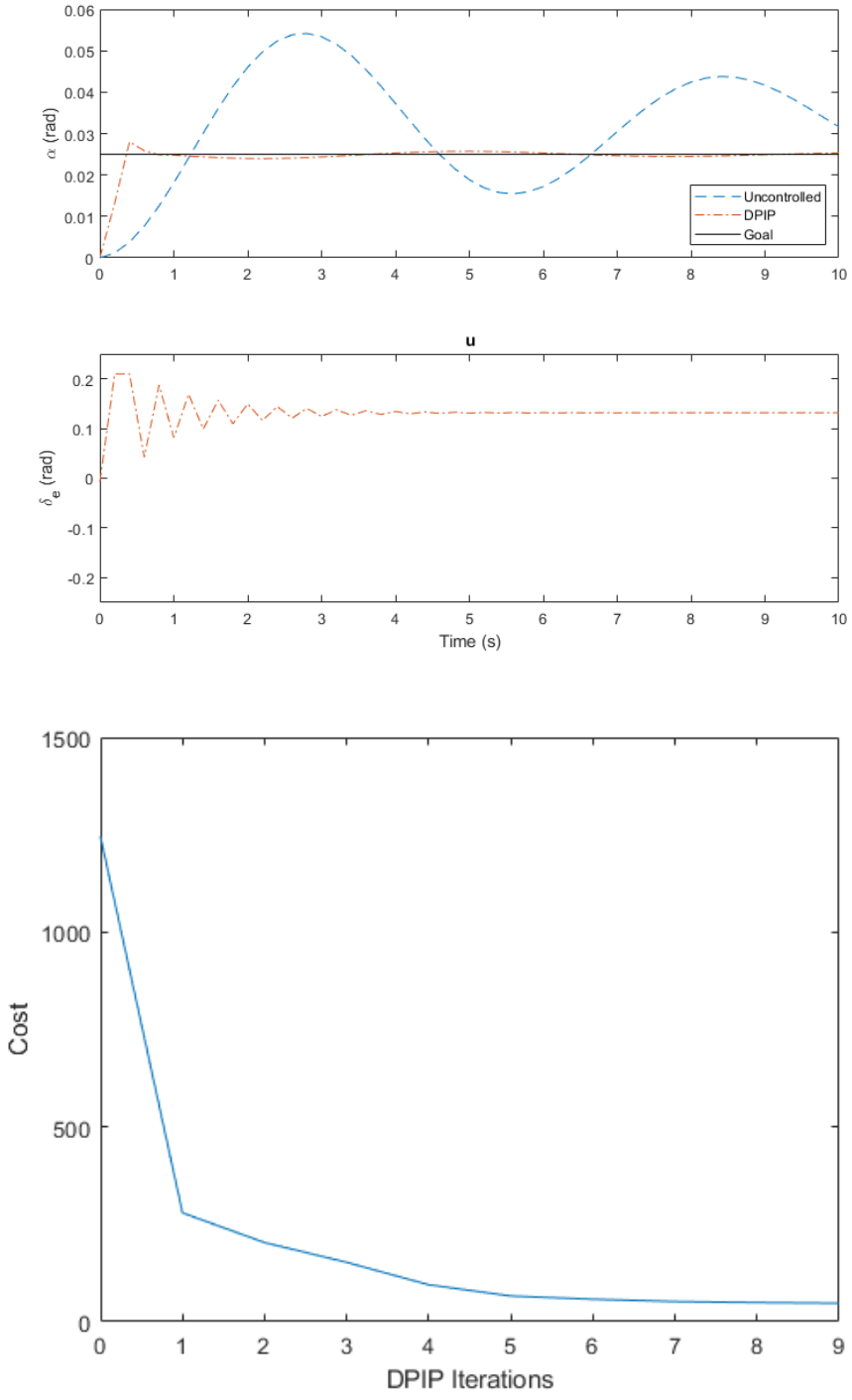

Figure 4.1: Flight trajectory of the uncontrolled and optimized flight path for OCP 1

calculated value, essentially giving most of the "answer" to DPIP as it was not able to find an optimal trim condition on it's own.

4.2.2 Avoid No Fly Zone OCP

The following OCP was to be solved with the zero moment reduced state dynamics. The goal was to maneuver the aircraft around strict "no fly" zones. The uncontrolled dynamics of the aircraft caused it to fly directly through both no fly zones, meaning it is up to DPIP to find a solution where the aircraft avoids these zones. The OCP uses 121 nodes spaced 1 second apart each for a 120 second long OCP. The control step scaling between iterations is kept at 1. The stopping criteria for the optimal solution requires that all inequality constraints are met and the norm of the control step is less than or equal to 0.001. The active no fly altitude constraints are defined as:

$$C = \begin{cases} -z + 7\text{E}4, & \text{for } 1.5e5 \leq x \leq 2.0e5 \\ z - 5\text{E}4, & \text{for } 4.0e5 \leq x \leq 4.5e5 \end{cases} \tag{4.7}$$

The cost function for this problem is:

$$J = 1\text{E}2\frac{1}{2}\int_0^{120} Q \, \mathrm{d}t \tag{4.8}$$

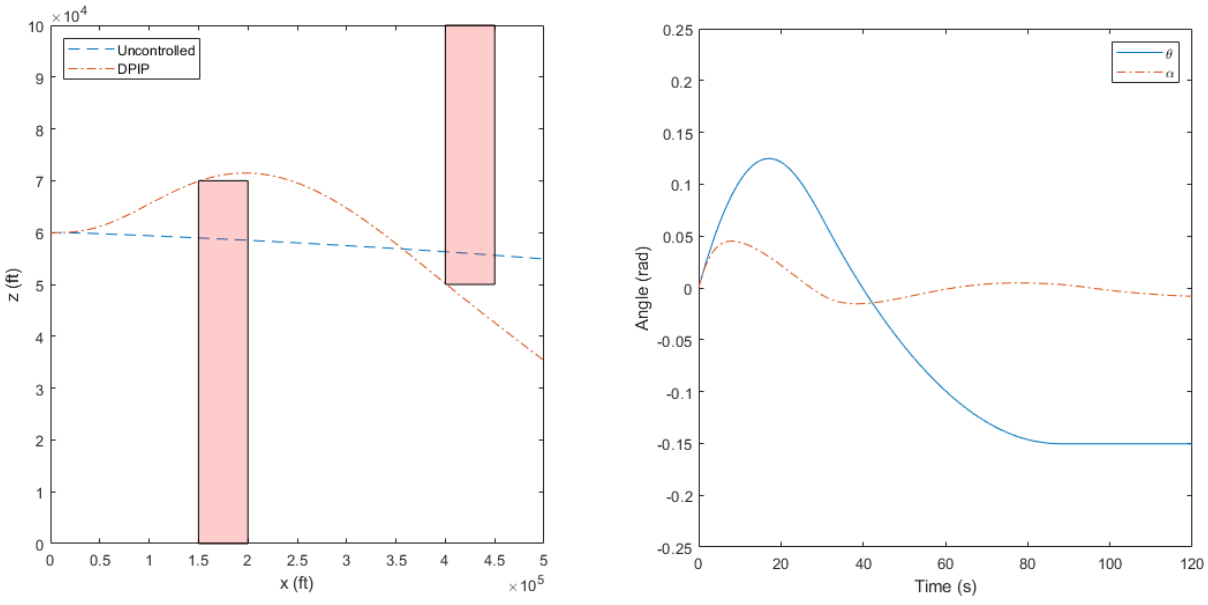

Figure 4.2: Flight trajectory and angles of the uncontrolled and optimized flight path for OCP 2

These results are also very promising. DPIP successfully solved the OCP in only 4 iterations and in 27.8 seconds. The aircraft rises and dives around the no fly zones easily. The psuedocontrol of pitch rate is the only term that is penalized, and it is kept very low and the aircraft pitches smoothly, unlike the erratic behavior of the solved control in the previous OCP. Elevon deflection was also able to be recovered after the solution process. Using the state histories and a $\boldsymbol{\xi}$ vector trained for elevon deflection, the true control can easily be recovered from the aircraft.

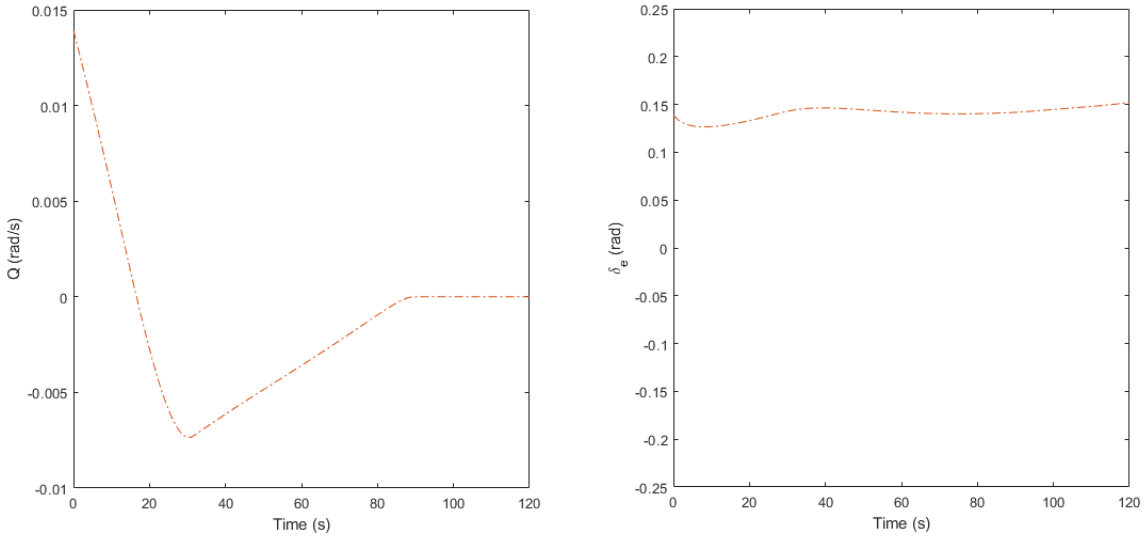

Figure 4.3: Psuedo-controlled pitch rate and calculated elevon deflection for OCP 2

4.2.3 Temperature Limited OCP

The following OCP was to be solved with the zero moment reduced state dynamics. The goal was to meet strict temperature constraints on the forward surfaces of the aircraft. To ensure the temperature constraints are active, first the uncontrolled aircraft is simulated and temperature is measured as a function of time. Then a maximum temperature constraint is imposed on the aircraft such that DPIP must account for the constraint. Otherwise the number of nodes, step scaling, stopping criteria and cost function are the same as the previous OCP. The temperature constraint is defined as:

$$C = T - 530 \tag{4.9}$$

The partial derivatives of the maximum temperature are derived using the TFC method as described in section 3.6. DPIP solves the problem by pitching the aircraft down to reduce the strength of the oblique shock waves which would otherwise increase temperature.

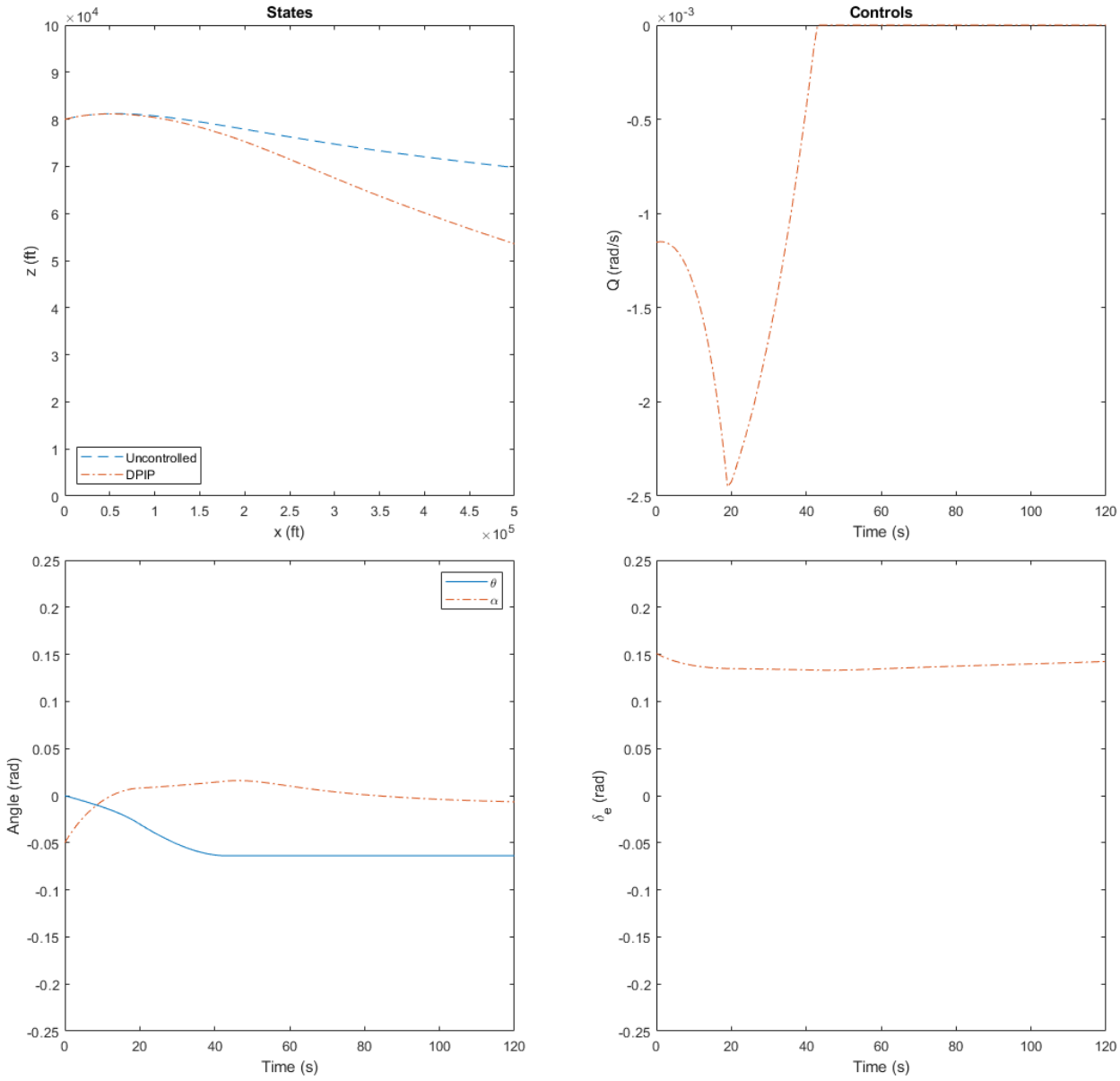

Figure 4.4: Flight trajectory of the uncontrolled and optimized flight path for OCP 3

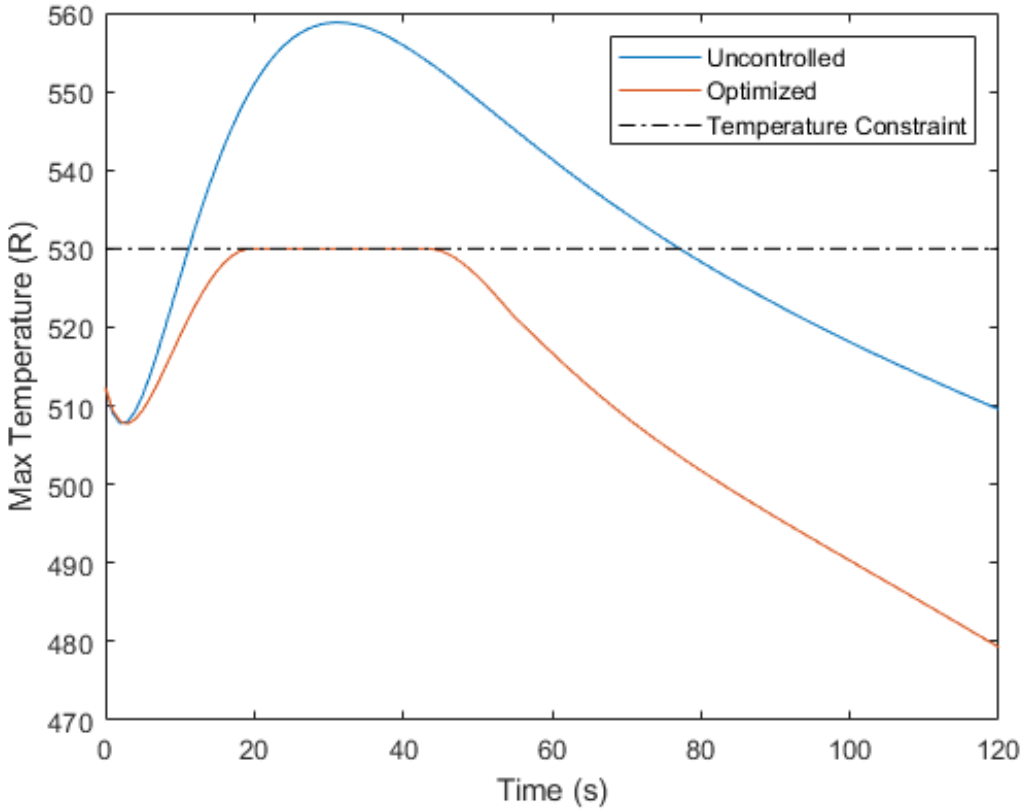

Figure 4.5: Temperature of the leading surfaces for the uncontrolled and optimized OCP 3

4.3 Comparing TFC Results to Direct Results

One of the goasl of this research is to find new ways to simplify the dynamics of hypersonic vehicles such that OCP can be solved faster. With this in mind, the results of the OCP solved using dynamics simplified by TFC were compared to solutions using the direct method of calculating aerodynamic forces using the panel method. OCP 1 and 2 from subsections 4.2.1 and 4.2.2 respectively are considered. The temperature constrained OCP 3 cannot be tested using the direct method because the temperature constraints rely on partial derivatives of the maximum temperature with respect to states and control. As mentioned before, analytically solving these partial derivatives is outside the scope of this research. Instead, consider that OCP 3 could not be solved easily without the use of TFC which speaks to it's usefulness of the method.

In the table below, the computation time and final cost function of the solved problems are compared.

	OCP 1	OCP2	OCP3 (rad)
Direct Time (s)	7363	8417	—
TFC Time (s)	33	27	40
Direct Cost	50.79	0.1258	—
TFC Cost	47.45	0.1258	0.0054

Table 4.2: Solved OCP Solution Time and Final Cost

From the tabulated results, it is clear that using TFC offers a huge advantage in terms of decreasing computation time compared to calculating the aerodynamic forces directly. For both OCP 1 and 2, the DPIP program using TFC finds a solution which is lower or equal in cost in far less time.

5. SUMMARY AND CONCLUSIONS

In this research, a hypersonic model was developed to approximate the dynamics of a longitudinal hypersonic vehicle. Using a panel method and Pradtl-Meyer wave theory, aerodynamic forces and moment were calculated for the aircraft. These results were then verified against a CFD model created in SOLIDWORKS Flow Simulator. The panel method was then used to simulate the dynamics of the hypersonic vehicle in MATLAB. A reduced order model was also created by assuming that the aircraft is trimmed at any given point in the flight. The resulting flight trajectories of this model were then verified by deriving the elevon deflection required to trim the aircraft and running the full state model with the elevon deflection history. A strong match occurred when the aircraft was statically stable but not when the aircraft was statically unstable. To address this, a small feedback loop was placed on the angle of attack of the aircraft using the state history of the zero moment model. This stabilized the aircraft. A temperature model was also created to record the maximum temperature on the front panels of the aircraft. This temperature model was also verified using CFD.

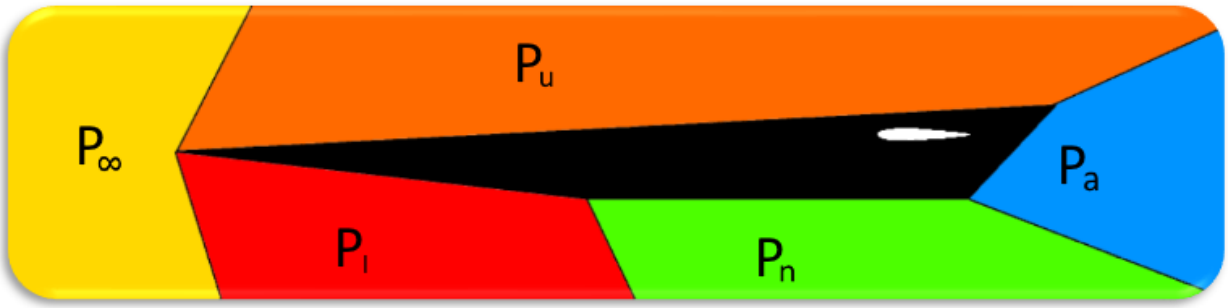

Figure 5.1: Pressure acting on each panel of the aircraft

Calculating the aerodynamic forces and moment of the aircraft was shown to be a computationally expensive process. Methods to estimate these forces to improve computation time were investigated. The first method utilized a lookup tables which interpolated lift, drag, and pitching moment from a large array using altitude, velocity, angle of attack, and elevon deflection. This lookup table was later reduced in size by normalizing the atmospheric pressure to sea level, nondimensionallizing the velocity to Mach number, and later multiplying the calculated forces and moment by the ratio between the atmospheric pressure of the freestream and sea level. This resulted in a reduction in the size of the lookup table as well as an increase in the accuracy of the estimation. Finally, a new method to estimate aerodynamic forces using the Theory of Functional Connections was developed. Aerodynamic forces and moment, as well as other parameters were approximated as a linear combination of orthogonal manifolds whose weightings were solved by a least squares algorithm. Ultimately, each of the methods was tested for a large sample of varied flight conditions against the direct panel method. The error and computation time of each method was recorded, with the TFC method outperforming all other methods by a large margin.

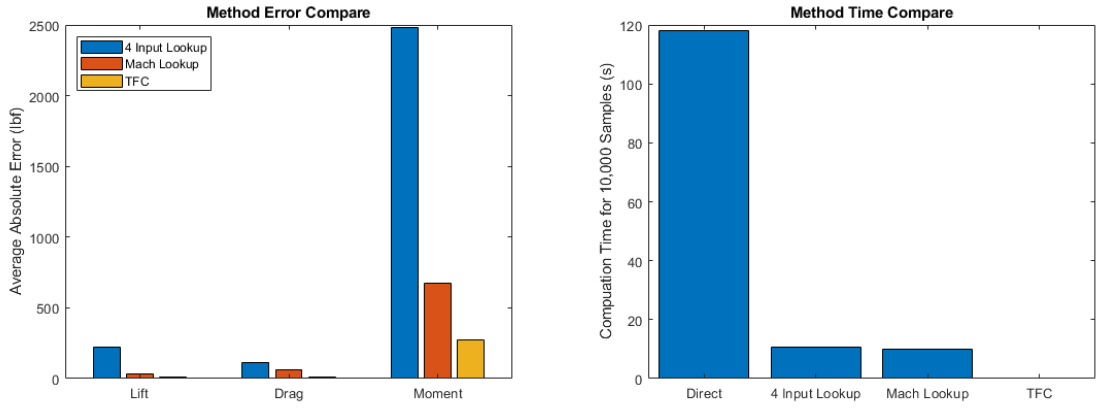

Figure 5.2: Performance of force approximation methods

The TFC method was then used to approximate aerodynamic forces, moment, elevon deflection, and temperature constraints for three optimal control problems. The OCP were solved using DPIP, and the resulting trajectories were compared. DPIP successfully solved OCP using the full state model and the zero moment model. However, the full state model needed to be initialized with a calculated trim condition in order to converge. Otherwise, the TFC derived forces and constraints cooperated with DPIP and resulting in a nonlinear OCP solver that could quickly solve complex hypersonic dynamics with strict constraints.

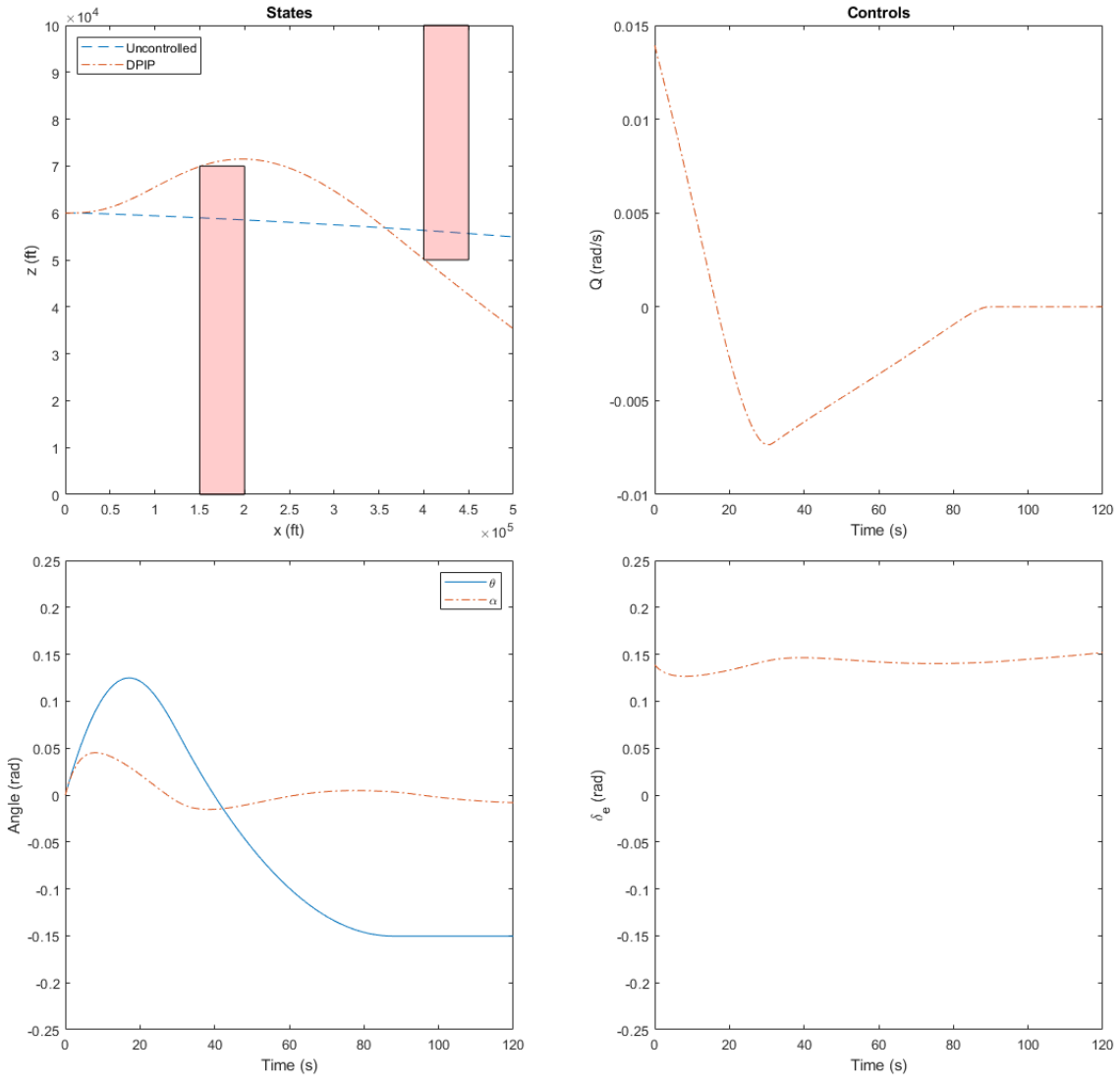

Figure 5.3: Flight trajectory of the uncontrolled and optimized flight path for OCP 2

REFERENCES

[1] F. Lewis, D. Vrabie, and V. Syrmos, *Optimal Control*. Hoboken, NJ, USA: Wiley, 3rd ed., 2012.

[2] M. Bolender and D. Doman, "Nonlinear longitudinal dynamical model of an air-breathing hypersonic vehicle," *Journal of Spacecraft and Rockets*, vol. 44, no. 2, pp. 374–387, 2007.

[3] G. Rigatos, P. Wira, M. Abbaszadeh, K. Busawon, and L. Dala, "Nonlinear optimal control for autonomous hypersonic vehicles," *Aerospace Systems*, vol. 2, pp. 197–213, 11 2019.

[4] E. Rollins, J. Valasek, J. A. Muse, and M. A. Bolender, "Nonlinear adaptive dynamic inversion applied to a generic hypersonic vehicle," in *AIAA Guidance, Navigation, and Control (GNC) Conference*, 2013.

[5] D. Mortari, H. Johnston, and C. Leake, "The theory of functional connections." Forthcoming book, 2021.

[6] K. Groves, D. Sigthorsson, A. Serrani, S. Yurkovich, M. Bolender, and D. Doman, "Reference command tracking for a linearized model of an air-breathing hypersonic vehicle," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2005.

[7] J. Mattingly and K. Boyer, *Elements of Propulsion: Gas Turbines and Rockets*. Reston, VA, USA: American Institute of Aeronautics and Astronautics, 2nd ed., 2016.

[8] MathWorks, "Solve system of nonlinear equations." Available at `https://www.mathworks.com/help/optim/ug/fsolve.html`.

[9] D. Raymer, *Aircraft Design: A Conceptual Approach*. Reston, VA, USA: American Institute of Aeronautics and Astronautics, 5th ed., 2012.

[10] MathWorks, "Solve nonstiff differential equations." Available at `https://www.mathworks.com/help/matlab/ref/ode45.html`.

[11] W. Chao, L. Xinyu, and L. Feng, "Six-dof modeling and simulation for generic hypersonic vehicle in reentry phase," *Procedia Engineering*, vol. 99, pp. 600–606, 2015.

[12] MathWorks, "Reshape array." Available at `https://www.mathworks.com/help/matlab/ref/reshape.html`.

[13] MathWorks, "Solve systems of linear equations ax = b for x." Available at `https://www.mathworks.com/help/matlab/ref/mldivide.html`.

[14] R. Robinett, D. Wilson, R. Eisler, and J. Hurtado, *Applied Dynamic Programming for Optimization of Dynamical Systems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2005.

[15] K. Kowalski and W. Steeb, *Nonlinear Dynamical Systems and Carleman Lineaization*. Singapore: World Scientific, 1991.

APPENDIX

CARLEMAN LINEARIZATION

At the beginning of this research, a method referred to as Carleman linearization was studied as a means to simplify the dynamics of nonlinear systems. The results were unsuccessful, but have been included in this appendix for the sake of posterity.

Carleman linearization, or Carleman embedding, offers a way to expand a finite series of nonlinear equations into an infinite series of linear equations [15]. After performing this linearization, the infinite series of linear equations can be used to calculate the behavior of the nonlinear system exactly. This is particularly useful to the field of optimal control because, unlike a nonlinear control system, linear control systems can be solved directly and without iteration [1].

However, it is not feasible to attempt to find the solution to the algebraic Ricatti equation or otherwise solve an OCP for an infinite series of linear equations. In expanding a nonlinear system into an infinite system, solution is no longer possible. At this point, truncating the infinite system to a finite number of linear equations was considered. For example, a single nonlinear equation could be expanded into an infinite series of linear equations which exactly calculate the behavior of the nonlinear system. Then, the infinite series of equations is reduced to the first 10 equations and all higher order terms and equations are omitted. Now the nonlinear equation, which requires iteration to solve for an OCP, is approximated by a series of 10 linear equations, which can be solved using a linear method such as LQR or through 1 iteration of a nonlinear OCP solver such as DPIP [14].

Carleman Embedding works by creating lifted states based on a known basis, a convenient basis being a power series. The state could then be recovered through the first lifted state, which is equal to the actual state. An example of Carleman linearization being applied to an uncontrolled scalar system is shown below.

$$\text{Let } y_n \equiv x^n$$

$$\dot{x} = x^2 \qquad \rightarrow \qquad \begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 2 & 0 & \dots \\ 0 & 0 & 0 & 3 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \end{bmatrix} \tag{A.1}$$

Now that Carleman Embedding had been applied to uncontrolled systems it was necessary to expand the process to controlled systems. However, the presence of a control term made it difficult to completely linearize the system, so several different approaches were tested. For each approach, the same controlled scalar system was considered.

$$\dot{x} = x + \epsilon x^2 + u \tag{A.2}$$

Approach 1 involved directly applying the Carleman linearization process to the system with no special considerations. This usually resulted in a bilinear system as the control's effect on the lifted states would be affected by other lifted states. An example of Approach 1 is shown below.

$$\text{Let } y_n \equiv x^n$$

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & \epsilon & 0 & 0 & \dots \\ 0 & 2 & 2\epsilon & 0 & \dots \\ 0 & 0 & 3 & 3 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \end{bmatrix} + \begin{bmatrix} 1 \\ 2y_2 \\ 3y_3 \\ \vdots \end{bmatrix} u \tag{A.3}$$

$$\text{Form:} \qquad \dot{\boldsymbol{y}} = A\boldsymbol{y} + B(\boldsymbol{y})u$$

Approach 2 was developed as a method to remove the nonlinearities of the bilinear control solution by defining new lifted controls that were an explicit function of the lifted state and the original control. This created a linear system with the caveat that only the original control is

available for control design. An example of Approach 2 is shown below.

$$\text{Let } y_n \equiv x^n \quad \text{and} \quad u_n = y_{n-1}u$$

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & \epsilon & 0 & 0 & \dots \\ 0 & 2 & 2\epsilon & 0 & \dots \\ 0 & 0 & 3 & 3\epsilon & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & \dots \\ 0 & 2 & 0 & \dots \\ 0 & 0 & 3 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} u \\ u_2 \\ u_3 \\ \vdots \end{bmatrix} \tag{A.4}$$

$$\text{Form:} \qquad \dot{\boldsymbol{y}} = A\boldsymbol{y} + B\boldsymbol{u}$$

Finally, Approach 3 was developed as a method to remove the control matrix all together and instead compound the effects of control into the lifted state matrix by assuming a forced feedback form where the control is defined by a scalar factor of the state itself.

$$\begin{aligned} \dot{x} &= x + \epsilon x^2 + u \\ \dot{x} &= (1+k)x + \epsilon x^2 \qquad \text{where:} \quad u \equiv kx \\ \dot{x} &= k'x + \epsilon x^2 \qquad\quad \text{where:} \quad k' \equiv 1 + k \end{aligned} \tag{A.5}$$

An example of Approach 3 is shown below.

$$\text{Let } y_n \equiv x^n$$

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} k' & \epsilon & 0 & 0 & \dots \\ 0 & 2k' & 2\epsilon & 0 & \dots \\ 0 & 0 & 3k' & 3\epsilon & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \end{bmatrix} \tag{A.6}$$

$$\text{Form:} \qquad \dot{\boldsymbol{y}} = A(k')\boldsymbol{y}$$

To test each approach, a scalar OCP was defined and DPIP was used to solve the original system. The cost function and initial and final state conditions for the problem are defined as.

$$J = \int_0^1 x^2 + u^2 dt \qquad \text{where: } \begin{cases} x(0) = 1 \\ x(1) = 0.1 \end{cases} \tag{A.7}$$

For the Carleman linearized systems, the first lifted state y_1 is penalized instead of x. In the case of Approach 3, the cost function is rewritten to account for the k' term.

$$J = \int_0^1 x^2 + (k' - 1)^2 dt \tag{A.8}$$

The optimal state and control trajectories were recorded as the true solution to the problem and the time to converge on a solution was also recorded. Then the problem was solved for a Carleman Embedded system derived from each of the three approaches where the first five states were kept. Again, the optimal state and control trajectories were recorded for each method as well as the time to converge. A close solution to the original problem was found, however every lifted system took longer to solve than the original nonlinear equation.

Furthermore, when the process was expanded to a multistate OCP, it was found that increasing the number of states required an increasing number of embedded equations to reach the same order as a scalar system. While this result of the curse of dimensionality was expected, what was not expected was that the computation time for solving the Carleman systems also exploded and was significantly larger than just solving the nonlinear system. With this current implementation, Carleman Embedding was not effective as a method of rapid trajectory because it slowed computation time while also being less accurate than solving the system directly.